NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

COMPUTER-AIDED SOFTWARE EVOLUTION BASED ON INFERRED DEPENDENCIES

by

Meng-Chyi Harn

December 1999

Thesis Advisor: **Valdis Berzins**

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 4

20000203 035

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
December 1999

3. REPORT TYPE AND DATES COVERED
Ph.D. Dissertation

4. TITLE AND SUBTITLE
COMPUTER-AIDED SOFTWARE EVOLUTION BASED ON INFERRED DEPENDENCIES

5. FUNDING NUMBERS

6. AUTHOR(S)
Harn, Meng-Chyi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The major problem addressed by this research is how to automate parts of software evolution using dependency rules, especially for large and complex real-time embedded systems. The main topics of this study are the development of a Relational Hypergraph model (RH model) and the design of a Computer-Aided Software Evolution System (CASES). The goals of this dissertation are to explore the existing issues, to formalize software evolution, to reuse software evolution components, and to build a dependency-computing model. We have resolved parts of essential software evolution issues in the following categories: software evolution process, software evolution traceability, software evolution description, software evolution management, and software evolution control.

The RH model can realize automated software evolution in multi-dimensional phases, such as software prototype or product demo, issue analysis, requirement analysis, specification design, module implementation, program integration, and software product implementation. Many types of software evolution objects in each phase, and dependencies among these objects have been defined to describe software evolution processes.

14. SUBJECT TERMS
Software evolution, Hypergraph model, Dependency rules, Software reuse, Software evolution objects, Software evolution processes

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

COMPUTER-AIDED SOFTWARE EVOLUTION BASED ON INFERRED DEPENDENCIES

by

Meng-Chyi Harn
LTC, Taiwan, R.O.C. Army
B.S., Chung-Cheng Institute of Technology, Taoyuan, Taiwan, 1980
M.B.A., National Defense Management College, Taipei, Taiwan, 1986

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1999

Author: Mengchyi Harn
Meng-Chyi Harn

Approved By:

Dan Boger
Professor of Computer Science

Luqi
Professor of Computer Science

Xiaoping Yun
Associate Professor of Electrical and Computer Engineering

Craig Rasmussen
Associate Professor of Mathematics

Valdis Berzins
Department of Computer Science
Dissertation Supervisor

Approved by: Dan Boger
Dan Boger, Chairman, Department of Computer Science

Approved by: Anthony P. Ciavarelli
Anthony P. Ciavarelli, Associate Provost for Instruction

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The major problem addressed by this research is how to automate parts of software evolution using dependency rules, especially for large and complex real-time embedded systems. The main topics of this study are the development of a Relational Hypergraph model (RH model) and the design of a Computer-Aided Software Evolution System (CASES). The goals of this dissertation are to explore the existing issues, to formalize software evolution, to reuse software evolution components, and to build a dependency-computing model.

We have resolved parts of essential software evolution issues in the following categories: software evolution process, software evolution traceability, software evolution description, software evolution management, and software evolution control.

The RH model can realize automated software evolution in multi-dimensional phases, such as software prototype or product demo, issue analysis, requirement analysis, specification design, module implementation, program integration, and software product implementation. Many types of software evolution objects in each phase, and dependencies among these objects have been defined to describe software evolution processes.

CASES is developed using the object-oriented tool: Java JDK1.1.7. CASES can enhance software evolution capacities of the Computer-Aided Prototyping System (CAPS) and provide automated assistance throughout software evolution processes, using inferred dependencies to support the practical development of complex systems by physically distributed teams of developers. CASES also has generalization characteristics for designing a software system in different software evolution processes and good performance when compared to other tools: QSS DOORS, PST, and ECS by different criteria. We have developed prototypes of C4I systems to conduct and validate our results.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
- A. PURPOSE OF THIS RESEARCH 1
- B. PREVIOUS WORK AND OPEN ISSUES 1
  - 1. Software evolution process 2
  - 2. Software evolution traceability 2
  - 3. Software evolution description 3
  - 4. Software evolution control 4
  - 5. Software evolution management 5
- C. PRELIMINARY IDEAS 5
- D. RESEARCH METHODOLOGY 6
  - 1. Formalization of software evolution 6
  - 2. Software evolution component reuse 7
  - 3. Dependency-computing model 7
- E. CONTRIBUTIONS 8
- F. ORGANIZATION OF DISSERTATION 9

II. RELATED TECHNICAL BACKGROUND 11
- A. FORMAL METHODS 11
  - 1. Definitions 12
  - 2. Rules 13
  - 3. Graphs 14
  - 4. Object-oriented implementation tools 15
- B. SOFTWARE PROTOTYPING 16
  - 1. Computer-aided prototyping system 16
  - 2. Software evolution through rapid prototyping 17
  - 3. Requirements engineering 18
  - 4. Specification-based prototyping 19
- C. DEPENDENCY APPROACH 20
  - 1. Consistency maintenance 20
  - 2. Project network 21
  - 3. Dependence graph 22
- D. SOFTWARE COMPONENT REUSE 22
  - 1. Reusable software component storage and retrieval 22
  - 2. Multi-level filtering for software component retrieval 23
- E. AUTOMATED SOFTWARE EVOLUTION 24
  - 1. Graph model 24
  - 2. Management and control of software evolution 25
  - 3. Object-oriented software evolution 26
- F. C4I SYSTEMS 27

1. C2 systems 28
2. C3 systems 29
3. C3I systems 30
4. C4I systems 30
III. RELATIONAL HYPERGRAPH MODEL....33
A. HYPERGRAPH 33
B. RELATIONAL HYPERGRAPH 41
C. SOFTWARE EVOLUTION STEPS 45
D. SOFTWARE EVOLUTION PROCESS 50
E. RELATIONAL HYPERGRAPH NET 55
IV. COMPUTER-AIDED SOFTWARE EVOLUTION....63
A. COMPUTER-AIDED SOFTWARE EVOLUTION SYSTEM 63
1. Software evolution description 63
2. Preliminary software evolution process....64
3. Formalized software evolution process....66
4. CASES functions 66
a. Step refinement....68
b. Project evaluation....68
c. Constraint management 68
d. Personnel management 68
e. Step management....69
f. Component management 69
g. Component traceability....69
h Version control and configuration management....69
i. Dependency management....69
j. Inference rule management....70
5. Evolution process using CASES and CAPS 70
a. Software prototype demo step 70
b. Issue analysis step....70
c. Requirement analysis step....70
d. Specification analysis step 71
e. Module implementation step....71
f. Program integration step....71
g. Software product demo step....71
h. Software product implementation step 71
6. Dynamic state model of evolution steps 72
7. Project team organization....75
a. Stakeholder classification 75
b. Organization structure of project teams....77
B. REUSABLE ARCHITECTURE 78
1. Repositories of software evolution steps and components 78
2. Lightweight inference engines and input component search engine 79
3. Attribute and content retrieval engines 80

4. Job assignment engine 81
5. Software evolution search functions 82
a. Relational hypergraph net search functions 82
b. Component search functions 82
c. Step search functions 82
d. Dependency search functions 83
e. Object number functions 83
f. Property evaluation functions 83
V DEPENDENCY-COMPUTING MODEL 85
A. SOFTWARE EVOLUTION 86
1. Preliminary 86
a. Object types 86
b. Object attributes and functions 86
c. Dependencies 87
2. Dependency rules 89
a. Object identifiers 89
b. Object variants and versions 90
c. Primary and secondary inputs 91
d. Evolution history splitting and merging 91
e. Variant and version numbering 92
f. Object decomposition 94
g. Test scenarios and the virtual teams 95
h. Component generation 96
i. Component retrieval: SPIDER formation 96
j. Unknown dependencies 99
B. JOB SCHEDULING 100
1. Job scheduling model 100
a. Schduling constraints 100
b. Scheduling heuristics 102
c. Scheduling algorithms 102
2. Dependency rules 104
a. Step attributes and dependency functions 104
b. Scheduling dependency rules 104
c. Scheduling policy rules 106
(1) Policy 1 106
(2) Policy 2 106
(3) Policy 3 106
(4) Policy 4 107
(5) Policy 5 107
(6) Policy 6 107
VI. DESIGN OF CASES 109
A. CLASS STRUCTURE 110
B. FILE STRUCTURE 111

C. USER INTERFACE 114
1. Project 115
a. Create project 116
(1) Step type 116
(2) Component type 117
(3) Evolution process 118
(4) Dependency 118
b. Open project 119
c. Delete project 119
d. Exit 120
2. Automated version control 120
a. Create step version 120
b. Open step version 121
c. Evolution history splitting 121
d. Evolution history merging 122
3. SPIDER 122
a. Edit 122
b. Decompose 123
c. Component content 125
d. Step content 127
e. Trace 129
(1) Trace 129
(2) Decompose 130
(3) Component content 130
(4) Step content 133
(5) Home, forward, and backward 133
4. Tools 134
a. Text editor 134
b. MS Word 134
c. MS Excel 134
d. Netscape 134
e. CAPS 135
f. Personnel data 135
5. Job schedule 136
a. Scheduling 136
b. Assignment 137
VII. EVOLUTION OF C4I SYSTEMS 139
A. FEATURES OF C4I SYSTEMS 139
B. APPROACHES 140
1. Prototyping a C4I system 140
2. Automating evolution processes 141
C. C4I/MD SYSTEM EVOLUTION 141
1. Initial prototype of MD systems 142

a. Requirement analysis step....142
b. Specification design step....143
c. Module implementation step....146
d. Program integration step....146
2. Second prototype of MD systems....147
a. Software prototype demo step....147
b. Issue analysis step....149
c. Requirement analysis step....150
d. Specification design step....150
e. Module implementation step....153
f. Program integration step....154
VIII. EVALUATION AND VALIDATION....157
A. EVALUATION....157
1. By a requirements management tool: QSS DOORS....157
a. QSS DOORS....157
b. CASES....159
(1) Design purpose....159
(2) Software evolution process....159
(3) Automation....159
(4) Connection....160
(5) Traceability....160
(6) Job scheduling and assignment....160
2. By job scheduling and assignment algorithms....160
a. Scheduling algorithms of John Evans....160
b. Scheduling and assignment algorithm of CASES....162
3. By inferred dependencies....164
B. VALIDATION....164
1. By CASES users....164
a. Project organizers....165
b. Project evaluators....167
c. System analysts and system designers....168
2. By C4I/MD systems....169
a. Organize a project....169
b. Propose a job....170
c. Approve a job....170
d. Schedule and assign a job....171
e. Complete or decompose a job....171
IX. CONCLUSIONS....173
A. CONCLUSIONS....173
B. LIMITATIONS....174
C. FUTURE DIRECTIONS....174
1. Network manipulation....174
2. Job scheduling improvement....175

3. Special skill management 175
4. Inter-project component reuse 175
5. Database management 175
6. Lightweight inference engine 176
7. Risk assessment 176
LIST OF REFERENCES 177
APPENDIX A. ATTRIBUTES OF FILES 185
APPENDIX B. CASES USER INTERFACE 193
APPENDIX C. SPIDERS OF C4I/MD SYSTEMS 257
APPENDIX D. JAVA CODE OF CASES 269
GLOSSARY 485
INITIAL DISTRIBUTION LIST 499

LIST OF FIGURES

Figure 1: A hypergraph H reachable from another hypergraph R 34
Figure 2: A decomposition of nodes n_4' and n_1 36
Figure 3: A hypergraph H is coreachable from another hypergraph R. 37
Figure 4: A minimal hypergraph H 37
Figure 5: A hyperpath P in hypergraph H 38
Figure 6: A refinement of minimal hypergraph H_m 40
Figure 7: Primary-input-driven and secondary-input-driven hypergraph 44
Figure 8: A relational hypergraph 45
Figure 9: Software evolution steps with their related components 49
Figure 10: The software evolution graph of an object driven by primary input 50
Figure 11: The software evolution graph driven by secondary input 51
Figure 12: Software evolution process driven by primary and secondary inputs 54
Figure 13: Separated relational hypergraphs 57
Figure 14: Relational hypergraph net 59
Figure 15: Different software evolution processes 65
Figure 16: Software evolution processes with CASES 67
Figure 17: State transition diagram for software evolution steps 73
Figure 18: State transition diagram for software evolution steps (extended) 74
Figure 19: Stakeholder classification 75
Figure 20: Organization structure of project teams 77
Figure 21: The variant and version numbering 93
Figure 22: The new component generation in the software prototype demo step 97
Figure 23: Three-tier object-oriented architecture of CASES 109
Figure 24: System context diagram of CASES 110
Figure 25: CASES class structure 112
Figure 26: CASES file structure 113
Figure 27: CASES user interfaces 115
Figure 28: CASES (1) 193
Figure 29: CASES (2) 193
Figure 30: Project menu bar 194
Figure 31: Automated version control menu bar 194
Figure 32: SPIDER menu bar 194
Figure 33: Tools menu bar 195
Figure 34: Job schedule menu bar 195
Figure 35: Create project frame 195
Figure 36: Open project – file chooser (1) 196
Figure 37: Open project – file chooser (2) 196
Figure 38: Open project – confirmation 196
Figure 39: Project schema – step type (1) 197
Figure 40: Project schema – step type (2) 197
Figure 41: Project schema – component type (1) 198

Figure 42: Project schema – component type (2) 198
Figure 43: Project schema – evolution process (1) 199
Figure 44: Project schema – evolution process (2) 199
Figure 45: Project schema – dependency (1) 200
Figure 46: Project schema – dependency (2) 200
Figure 47: Project schema – dependency (3) 201
Figure 48: Project schema – dependency (4) 201
Figure 49: Project schema – dependency (5) 202
Figure 50: Delete project (1) 202
Figure 51: Delete project (2) 202
Figure 52: Automated version control – create step version (1) 203
Figure 53: Automated version control – create step version (2) 203
Figure 54: Automated version control – create step version (3) 204
Figure 55: Automated version control – create step version (4) 204
Figure 56: Automated version control – open step version (1) 205
Figure 57: Automated version control – open step version (2) 205
Figure 58: Automated version control – open step version (3) 206
Figure 59: Automated version control – open step version (4) 206
Figure 60: Automated version control – evolution history splitting (1) 207
Figure 61: Automated version control – evolution history splitting (2) 207
Figure 62: Automated version control – evolution history splitting (3) 208
Figure 63: Automated version control – evolution history splitting (4) 208
Figure 64: Automated version control – evolution history merging (1) 209
Figure 65: Automated version control – evolution history merging (2) 209
Figure 66: Automated version control – evolution history merging (3) 210
Figure 67: Automated version control – evolution history merging (4) 210
Figure 68: Automated version control – evolution history merging (5) 211
Figure 69: Automated version control – evolution history merging (6) 211
Figure 70: Automated version control – evolution history merging (7) 212
Figure 71: File chooser (1) of SPIDER – edit 212
Figure 72: File chooser (2) of SPIDER – edit 213
Figure 73: File chooser (3) of SPIDER – edit 213
Figure 74: SPIDER – edit 214
Figure 75: File chooser (1) of SPIDER – decompose 214
Figure 76: File chooser (2) of SPIDER – decompose 215
Figure 77: SPIDER – decompose 215
Figure 78: File chooser (1) of SPIDER – component content 216
Figure 79: File chooser (2) of SPIDER – component content 216
Figure 80: File chooser (3) of SPIDER – component content 217
Figure 81: SPIDER – component content (1) 217
Figure 82: SPIDER – component content (2) 218
Figure 83: SPIDER – component content (3) 218
Figure 84: Add a component content in the component content editor (1) 219

Figure 85: Add a component content in the component content editor (2).....219
Figure 86: Add a component content in the component content editor (3).....220
Figure 87: Add a component content via browser (1).....220
Figure 88: Add a component content via browser (2).....221
Figure 89: Add a component content via browser (3).....221
Figure 90: Add a component content via browser (4).....222
Figure 91: Add a component content via browser (5).....222
Figure 92: Edit a component content in the component content editor (1).....223
Figure 93: Edit a component content in the component content editor (2).....223
Figure 94: Edit a component content in the component content editor (3).....224
Figure 95: Edit a component content in the component content editor (4).....224
Figure 96: Edit a component content via browser (1).....225
Figure 97: Edit a component content via browser (2).....225
Figure 98: Edit a component content via browser (3).....226
Figure 99: File chooser (1) of SPIDER – step content.....226
Figure 100: File chooser (2) of SPIDER – step content.....227
Figure 101: File chooser (3) of SPIDER – step content.....227
Figure 102: SPIDER – step content (1).....228
Figure 103: SPIDER – step content (2).....228
Figure 104: SPIDER – step content (3).....229
Figure 105: SPIDER – step content (4).....229
Figure 106: SPIDER – step content (5).....230
Figure 107: SPIDER – step content (6).....230
Figure 108: SPIDER – step content (7).....231
Figure 109: SPIDER – step content (8).....231
Figure 110: SPIDER – step content (9).....232
Figure 111: Skill list (1) of SPIDER – step content.....232
Figure 112: Skill list (2) of SPIDER – step content.....233
Figure 113: Skill list (3) of SPIDER – step content.....233
Figure 114: Predecessor list of SPIDER – step content.....234
Figure 115: Date chooser of SPIDER – step content.....234
Figure 116: File chooser (1) of SPIDER – trace.....235
Figure 117: File chooser (2) of SPIDER – trace.....235
Figure 118: SPIDER – trace.....236
Figure 119: Trace a component in the SPIDER – trace (1).....236
Figure 120: Trace a component in the SPIDER – trace (2.....237
Figure 121: Trace a component in the SPIDER – trace (3).....237
Figure 122: Trace a component in the SPIDER – trace (4).....238
Figure 123: Decompose a component in the SPIDER – trace (1).....238
Figure 124: Decompose a component in the SPIDER – trace (2).....239
Figure 125: Decompose a component in the SPIDER – trace (3).....239
Figure 126: Review component content in the SPIDER – trace (1).....240
Figure 127: Review component content in the SPIDER – trace (2).....240

Figure 128: Review component content in the SPIDER – trace (3) 241
Figure 129: Review component content in the SPIDER – trace (4) 241
Figure 130: Review component content in the SPIDER – trace (5) 242
Figure 131: Review component content in the SPIDER – trace via notepad 242
Figure 132: Review step content in the SPIDER – trace 243
Figure 133: Trace a component in the SPIDER – trace by backward button 243
Figure 134: Trace a component in the SPIDER – trace by forward button 244
Figure 135: Trace a component in the SPIDER – trace by home button 244
Figure 136: Tools – notepad 245
Figure 137: Tools – MS Word 245
Figure 138: Tools – MS Excel 246
Figure 139: Tools – Netscape 246
Figure 140: Tools – CAPS 247
Figure 141: Tools – personnel data 247
Figure 142: Add personnel data in the Tools – personnel data (1) 247
Figure 143: Add personnel data in the Tools – personnel data (2) 248
Figure 144: Add personnel data in the Tools – personnel data (3) 248
Figure 145: File chooser for editing personnel data in the Tools – personnel data 249
Figure 146: Edit personnel data in the Tools – personnel data 249
Figure 147: Delete personnel data in the Tools – personnel data (1) 249
Figure 148: Delete personnel data in the Tools – personnel data (2) 250
Figure 149: Delete personnel data in the Tools – personnel data (3) 250
Figure 150: Job schedule – scheduling (1) 251
Figure 151: Job schedule – scheduling (2) 251
Figure 152: Job schedule – scheduling (3) 252
Figure 153: Job schedule – scheduling (4) 252
Figure 154: Job schedule – scheduling (5) 253
Figure 155: Job schedule – scheduling (6) 253
Figure 156: Job schedule – assignment (1) 254
Figure 157: Job schedule – assignment (2) 254
Figure 158: Job schedule – assignment (3) 255
Figure 159: Job schedule – assignment (4) 255
Figure 160: Job schedule – assignment (5) 256
Figure 161: A top-level data flow diagram – *c4i* 144
Figure 162: A decomposed data flow diagram – *gui* 145
Figure 163: A decomposed data flow diagram – *ctrl* 145
Figure 164: A demo panel of the initial prototype program 148
Figure 165: A top-level data flow diagram – *c4i (modified)* 151
Figure 166: A decomposed data flow diagram – *gui (modified)* 152
Figure 167: A decomposed data flow diagram – *ctrl (modified)* 152
Figure 168: A demo panel of the second prototype program 155
Figure 169: Life cycle phases and tools 158
Figure 170: Plot of scheduler run-time vs. number of tasks to schedule 161

Acknowledgements

First of all, I would like to express my gratitude to Professor Valdis Berzins, my advisor, and Professor Luqi, the chair of Software Engineering Program, for their guidance, support, and patience. Without their guidance this work would not have been possible. A special thanks to Professor Mantak Shing, Professor Bruce Shultes, Professor William G. Kemple, Professor Wei Kang, Professor Jiang Guo, Professor Swapan Bhattacharya, Juan Carlos Nogueira, and Hanh Le for taking time to discuss various aspects of software evolution and the formal model with me, and Ron Russell for editing this dissertation.

Secondly, I thank Professor Dan Boger, the chairman of the Computer Science Department, and his staff for supporting my dissertation research. Especially, I would like to acknowledge my remaining committee members: Professor Xiaoping Yun and Professor Craig Rasmussen, for their comments and support.

Finally, I would like to thank my lovely wife, Lifen Wang, and my two children, Po-wei, and Jenny for their encouragement and support. I also would like to share this honor with my dear father and mother for bearing, raising, and educating me.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE OF THIS RESEARCH

The fundamental purpose of this research is to automate the evolution of large and complex software systems. It is still true that industry is largely unsuccessful when it comes to building software in general and large software projects in particular. Of the 175,000 information technology projects underway in the U.S. as we speak, 31% overall will end in failure. For larger projects characteristic of enterprise development, 40% will be cancelled and another 33% will be completed significantly late and over budget. No industry, type of business, or company is immune [ROET98].

In our research, systematic management and automation of software evolution processes, such as formalizing the software evolution process [LUQI97], prototyping the software [LUQI88a], seeking the relationship among software evolution objects [BADR94] [LIEB93] [SEIT98], and reusing software evolution components [GOGU96] [SOMM96], can hold out the promise of significantly improving these numbers.

B. PREVIOUS WORK AND OPEN ISSUES

There are numerous results related to our research about the formalization and automation of software evolution processes that have been published in the past decade. These include object-oriented software evolution [LIEB93] [SEIT98], software evolution through computer-aided prototyping [LUQI92], a graph data model and control system for evolution [LUQI90], a hypergraph model for evolution [LUQI97], a merging process for prototyping [DAMP94], a mechanism for retrieving reusable components [GOGU96], an evolution control system model and algorithms for software evolution [BADR94], and a model and decision support mechanism for software requirements engineering [BERZ97].

A review of previous software evolution research shows that many important problems still need to be resolved. These problems related to our research can be classified into five

problem domains: software evolution processes, software evolution traceability, software evolution description, software evolution control, and software evolution management.

1. Software evolution process

In [IBRA96], Ibrahim studied software evolution processes and created a schematic model of the analysis process based on the Issue-Based Information Systems (IBIS) model [CONK88]. The IBIS model follows the principle that the design process for complex systems is fundamentally a conversation among the stakeholders (e.g., customers, designers, and implementers) to resolve design issues. This model was extended to encompass prototype demos, analysis, and design activities, and was applied to design a decision support mechanism for software requirements engineering.

In [BADR94], the Evolution Control System (ECS) provides automated assistance for the software evolution process in an uncertain environment where designer tasks and their properties are always changing. The functions of the ECS are available. However, the following issues raised by this work have not been resolved:

- What are the details of software evolution processes?
- What dependencies are most important in these processes?
- How do you construct a software product from a validated software prototype?

In [IBRA96], Ibrahim suggested to augment the software evolution process by a mechanism that checks consistency in requirements as new requirement components are added or existing ones are changed. Unfortunately, he was unable to exploit this insight.

2. Software evolution traceability

In [IBRA96], Ibrahim did not discuss the issue of software evolution traceability, although he extended the graph model [LUQI90] to better represent software requirements issues. Ibrahim also did not discuss the details of recording the software evolution activities in the software evolution process, even though the decision support mechanism on the specific tasks and activities is described well.

In [LUQI97], Luqi and Goguen described the importance of hyper-requirements and pursued several projects on improving the acquisition, traceability, accessibility, modularity, and reusability of the many objects that arise and are manipulated during software development, with a particular focus on the role of requirements. We recognized that there are several challenges, including formalizing dependencies and developing methods for calculating dependencies and propagating the implications of changes. The following issues remain to be studied:

- How do you trace software evolution history within software evolution processes?
- How do you efficiently record and trace software evolution activities within the software evolution process?

3. Software evolution description

In [LUQI90], Luqi proposed a graph model of software evolution and showed how the model can help in maintaining the consistency of a changing system. The graph model was particularly concerned with large and complex systems, which often have long lifetimes and undergo gradual but substantial modifications because they are too expensive to discard and replace. The graph model represents the evolution history as a directed acyclic graph $G = [C, S, I, O]$, which is oriented bipartite with respect to the edges I and O, where

- C: software component nodes,
- S: evolution step nodes,
- $I \subseteq C \times S$: input relation from the system components to the evolution steps that operate on them, and
- $O \subseteq S \times C$: output relation from evolution steps to the components they produced.

In [BADR94], to model the hierarchical structure of the evolution history, the graph model was modified to be a graph $G = [C, S, CE, SE, I, O]$, where

- $CE \subseteq C \times C$: the *part_of* and *used_by* relations between the software components of a given configuration, and
- $SE \subseteq S \times S$: the *part_of* relation between a substep of a composite step and the composite step.

In [LUQI90], the hypergraph model was introduced to formalize the hierarchical structure in more detail. In order to realize automated software evolution in multi-dimensional phases, such as software prototype or product demo, issue analysis, requirement analysis, specification design, module implementation, program integration, and software product implementation, we need to define software evolution objects in each phase, and dependencies among these objects.

Therefore, the following issues should be resolved:

- How do you formally define and classify software evolution objects in each phase?
- What are the details of the dependencies among software evolution objects?
- Are the *part_of* and *affect/used_by* relationships enough to describe the dependencies among software evolution objects?
- What kinds of rules should be used within software evolution reasoning processes?

4. Software evolution control

In [BADR94], an ECS provides automated assistance to the software evolution manager to help him or her make the right decisions. An ECS has two main functions. The first is to control and to manage evolving software system components (version control and configuration management). The second is to control and to coordinate evolution team interactions (planning and scheduling software evolution tasks, which team members refer to as evolution steps). The ECS system manages both human resources and the design database, provides help needed by the software engineers, and facilitates the designers' tasks. Current ECS provides a simple way to record and track management decisions related to software modifications, but ECS lacks support for controlling and monitoring all activities related to managing a software maintenance effort.

Therefore, the following issues should be addressed:

- How do you automatically control the software evolution objects within software evolution processes?
- How to automate version control within software evolution processes?

5. Software evolution management

The problem of scheduling tasks with arbitrary precedence constraints and unit computation time in multiprocessor systems is NP-hard for both the preemptive and nonpreemptive cases [BADR94]. Badr developed and implemented an on-line scheduling algorithm for finding a feasible schedule that meets the deadlines and precedence constraints of all active steps or suggests new deadlines for lowest priority deadlines until a feasible schedule that meets all deadlines is reached. He did not attempt to improve the computation time of the algorithm. The problem studied in [BADR93] and [BADR94] is described below:

- Designers receive a skill level rating of low, medium, or high.
- Each evolution step has an estimated task duration, deadline, priority, and a required skill level.
- Each time a step is transited to the scheduled state, the utility recomputes the project schedule or alerts management that no schedule is possible with the given constraints.
- When the designer finishes the current assignment, the scheduler automatically assigns the next task based upon the scheduling algorithm and the timing, priority and skill level information stored in the ECS.

In [BADR94], the job scheduling model seems to be straightforward; however, it does not provide project personnel with flexibility, variation, or choice.

The following difficult questions remain to be solved:

- Is there a good heuristic scheduling algorithm for scheduling a set of independent processes on a set of identical processors?
- How do you define a level-of-skill indicator to match stakeholders with evolution activities in each phase?

C. PRELIMINARY IDEAS

The primary ideas of this paper are to identify the dependencies among software evolution objects and to provide a framework for integrating software evolution activities with configuration management, automated version control, and computer-aided project management. We focus on supporting practical software development by creating

automated tools that assist in the management of the software evolution process for a rapidly evolving system. The main topics of this study are the development of a Relational Hypergraph model (RH model) and the design of a Computer-Aided Software Evolution System (CASES) that can provide automated assistance throughout software evolution processes, using inferred dependencies to support the practical development of complex systems by physically distributed teams of developers.

Based on the state of an evolution step [BADR94] [LUQI97], the CASES has five common functions related to the software evolution activities: step refinement, project evaluation, constraint management, personnel management and step management. The CASES has five functions that are related to the software evolution components: component management, component traceability, version control and configuration management, dependency management, and inference rule management.

D. RESEARCH METHODOLOGY

1. Formalization of software evolution

Formal methods provide the best approach to create a large and complex software system and to describe its development and evolution process [LUQI97]. Because development environments and processes have different problem domains, some formal models have technical difficulties. However, the formal model of software evolution with object-oriented methods works well for describing software evolution processes, tracing evolution histories, exploring the dependencies among software objects, and retrieving reusable software evolution components [BADR94] [BERZ97] [GOGU96] [LIEB93] [MADH92] [SEIT98].

Several formal supports for software evolution have been developed in the past decade, such as the graph data model, the prototyping method, and the hypergraph model. Recently, the hypergraph model has been extended to the evolutionary hypergraph model and the RH model to explore and represent the complicated hierarchy and multidimensional structure of software evolution [LUQI90] [LUQI97] [HARN99a].

2. Software evolution component reuse

In order to reduce software development costs, software component reuse has been widely studied [GOGU96]. The concept of software component reuse was applied not only to the reuse of software code but also to the reuse of abstract code of specifications and designs [SOMM96]. In our study, reuse components should include software and all of the components that are related to software evolution, such as criticisms, issues, requirements, specifications, modules, programs, optimizations, test scenarios, and stakeholders, within software evolution processes. We call them software evolution components.

In the past few years, many of our efforts in the automated retrieval of reusable software components from software bases have been published [GOGU96] [SEIT98] [SOMM96]. In this study, we explore a new automation mechanism for the software evolution component reuse architecture. The software evolution component can be automatically retrieved by a lightweight inference engine to support the developer for executing the software evolution activities.

3. Dependency-computing model

The dependency-computing model integrates the fundamental software evolution model, such as the hypergraph model, the evolutionary hypergraph model, and the RH model, with the dependency rules that are driven by a lightweight inference engine. The lightweight inference engine is suitable to compute the small scale and specific domain inference rules [BERZ98].

There are two kinds of dependency rules: dependency generation rules and dependency action rules. According to the existing data, the lightweight inference engine computes dependencies among software evolution objects via dependency generation rules. The specific combination of dependencies can automatically support the software evolution via dependency action rules.

The preliminary dependency rules of automated decision support for processes are created as follows:

$$ALL(s : S, c : C :: c\ output\ s \Leftrightarrow c \in output(s)) \quad (1)$$

$$ALL(c1\ c2 : C :: c1\ same\_objcet\ c2 \Leftrightarrow c1.version.object\text{-}id = c2.version.object\text{-}id) \quad (2)$$

$$ALL(s : S, (c : C :: c\ input\ s \Leftrightarrow c \in input(s)) \quad (3)$$

$$ALL(s : S, c1\ c2 : C :: c1\ primary\_input\ s \Leftrightarrow c1\ input\ s\ \&\ c2\ output\ s\ \&\ c1\ same\_object\ c2) \quad (4)$$

$$ALL(s : S, c1\ c2 : C :: c1\ secondary\_input\ s \Leftrightarrow c1\ input\ s\ \&\ c2\ output\ s\ \&\ \neg(c1\ same\_object\ c2)) \quad (5)$$

E. CONTRIBUTIONS

We propose that a new mechanism, computer-aided software evolution based on inferred dependencies, can control and monitor software evolution activities related to manage- and design-phases and solve the problems listed in section B. There are at least five fundamental contributions:

- building a new automated software evolution architecture, CASES, to solve the problems of software evolution processes,
- enhancing the evolution graph model and the hypergraph model into the RH model to solve the problems of software evolution traceability,
- formalizing software evolution objects and their dependencies to solve the problems of software evolution description,
- determining and computing dependency rules to solve the problems of software evolution control, and
- improving the scheduling algorithm and developing a new ECS mechanism to solve the problems of software evolution management.

In addition to the above fundamental contributions, the following are the significantly creative contributions in the current software evolution study:

- extending the RH model from the hypergraph model to build a standard and

domain-specific software evolution process.

- creating the CASES based on the RH model and interfacing to CAPS to manage and control the software evolution process for iterative real-time embedded software prototyping.
- exploring the fact that the software evolution process is not unique and developing a mechanism in CASES for creating different software projects with different software evolution processes.
- inventing the SPIDER (Step Processed In Different Entrance Relationship) to simplify the dependency complexity of the software evolution and to construct the relational hypergraph net.
- using scheduling policy heuristics and allocating two job slots to each stakeholder for improving the job scheduling and assignment performance.

F. ORGANIZATION OF DISSERTATION

In Chapter II, we present the previous work and discuss some of the open issues of software evolution. In Chapter III, we build the RH model with mathematical definitions and apply these definitions to the software evolution process. In Chapter IV, we introduce the objects and functions of CASES, as well as the reusable architecture of software evolution. In Chapter V, we describe the dependency-computing model built by the dependency rules and the lightweight inference engines. In Chapter VI, we address the design of CASES by class diagrams, a file structure and user interface requirements. In Chapter VII, we study the evolution processes of C4I systems. In Chapter VIII, we evaluate and validate CASES by comparing it to other similar tools: QSS DOORS, PST, and ECS using different criteria. Chapter IX contains conclusions, limitations, and future directions.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATED TECHNICAL BACKGROUND

Before proceeding with this dissertation, some technical background, such as formal methods, software prototyping, dependency approach, software component reuse, automated software evolution, and C2 (Command and Control) Systems, must be explored.

A. FORMAL METHODS

Formal method is an approach that uses mathematical and logical definitions to formalize real-word object behavior and to obtain mature engineering disciplines. *Webster's Dictionary* defines *formal* as *definite, orderly*, and *methodical.* Formalization can be represented by definitions, rules, graphs, formulae, flow charts, Petri nets and other regular, orderly, and definite mechanisms. However, in computer science, the phrase "formal methods" has acquired a narrower meaning, referring specifically to the use of a formal notation to represent system models during program development [CRAI93] [LUQI97]. For example, we may develop a software evolution component with a formal notation, and then gradually transfer it into another component via a software evolution step or a history path; that is, software evolution steps need related inputs to generate an output component by a formal notation. Definitions and inference rules can be used to define software evolution objects and their fundamental dependencies, and to prove that input components are correct in the current step. In theorem-proving, however, definitions can be used manually because automating the notation used is difficult. Inference rules can be used easily to define software evolution objects as well as to prove the inputs of the current step are correct by inferred dependencies. These fallacies related to the practical usefulness of formal methods can be found in "Seven More Myths of Formal Methods" [BOWE95] and "An International Survey of Industrial Applications of Formal Methods" [CRAI93].

Formalization is a study or observance of prescribed or traditional forms. There are degrees of formalization, ranging from the highly formal *dry*, context-insensitive, to the highly informal *wet*, socially situated aspects of information [LUQI97]. Formalization is useful only to the extent that it helps meet concrete goals. For example, problem domains in automata, algorithms, and symbolic logic have already been developed, and practicing engineers of computer science only apply formalizations to solve practical problems. But software engineering is extremely intractable due to requirements instability. In this aspect, software engineering differs from the above more developed and traditional forms. Since there are no firm disciplines in software engineering, software engineers are hard pressed to effectively follow principles supporting the development of large scale and complicated software systems. In general, programmers consider application programs to satisfy certain mathematical properties, such as sorting, searching, etc., by using formal methods. However, demonstration programs cannot always satisfy a user's requirements. Experience suggests that formal methods are often tailored to a specific application especially involving human factors in requirements analysis. Therefore, what types of formal methods must be applied in such informal *wet* domains is an essential issue in software engineering.

Formal notations, such as definitions, rules, graphs, and object-oriented implementation tools, are very commonly used to explain the behavior of object-oriented components, although they are not the only ways to formalize software evolution processes. Each method has distinct functions for representing and interpreting software evolution. The best approach is to select appropriate formal methods for parts of issues and then to combine them to model software evolution in its entirety. Examples of graphs are instantiated to illustrate some profound definitions and rules. In our research, the following formal methods embrace different aspects for formalizing software evolution:

1. Definitions

In the formalization of software evolution, definitions are represented by mathematical and logical notations and are used to build the hypergraph model,

evolutionary hypergraph model, RH model, relational hypergraph net, and software evolution processes. We use definitions to construct a multidimensional and hierarchical structure of software evolution within a domain-specific development model that includes specified software evolution components and steps.

Generally, definitions are accompanied by theorems, corollaries, lemmas, propositions, and their proofs to derive further abstractions. Most formalism is represented by a long series of definitions in various computer science books, such as *Symbolic Logic and Mechanical Theorem Proving* [CHAN73], *Introduction to Automata Theory, Languages, and Computation* [HOPC79], and *Introduction to Algorithms* [CORM94]. Definitions can also be used well in describing software evolution. For example: in "A Theory of Program Modifications" [RAMA91], Ramalingam formalizes an algebra of program modifications through plentiful definitions, theorems, lemmas, and propositions; and in "Object-Oriented Software Evolution" [LIEB93], Lieberherr and Xiao successfully use a long series of 35 definitions and theorems to review propagation patterns for describing object-oriented software at a higher level of abstraction. In these papers, these authors observe that definitions, theorems, lemmas, and propositions have a fairly clear and simplistic nature in order to formalize object-oriented software evolution.

2. Rules

Rules are logical formulae that can be used to address software problems with formal logic. First-order logic is a prototypical formal notation that has been extensively studied and has inference rule sets known to be sound and complete for a convenient class of models [LUQI97].

Inference rules in software evolution formalization play a crucial role in automation of software evolution processes. Inference rules are denoted by formal logic and inferred by lightweight inferences [BERZ98] in the following domains: version control and configuration management, task decomposition and assignment, component

generation and retrieval, and dependency inference. We use inference rules to build up a small scale search space of different domain-specific environments.

Some definitions can be transferred into inference rules according to inference mechanism needs, but some definitions may not. If inference rules are interpreted as natural language with mathematical and logical notations, they can be considered as definitions. For example, in Chapter I the inference rule *(1)*: $ALL(s : S, c : C :: c\ output\ s \Leftrightarrow c \in output(s))$ can be interpreted as follows: "Let s denote steps and c denote components. For all s in step set S and all c in component set C it is the case that the dependency between c and s is *output* if and only if c belongs to the output of s."

The inference rules are stated by a logical notation which is used in the Spec language. Spec is a formal specification language based on logic. Berzins and Luqi have successfully used the Spec language for defining environment models to help develop precise descriptions of complex problems [BERZ91]. In software evolution formalization, logical statements in Spec are built from functions and relationships using the connective & (and), | (or), ¬ (not), ⇒ (implies), ⇔ (if and only if), :: (such that or it is the case that) and the quantifiers *ALL* (for all) and *EXISTS* (there exists).

3. Graphs

Graph theory is a relatively new field in mathematics. Most of the basic results were discovered only in this century. Nevertheless, by now graph theory is a developed and well-understood field, with thousands of results. Graphs can be used to formalize software evolution objects and their relationships. Graphs can model a large variety of situations, and they have been used in diverse fields ranging from archaeology to social psychology [MANB89].

Graphs are very suitable to the formalization of object-oriented programming problems. Most object-oriented formalizations in software engineering use vertices and edges to address complex graph-theoretical problems. For example:

- program summary graphs are used to analyze flow-sensitive interprocedural data flow [CALL88];

- a graphic model is used to formalize software evolution [LUQI90];
- program dependence graphs are used in interprocedural software slicing [HORW92]; and
- a dataflow diagram in CAPS is used to describe program specifications and automated merging of software prototypes [DAMP94].

In order to formalize software evolution processes, we have extended a graph model of software evolution into a hypergraph model, an evolutionary hypergraph model, and a RH model [HARN99a] [LUQI90] [LUQI97]. The RH model illustrates a basic structure of software component traceability through a top-level step, processed in different entrance relationships, and its refined steps.

We use some properties of directed acyclic graphs (dag) for tracing software evolution history, decomposing software evolution objects, building and inferring dependencies among software evolution objects, reusing software evolution objects, and assigning tasks to designers.

4. Object-oriented implementation tools

A formal model of software evolution is needed to serve as a basis for smarter software tools. In object-oriented programming languages, we choose Java to implement our formal model of software evolution. To automate real-time software, a prototyping tool CAPS (Computer-Aided Prototype System) has been developed using C, Ada, and TAE Plus (Transportable Applications Environment Plus) under the SunOS environment. Recently, a new PC version of Java CAPS has being developed using Java JDK 1.2. The formal model in this research is implemented by Java JDK 1.1.7 and uses Java CAPS.

Java and Ada both offer comprehensive support for object-oriented software development. A comparison of the object-oriented features of Ada 95 and Java has been accomplished by Brosgol [BROS98]. He pointed out the OO model is more closely related to so-called "pure" OO languages such as Smalltalk and Eiffel. Java directly supports single inheritance and also offers a partial form of multiple inheritance through a feature known

as an "interface." A key property of Java is that objects are manipulated indirectly through implicit references to explicitly allocated storage.

Sun [SUN96] has described Java as a "simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multi-threaded, and dynamic language." In brief, Java is an object-oriented language with features for objects/classes, encapsulation, inheritance, polymorphism, and dynamic binding [BROS98]. One can use Java to write stand-alone programs, known as applications, in much the same way that one would use Ada, C++, or other languages.

Therefore, in our research, we have used Java to implement the formal model of software evolution.

B. SOFTWARE PROTOTYPING

1. Computer-aided prototyping system

In [LUQI88a], to improve programming productivity and software reliability, Luqi introduced a software technology called computer-aided software prototyping. In this approach, the traditional Software Development Life Cycle (SDLC) is replaced by a life cycle with two phases: rapid prototyping and automatic program generation. Her approach to rapid prototyping uses a specification language, called Prototype System Description Language (PSDL), integrated with a set of software tools, including an execution support system, a rewrite system, a syntax-directed editor (SDE) with graphics capabilities, a software base, a design database, and a design-management system.

PSDL provides two kinds of basic building blocks for prototypes: data types and operators. Software systems are modeled as networks of operators communicating via data streams. PSDL provides graphical notation for dataflow diagrams enhanced with nonprocedural control timing constraints. Each operator is atomic or composite. Good modularity is important for increasing productivity. A system's understandability, reliability, and maintainability are especially important in rapid prototyping. The nonprocedural control constraints are easy to use because their meaning does not depend

on the order in which they appear. The language and its associated prototyping method lead to PSDL prototypes with a highly cohesive structure and few coupling problem. The prototyping method uses stepwise refinement to refine and decompose critical components selectively. These refinements and decompositions are kept in the design database. The prototype design is based on abstract functions, abstract data, and abstract control. Functional, data, and control abstracts can be used to hide lower level details [LUQI88a].

2. Software evolution through rapid prototyping

Since software evolution accounts for more than half of the total software costs, developers have focused on reducing the effort required. Prototyping provides one promising approach to achieving this goal. Rapid prototyping is the process of quickly building and evaluating a series of prototypes. Rapid prototyping supports software evolution as well as initial development. Computer-aided prototyping tools and object-based methods support the evolution of both prototypes and production software.

CAPS supports software evolution through object-based prototyping reusable software components. There are two kinds of prototyping objects in PSDL, corresponding to abstract data types (PSDL types) and abstract state machines (PSDL operators). Objects can also serve as natural units of work in a parallel implementation, since they can execute without interfering with each other.

In [LUQI89], Luqi states that one of the main difficulties of software evolution in traditional contexts is the lack of accurate requirements, specifications and design documents. What is needed is precise documentation to change the system reliably. In PSDL, specifications are part of the prototype description, and the implementation descriptions are provided at a design level. Therefore, PSDL can describe both the prototype and the production versions of the system.

Luqi also demonstrates that specification changes are needed when the customer finds the demonstrated behavior of the prototype unacceptable. PSDL provides statements for recording which requirements justify each attribute of an object in the prototype. The

system also has a set of heuristic rules for automatically propagating the effects of some types of specification changes, including changes to the maximum execution time, maximum response time, minimum calling period, and data types associated with the input streams and output streams of an object.

Thus, the customer and the developer must examine a series of changes to the proposed system's behavior and the perceived requirements to reach a common understanding. It costs less to use a prototype than to use a production-quality code to support this process because prototypes are simpler and easier to modify than production-quality implementations.

3. Requirements engineering

In [LUQI93], Luqi suggests how to best combine computer-aided prototyping with other aspects of requirements engineering, for the purpose of prototyping is to ensure that requirements specifications for the system adequately represent the "real requirements."

Luqi also states that the initial effort of requirements engineering is dominated by detective work – talking to many people to discover the stakeholders in the proposed system, their responsibilities, their roles in the relevant organizations, and what kinds of decision support each stakeholder wants. The prototyping process provides a concrete structure guide to refine the problem description and help analysts formulate questions that stakeholders will resolve. As a result, prototyping directly aids evaluating of proposed system behavior with respect to the goals of the stakeholders. The main problem solved by prototyping is communication: the actual goals of the stakeholders do not always match what they say, and those goals often shift in response to a better understanding of the system gained by concrete demonstrations of what it would be like to use that system.

In [LUQI93], prototyping helps to reveal questions, to elicit feedback from stakeholders, and to trigger some of the goal shifting before completion of requirements analysis rather than after implementation and delivery.

4. Specification-based prototyping

In [BERZ93], the use of software transformations for software evolution is explored. Transformation can be used to construct a program from a formal specification. But constructing correct specifications is difficult to accomplish because a set of informal ideas must be turned into a formal model in spite of incomplete and imprecise communication between people. Executable prototypes of the specification are useful for obtaining user confirmation by using a proposed specification to represent the problem correctly, and for guiding the reformulation of the specification in cases where it misrepresents the problem. Prototyping is most effective if the scenarios for demonstrations are carefully chosen to expose the most likely weaknesses of the requirements.

Berzins also describes how transformations can enhance the prototyping process by capturing the conceptual dependencies in a design. The prototyping process repeats a guess/check/modify cycle until the users agree that the demonstrated behavior is acceptable. There are two phases in the model of the prototyping process: prototype evolution and production code generation. The purpose of prototype evolution is to stabilize the software requirements before developers invest great effort in detailed implementation and optimization. The purpose of production code generation is to generate an efficient implementation when the requirements are stable. The prototype evolution phase consists of the activities labeled "analyze requirements," "construct prototype," and "execute prototype." The production code generation phase consists of the activities labeled "verify structure" and "implement (optimize)."

Berzins also states that prototype evolution phases are dominated by a series of nonmonotonic changes to the behavior of the prototype. These changes are achieved via contracting and extending transformations or via relaxing and constraining transformations. The production code generation phase is dominated by meaning-preserving transformations for optimizing the design and implementation.

C. DEPENDENCY APPROACH

1. Consistency maintenance

The dependency approach has been widely applied to software evolution research. In [AVEL92], a dependency approach to the consistency maintenance problem illustrates several viewpoints that provide effective management of the software evolution process [LEHM91]. In the following, Avellis summarizes that the dependency approach is useful to distinguish three families of dependencies vital in his research:

- *I-dependencies*: implementation or programming dependencies that use implementation consistency rules to make the software maintenance efficient so that if somebody changes part of a functional specification, then the implementation of the changed parts should "change appropriately";
- *T-dependencies*: technology dependencies that refer to properties of architectures interfaces, performance, user interaction, devices, operating systems and other tools; and
- *D-dependencies*: domain dependencies that exploit knowledge of the application domain that cannot be determined by examining the code alone.

In I-dependencies the implementation consistency rules are referred to as "fair low-level programming objects" belonging to a programming view of the system. The rule is as follows:

$$change(specification\text{-}A) \Rightarrow change(implementation\text{-}A)$$

This is certainly not detailed enough to explain software maintenance. Other examples of important dependencies are

- "*uses*" *dependencies*: If A uses B and B changes, then the implementation of A may have to be modified to adapt to the changes in B.
- "*input*" *dependencies*: If x is an input to A and the type of x changes, then the body of A should change.
- "*access methods*" *dependencies*: If x is a data structure, and the structure of x is changed, then the implementation of the access methods should change accordingly.

In T-dependencies we can explain the change in terms of knowledge about dependencies between: software architectures and performance; implementation

technology and performance; and data structure representation and algorithm efficiency. We call them *technology dependencies* because they are dependencies that relate to the objects involved in building the software system.

In D-dependencies, to actually know what the impact of a change is, and what other changes are required to make the system consistent, we need extensive domain-specific knowledge. As such, this knowledge is referred to as *domain-specific dependencies.*

Avellis focuses on the analysis of several classes of dependencies that are based on modeling the main objects in any view of the software system in a frame-based representation. He also focuses on making explicit the related dependencies in meta-knowledge bases. The dependency approach in [AVEL92] can be extended from software maintenance to automated software evolution.

2. Project network

In addition to I-dependencies, T-dependencies, and D-dependencies, we explore P-dependencies in project networks.

- *P-dependencies*: project dependencies that use *succeeds, precedes, input*, and *output dependencies* to define tasks and their relationships;

In software project management, each task's definition contains succeeds and precedes dependencies that link the task to its immediate neighbors in a schedule network. Furthermore input and output dependencies can be built to signify what a task requires as input to do its job and what it produces as output.

In [BIMS90], the input and output dependencies are used primarily as support for tracing dependencies through project network. However once we have both the precedes dependency and the input and output dependencies in place, it becomes apparent that precedes dependencies could be inferred directly from input and output dependencies; that is, task *A* precedes task *B* if *A* produces an output that *B* requires as input.

3. Dependence graph

In the software merging and slicing study, the dependence graph is a kind of dependency approaches to describe program components. A dependence graph has been used to solve the interprocedural-slicing-problem generating a slice of an entire program, where the slice crosses the boundaries of procedure calls. This dependence graph is introduced to represent programs, known as a *system dependence graph* [HORW90] [HORW92]. Slicing can help a programmer understand a complicated code, can aid in debugging [LYLE86], and can be used for automatic parallelization [BADG88] [WEIS83]. Horwitz's algorithm for interprocedural-slicing produces more precise answers than that produced by Weiser's algorithm presented in [WEIS84]. Horwitz follows the example of K. Ottenstein and L. Ottenstein by defining the slicing algorithm in terms of operations on a dependence graph representation program [OTTE84]. There are several dependences in the system dependence graph: *control dependencies*, *intraprocedural flow dependencies*, *transitive interprocedural flow dependencies* and so on.

To compute slicing problems in linear time, K. Ottenstein and L. Ottenstein introduced *program dependence graphs* to represent program. Once a program is represented by its program dependence graph, the slicing problem is simply a vertex-reachability problem [OTTE84]. To integrate several versions of a program into a common one, Horwitz introduced the program dependence graph to define the relationship between program components [HORW90] [HORW92]. Edges between program components represent dependencies. An edge represents either a control dependence or *data dependence*.

D. SOFTWARE COMPONENT REUSE

1. Reusable software component storage and retrieval

In [STEI91], R. Steigerwald, Luqi and J. McDowell show that a computer-aided software engineering (CASE) tool, which is used in conjunction with CAPS, will retrieve

reusable software components from a software base using a given specification. They employ five critical elements:

- The new technologies, often manifested in CASE tools, are based in reusable codes [MEYE94], computer-aided designs, and the automatic generation of programs. With respect to reusable component retrieval, the most important tool in CAPS is the software base management system (SBMS). Within this system the key to component storage and retrieval is the component's specification.
- The specification language used is PSDL. PSDL provides two kinds of building blocks for prototypes: abstract data types (ADTs) and operators. Software systems are modeled as networks of operators communicating via data streams.
- PSDL uses axioms of several different forms. The axioms are written using OBJ3. The axioms express the semantics of the specification and will be the basis of semantic normalization and matching, which is the second phase of the retrieval process. Syntactic and semantic normalization and matching provide the means for component storage and retrieval.
- The most widely known Ada software bases are the Common Ada Missile Parts Library (CAMP), the Ada Software Repository, and Booch component collection. Techniques that have been applied to the issue of component retrieval include browsers such as those found in Smalltalk, KEE, and Eiffel, keyword search algorithms, multi-attribute search algorithm, and expert systems. As stated above, the general methodology is to store components in an object-oriented database management system (OODBMS) and use PSDL specifications as the basis for retrieval.
- A query for a library component is a PSDL specification. The query is syntactically and semantically normalized and then matched against stored specifications. Syntactic and semantic normalization may proceed in parallel, but syntactic matching must occur before semantic matching in order to narrow the list of possible candidates. The main benefit of syntactic matching is speed, whereas the advantage of semantic matching is accuracy.

2. Multi-level filtering for software component retrieval

In [GOGU96], software reuse has been proven to improve the productivity and to produce a faster turnaround time for software projects. One of the central issues in software reuse is how to make better use of software libraries by improving the search and retrieval process [MEYE94]. Any solution to the retrieval problem should satisfy the following criteria:

- the retrieval process should be automated.

- any search algorithm in the retrieval process should be accurate.
- the search process should be effective.
- the user interface should allow a flexible and easy formulation of queries by the user and should provide insightful feedback to the user.

The algebraic specification language OBJ3 was used to perform some software-search-experiments in the context of the CAPS project. The search for reusable software components is organized as a series of increasingly stringent filters. These are profile and keyword filtering, signature matching, and ground-equation checking (semantic matching), all applied to software library components. The set of candidate components passing through the semantic matching process is referred to as a *choice set*. The output to the user includes the choice set as well as information about how the choice set is computed.

The results are based on the assumption that

- the components in a software library are written in a modern programming language, e.g., Ada.
- each component is assumed to have an executable algebraic specification with equations [GOGU96].
- the user's query is a partial algebraic specification, typically consisting of a signature and ground equations.

The results illustrated that users prefer partial specification over formal specification when formulating their queries. The study points out that the signature-matching algorithm produces identical results for approximate (using profile as a pruning mechanism) and non-approximate signature matching (exhaustive search).

E. AUTOMATED SOFTWARE EVOLUTION

1. Graph model

The evolution of software systems accounts for the bulk of their cost. However, there is currently little automated support for evolution, especially when compared to other aspects of software development. A graph model of software evolution is particularly concerned with large and complex systems, which often have long lifetimes and undergo

gradual but substantial modifications since they are too expensive to discard and replace [LUQI90].

Computer assistance is essential for the effective and reliable evolution of such systems because their representations and evolution histories are too complex for unaided human understanding. Computer-aided software evolution is particularly important in rapid prototyping, where exploratory design and prototype demonstrations guide the development of the requirements via an iterative process that can involve drastic conceptual reformulations and extensive changes to system behavior [LUQI89].

In [LUQI90], the CAPS graph model is a data graph model for evolution that records dependencies and supports automatic project planning, scheduling, and configuration management. According to this model, the evolution process of a software system is represented by a graph that at any given moment models the current and the past state of the software system. A typical instance of that graph consists of software objects that comprise the system configuration and the evolution activities (steps) applied to these objects. The graph model views a software evolution process as a partially ordered set of steps. Each change in the system design from the moment it is proposed is performed within the context of one or more steps. Steps have states that reflect the dynamic progression of the change from the moment the change is proposed until it is completed or abandoned. When rejected, the history of the activity remains in the project database. When completed, a step outputs a new version or versions of the subject software component underlying the change.

2. Management and control of software evolution

In [BADR94], Badr presents an ECS that provides automated assistance for the software evolution process in an uncertain environment where designer tasks and their properties are always changing.

Also in [BARD94], there are six different evolution states with each step expressing the different activities. In a proposed state, a proposed evolution step is

subjected to both cost and benefit analysis. Approval of a proposed step by the management triggers the decomposition process to create an atomic sub-step for each primary or affected component of the step. The "scheduled" state is reached from the "approved" state via the command "schedule_step" indicating that management constraints are complete and enabling scheduling and job assignment mechanisms. When a step is assigned, the version bindings of its inputs are automatically changed from generic to specific. In a completed state, the outputs of the step have been verified, integrated, and approved for release. In an abandoned state, the outputs of the step do not appear as components in the evolution history graph.

3. Object-oriented software evolution

Lieberherr and Xiao thought that software projects appeared as huge mountains [LIEB93]. Therefore, they invented evolution histories and growth plans to help programmers climb the mountain in small controlled phases with positive feedback after each testing phase. According to them, a propagation pattern defines a family of programs from which we can select a member by giving a class dictionary graph that details the structure of behavior through part-of and inheritance relationships between classes.

Lieberherr and Xiao introduced three concepts: evolution histories, growth-plans and propagation-directives. Their main contributions are that a programmer can incrementally write succinct high-level representations of programs so that his or her programs can be easily adapted and evolved to new environments and to constrain object descriptions through growth-plans for program testing.

Object-oriented concepts are applied to a software maintenance method named COMFORM (COnfiguration Management FORmalization for Maintenance). COMFORM is composed of several phases to provide the necessary guidance to maintain existing software systems [CAPR94].

In [CAPR94], a class (or type) is a template description which specifies common properties and behavior for a group of similar objects where object is an instance of a class.

The properties and behavior of objects, and their commonalities, are described in terms of attributes and operations. Inheritance is a powerful concept in [CAPR94], as it allows one to create new versions of forms which are a specialization of existing ones so that previous experience is reused.

In [PERR94], Perry thought it too limiting that software evolution is considered in terms of corrections, improvements and enhancements. He claimed that three inter-related ingredients are required in well-(software)-engineered systems: the domains that are relevant to these systems, the experience we gain from building, evolving and using these systems, and the processes we use in building and evolving these systems. Taking this holistic view, we gain insight into sources of evolution not only of the software systems themselves, but of their software evolution processes as well.

In [ABRA95], Abramson designed a tool for debugging programs, using evolutionary software techniques. He introduced a new distributed debugger Guard (Griffith University Relative Debugger), which operates in an open heterogeneous computing environment. Guard makes use of open-distributed-processing techniques to allow the reference code and the debugged code to execute concurrently on different computer systems. Guard relies on the user to make a number of assertions that compare data structures in the debugged code and in the reference version. These assertions make it possible to detect faulty code because they indicate where and when the data structures deviate from those in the reference code.

F. C4I SYSTEMS

In our study, we apply CASES to C4I (Command, Control, Communication, Computer, and Intelligence) systems to validate the RH model of software evolution. Why do we select C4I systems? The answer can be found in C4I systems which have the following features to match the formal model of software evolution:

- The requirement environment of developing C4I systems is mutable and unstable.
- The best way to create a C4I system is by an object-oriented modeling technique.
- C4I systems are real-time embedded systems.

- C4I systems are large and complex systems.
- Evolution of C4I systems can be formalized and carried out by automation tools.
- Rapid prototyping technology can help designers create adaptive C4I systems.
- Requirements of the new version C4I systems can be obtained after the performance of the C4I system is evaluated and measured.

In general, C4I systems include C2, C3, and C3I systems. The following are their definitions and research backgrounds related to the above features:

1. C2 systems

CALL (Center for Army Lessons Learned) Dictionary and Thesaurus defines C2 (Command and Control) as the exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission. C2 functions are performed by arranging personnel, equipment, communications, and procedures employed by a commander in planning, coordinating, and controlling forces and operations in the accomplishment of a mission. In [MALE98], Malerud defines C2 concept, C2 structure and C2 system as follows:

- C2 concept: A set of characteristics of a C2 system describing how it reaches its objective.
- C2 structure: An assembly of personnel, organization, procedures, equipment and facilities arranged to meet a given objective, and within fixed economical limits.
- C2 system: An assembly of personnel, organization, procedures, equipment and facilities organized to accomplish C2 related functions. A C2 system comprises three main components: C2 tasks, C2 functions and a C2 structure.

A formal model is developed by applying an object-oriented modeling technique to measure C2 systems. The formalization procedures are described in the following steps [MALE98]:

- An object model is developed capturing the static structure of the C2 system which include the objects of the system, relationships between the objects and attributes and operations that characterize each class of objects.
- A dynamic model is constructed consisting of state diagrams specifying when functions/processes in the system are executed.

- A functional model is developed specifying the functions/processes carried out in the system. The function model consists of flow diagrams which describe the flow of information between functions and objects.

These steps show that C2 systems can be formalized and measured by using an object-oriented modeling technique.

2. C3 systems

CALL Dictionary and Thesaurus defines C3 (Command, Control, and Communication) as the collection of capabilities required by commanders to exercise command and control of their forces. In C3 systems, communication systems are a collection of individual communications networks, transmission systems, relay stations, tributary stations, and data terminal equipment usually capable of interconnection and interoperation to form an integrated whole.

C3 systems are clearly complex systems that change frequently, and thus create demands for flexibility on the part of supporting software and flexibility in the supporting organizational architecture [LEE98]. Requirements analysis technology is one of the factors influencing the performance of C3 architectures. Lee's research on related C3 systems suggests that standard principles, such as minimum and priority, are insufficient in the modern world in which increased complexity and rapid changes are requiring greater flexibility [LEE98]. The ability to create an adaptive C3 system by rapid prototyping technology is an essential issue. Feedback from users via a prototype demo helps to control the adaptivity of the C3 architecture and the performance of a C3 system which can be evaluated and generated to a new version of a C3 system.

Once a C3 system is created, evaluation is required. For many years, the use of *mission threads* dominated the evaluation of C3 systems [BROD98]. Mission threads are a time-ordered series of messages required to perform an operational task. Unfortunately, testing in this manner is expensive and often fails to pinpoint bottlenecks or problem areas. Brodeen's research evaluates C3 systems by using message strings, so that experiments

involving the entire system can be better focused on military factors. These military factors include some validated service criteria provided by the communication system [BROD98].

3. C3I systems

CALL Dictionary and Thesaurus defines intelligence as the collection of functions that generates knowledge of the enemy, weather, and geographical features required by a commander in planning and conducting combat operations. In C3I systems, military intelligence collection can be considered a plan for gathering information from all available sources to meet an intelligence requirement. Therefore, real-time constraints are needed to build a C3I system for helping commanders make a correct decision. Because the C3I system represents the tactical commander's nerve center, it is an absolute necessity that the commander's C3I system has the sophistication to manage the data under the real-time decision aid.

4. C4I systems

CALL Dictionary and Thesaurus defines C4I as integrated systems of doctrine, procedures, organizational structures, personnel, equipment, facilities, and communications designed to support a commander's exercise of command and control through all phases of the operational continuum. In a C4I system, except for hardware, we focus on computer software. In our research, developing Object-Oriented Programs is one of the most important work for organizing a C4I system. In order to address an overwhelming unstable requirement environment, we intend to develop the next generation of information systems based on modular, object-oriented programming principles using rapid software prototyping to get early feedback from users.

For example, in [LOSS98], Lossau presents a technical architecture for distributed C4I-based applications. The architecture supports sharing of data instead of duplicating data, and provides a level of collaboration between C4I applications not existing in current environments. The structure is designed to support maximum platform portability and adherence to software standards. His conclusion is using distributed Object-

Oriented techniques whereby we are able to provide a mechanism for sharing information at its source without having to copy the data.

THIS PAGE INTENTIONALLY LEFT BLANK

III. RELATIONAL HYPERGRAPH MODEL

The *Relational Hypergraph model (RH model)* is a formal model for software evolution which can help us develop tools to manage both the activities in a software development project and the products that those activities produce [HARN99e]. This model incorporates some features of previous CAPS models into a more abstract mathematical structure. Our intention is to make the software evolution process easier to understand and implement. This model has been used successfully in CASES [HARN99d].

A. HYPERGRAPH

The *hypergraph model* [HARN99a] [LUQI97] represents the evolution history and future plans for software development as a hypergraph. Hypergraphs generalize the usual notion of a directed graph by allowing hyperedges, which may have multiple output nodes and multiple input nodes [BERG89] [GALL93] [LUQI97] [PRET93] [SACC85]. The following definitions that formalize hypergraph, path, and evolutionary hypergraph have been created by Luqi and Goguen [LUQI97]. We extend their work to define hypergraph set, minimal hypergraph, and refinement of a node, an edge, and a minimal hypergraph for building the RH model.

Definition 1. (Hypergraph) A *(directed) hypergraph* is a tuple $H = (N, E, I, O)$ where

1. N is a set of *nodes*,
2. E is a set of *hyperedges* (briefly called *edges*),
3. I: $E \rightarrow 2^N$ is a function giving the set of *inputs* of each hyperedge, and
4. O: $E \rightarrow 2^N$ is a function giving the set of *outputs* of each hyperedge [LUQI97].

Definition 2. (Path) Let $H = (N, E, I, O)$ be a hypergraph. A *path* p from a node n to a node n' is a sequence $e_1 \ldots e_k$ of $k > 0$ edges and a sequence $n_1 \ldots n_{k+1}$ of nodes such that $n_i \in I(e_i)$ and $n_{i+1} \in O(e_i)$ for $i = 1, \ldots, k$, where $n = n_1$ and $n' = n_{k+1}$ [LUQI97].

The traceability of the software evolution can be presented via the paths of a hypergraph.

Definition 3. A hypergraph $H = (N, E, I, O)$ is *acyclic* if and only if there is no path from any node in N to itself [LUQI97].

Definition 4. A set N of nodes is *reachable from* a set R of nodes if and only if there is a path to each $n \in N$ from some $n' \in R$. A hypergraph $H = (N, E, I, O)$ is *reachable from* a set R of its nodes if and only if its set N of nodes is reachable from R. A *root* of H is a node from which H is reachable. A *leaf* of H is a node from which no other node is reachable [LUQI97].

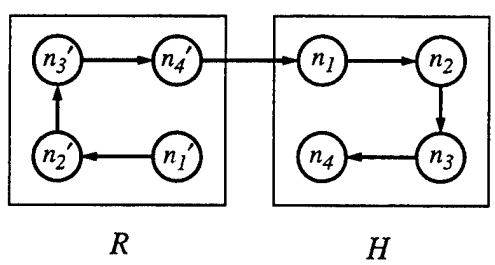


Figure 1: A hypergraph *H* is reachable from another hypergraph *R*.

Later hypergraphs are reachable from earlier hypergraphs through an evolution path [LUQI97]. For example, in Figure 1, the hypergraph H is reachable from another

hypergraph R. Each node in hypergraph H is reachable from the nodes in hypergraph R via a path. In this case, n_1' is a root of H and n_4 is a leaf of H.

These concepts can be enhanced to the composite node and composite edge, hypergraph set, minimal hypergraph, and refinement of a composite node (or called node decomposition) as follows:

Definition 5. (Hypergraph Set) Let $H = (N, E, I, O)$ be a hypergraph and $H_1 = (N_1, E_1, I, O)$, ... , $H_n = (N_n, E_n, I, O)$, be n hypergraphs. The hypergraph $H = H_1 \cup ... \cup H_n$ is called the *hypergraph set* of hypergraphs $H_1,..., H_n$ if the following conditions hold:

1. $N = N_1 \cup ... \cup N_n$ and $E = E_1 \cup ... \cup E_n$, and
2. if $O(e) \cap O(e') \neq \varnothing$ then $e = e'$; we call this the *identifiability* condition.

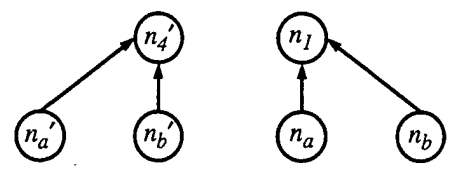
We say that the hypergraph H *combines* hypergraphs $H_1, ..., H_n$.

Definition 6. (Minimal Hypergraph) Let $H = (N, \{e\}, I, O)$ be a hypergraph with precisely one hyperedge, e. We say that H is the *minimal hypergraph.*

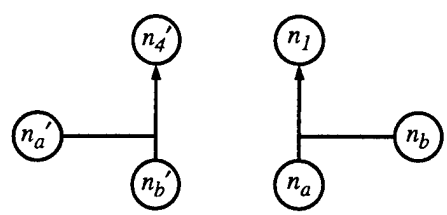
Definition 7. (Refinement of a Node) Let $H = (N, E, I, O)$ be a hypergraph. The *refinement of a node* $n \in N$ in H is a (directed) minimal hypergraph $H_m = (N_{in} \cup N_{out}, \{e\}, I, O)$, whose input node set N_{in} is $\{n_1, ... , n_n\}$, output node set N_{out} is $\{n\}$, and edge set is $\{e\}$, where edge e is called a *decomposition edge*. We say that node n is *decomposed* or *refined* into nodes $n_1, ... , n_n$, and node n is called a *composite node.*

In Figure 2 [a], node n_4' and node n_1 are two composite nodes that are decomposed into n_a' and n_b' as well as n_a and n_b, respectively. Node n_4' is reachable from subnodes n_a' and n_b', and node n_1 is reachable from subnodes n_a and n_b. Due to the *identifiability* condition, the decomposition edge of Figure 2 [a] can be briefly drawn as [b].

In Figure 1, if node n_4' has a set of subnodes n_a' and n_b' as Figure 2 [a], H is also reachable from each of subnodes because the *path directions* (also called *decomposition edge directions*) between parent node and subnodes are pointed to parent node. But if node n_1 has a set of subnodes n_a and n_b as Figure 2 [a], whose path directions are pointed to parent node, each of subnodes would not be reachable from R because of the path directions.



[a]



[b]

Figure 2: A decomposition of nodes n_4' and n_1

Definition 8. If $H = (N, E, I, O)$ is a hypergraph, then its *opposite*, denoted H^{op}, is the hypergraph (N, E, O, I). We say that H is *co-reachable from* another hypergraph $R = (N', E', I, O)$ if and only if H^{op} is reachable from R [LIQI97].

The definition of opposite hypergraph and co-reachable from can resolve the confusion issue of decomposition. In Figure 3, we focus on discussing the decomposition of nodes n_4' and n_1, and illustrate that a hypergraph H is co-reachable from another

hypergraph R if and only if hypergraph H^{op} is reachable from hypergraph R. Since each member in hypergraph H^{op} is reachable from some of members in hypergraph R via a path, we say that hypergraph H is co-reachable from hypergraph R.

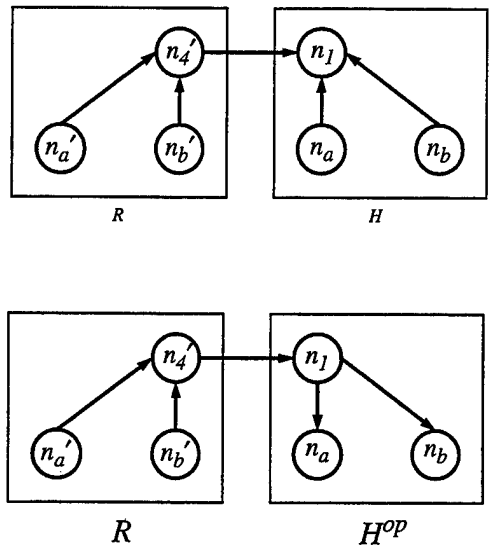


Figure 3: A hypergraph *H* is co-reachable from another hypergraph *R* if and only if hypergraph H^{op} is reachable from hypergraph *R*.

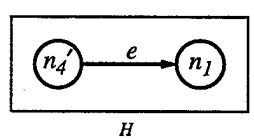


Figure 4: A minimal hypergraph *H*

In Figure 4, if nodes n_4' and n_1 in H represent the input and output node sets to a hyperedge e respectively, nodes n_4', n_1 and hyperedge e form a *minimal hypergraph.* If nodes n_4' and n_1 are decomposed into two sets of subnodes in minimal hypergraph, the

characteristics of reachable from and co-reachable from between parent node and subnodes in hypergraph H can be conducted by hyperpath. These definitions can be specified as follows:

Definition 9. (Hyperpath) A *hyperpath* in a hypergraph $H = (N, E, I, O)$ from $D \subseteq N$ to $T \subseteq N$ is a minimal hypergraph contained in H, whose node set contains D and T, and that is reachable from D and co-reachable from T; we call D and T the *input and output sets of the hyperpath*, respectively.

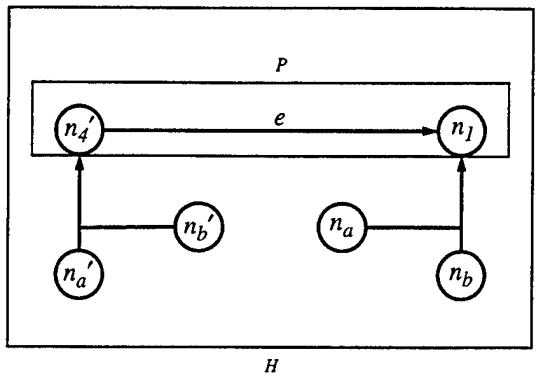


Figure 5: A hyperpath *P* in hypergraph *H*

In Figure 5, a hyperpath P from input set n_4' via hyperedge e to output set n_1 appears in a hypergraph H. Node n_4' is a set of subnodes n_a' and n_b' which are inputs to hyperedge e and node n_1 is a set of subnodes n_a and n_b which are outputs to hyperedge e. Therefore hypergraph P is reachable from input subnodes n_a' and n_b' and opposite hypergraph P^{op} is reachable from output subnodes n_a and n_b.

Actually, the hyperpath can be applied to the refinement not only of a composite node but also of a composite hyperedge and a minimal hypergraph. The refinement of a composite edge (also called edge expansion) and the refinement a minimal hypergraph can be described as edge expansion.

Definition 10. (Refinement of an Edge) Let $H = (N, E, I, O)$ be a hypergraph. The *refinement of an edge* $e \in E$ is a hypergraph set $R = (N_{in} \cup N_{out}, E_r, I, O)$ of minimal hypergraphs $H_1 = (A_1 \cup B_1, \{e_1\}, I, O), \ldots, H_n = (A_n \cup B_n, \{e_n\}, I, O)$, whose input node set N_{in} is $A_1 \cup \ldots \cup A_n$, output node set N_{out} is $B_1 \cup \ldots \cup B_n$, and edge set E_r is $\{e_1, \ldots, e_n\}$, where $e_1, \ldots, e_n$ are called *subedges*. We say that edge e is *expanded* or *refined* into subedges $e_1, \ldots, e_n$, and edge e is called a *composite edge*.

Definition 11. (Refinement of a Minimal Hypergraph) Let $H = (N, E, I, O)$ be a hypergraph. Let $H_m = (N_{in} \cup N_{out}, \{e\}, I, O)$ be a minimal hypergraph in H where input node set N_{in} and output node set N_{out} to edge e are composite nodes, and edge e is a composite edge. The *refinement of a minimal hypergraph* $H_m = (N_{in} \cup N_{out}, \{e\}, I, O)$ in H is a hypergraph set $R = H_{in} \cup H_{out} \cup H_e$ where H_{in} is a refinement of N_{in}, H_{out} is a refinement of N_{out}, and H_e is a refinement of e.

In other words, given a hypergraph $H = (N, E, I, O)$, a hyperedge e is said to be expanded if and only if e is a composite edge and e is refined by a set of subedges associated with their input node(s) and output node. So, an edge expansion of a hypergraph $H = (N, E, I, O)$ is a refinement of a hyperedge $e \in E$ by a hypergraph set having the same input and output sets.

In Figure 6, the refinement of input node n_4' is a minimal hypergraph $H_{in} = (\{n_a', n_b'\} \cup \{n_4'\}, \{d1\}, I, O)$, the refinement of output node n_1 is a minimal hypergraph $H_{out} = (\{n_a, n_b\} \cup \{n_1\}, \{d2\}, I, O)$, and the refinement of edge e is a hypergraph set $H_e = (\{n_a', n_b'\} \cup \{n_a, n_b\}, \{e1, e2\}, I, O)$. Therefore the refinement of a minimal hypergraph $H_m = (N_{in} \cup N_{out}, \{e\}, I, O)$ is a hypergraph set $R = H_{in} \cup H_{out} \cup H_e$ $= (\{n_a', n_b', n_a, n_b, n_4', n_1\}, \{e1, e2\}, I, O)$.

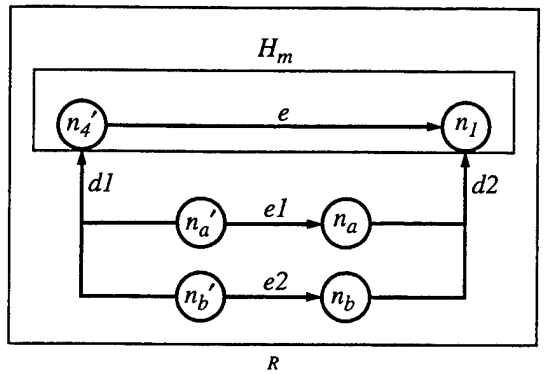


Figure 6: A refinement of minimal hypergraph H_m

Definition 12. (Evolutionary Hypergraph) An *evolutionary hypergraph* is a labeled, directed, and acyclic hypergraph $H = (N, E, I, O)$ together with label functions $L_N : N \rightarrow C$ and $L_E : E \rightarrow A$ such that the following assumptions are satisfied:

1. The elements of N represent unique identifiers for software evolution components;
2. The elements of E represent unique identifiers for software evolution steps;
3. The functions I and O give the inputs and outputs of each software evolution step, such that $O(e) \cap O(e') \neq \varnothing$ implies $e = e'$;
4. The function L_N labels each node with *component attributes* from the set C, including the corresponding version of the software evolution component;
5. The function L_E labels each edge with *step attributes* from the set A, including the current status of the software evolution step, such that $A = \{s, d\} \cdot A'$ (that is, each element of A has the form (s, a') or (d, a'), where $a' \in A'$). An edge labeled "s" is called a *step* and one labeled "d" is called a *decomposition* [LUQI97].

In the next section, we will use the definition of evolutionary hypergraph to construct relational hypergraph model.

B. RELATIONAL HYPERGRAPH

To describe a multi-dimensional hierarchy of software evolution objects and their relationships, we explore the RH model that is based on a hypergraph model and an evolutionary hypergraph model. This multi-dimensional hierarchy includes top-level, refined-level, and atomic-level relational hypergraphs with software evolution histories. The top-level relational hypergraph can be decomposed into refined levels until it becomes an atomic-level hypergraph that is a leaf of the decomposition tree. The software evolution history can be traced step by step along the software evolution path within each level of a relational hypergraph. No matter which level software evolution objects are in the relational hypergraph, the RH model can completely describe the *primary_input* and *secondary_input* dependencies.

A relational hypergraph is an evolutionary hypergraph in which the dependency relationships between components and steps can have a hierarchy of specialized interpretations. The input part to each hyperedge in a path could be a set of multiple input nodes containing various software evolution components. If an input node and an output node to an evolutionary hyperedge that are different versions of the same component exist, then the path from the input node via the hyperedge to the output node is called a *primary-input-driven path,* and the relationship between the input node and the step is called a *primary_input dependency*. If an input node and an output node of an evolutionary hyperedge exist and these are different components, then the path from the input node via the hyperedge to the output node is called a *secondary-input-driven path,* and the relationship between the input node and the step is called a *secondary_input dependency*.

The relational hypergraph records the history of software evolution and the relationship between software components and their related components. Before evolving a software system, we don't know what is the new version of the proposed software system until we run a sequence of evolution activities to get the proposed requirements. The development of the new software version is based on the new requirements and the old software version. The whole process can be recorded by a relational hypergraph.

The evolutionary hypergraph is a multi-level structure due to the refinement of the hyperedge. Actually, the hyperedge is a multi-level structure of the evolution step. The top-level evolution step can be refined into several atomic evolution steps as soon as its top-level evolutionary hypergraph is refined into several atomic evolutionary hypergraphs. This can be defined as follows [HARN99g]:

Definition 13. (Top-level Evolution Step) Let $H = (N, E, I, O)$ be an evolutionary hypergraph. A hyperedge $e \in E$ is called *top-level evolution step* if and only if the hyperedge e has no *parent evolution step*.

Definition 14. (Atomic Evolution Step) Let $H = (N, E, I, O)$ be an evolutionary hypergraph. A hyperedge $e \in E$ is called *atomic evolution step* if and only if the hyperedge e cannot be expanded to a step and its output set has at most one component.

Definition 15. (Top-level Evolutionary Hypergraph) A *top-level evolutionary hypergraph* is an evolutionary hypergraph $H = (N, E, I, O)$, each of whose hyperedge is a top-level evolution step.

Definition 16. (Atomic Evolutionary Hypergraph) An *atomic evolutionary hypergraph* is an evolutionary hypergraph $H = (N, E, I, O)$, each of whose hyperedge is an atomic evolution step.

Inputs to a hyperedge are classified as either *primary input* or *secondary,* or *nonprimary, input.* In a software evolution path, generally, there is a sequence of nodes from source to sink driven by primary input. However, the input part $I(e)$ to each hyperedge e in a path might not map into only one identical component but could be a set of multiple input nodes combining many kinds of software evolution components. Within these input

nodes to a hyperedge in an evolutionary hypergraph, a node in a path traced or driven by secondary input exists. This phenomenon can be addressed as follows:

Definition 17. (Primary-input-driven) A path p in an evolutionary hypergraph $H = (N, E, I, O)$ is called *primary-input-driven* if and only if for every hyperedge e in the path p, input node n and output node n' to hyperedge e are different versions of the same component and output node n' depends on input node n.

Definition 18. (Secondary-input-driven) A path p in an evolutionary hypergraph $H = (N, E, I, O)$ is called *secondary-input-driven* if and only if for every hyperedge e in the path p, input node n and output node n' to hyperedge e are different components and output node n' depends on input node n.

A primary-input-driven path addresses the evolution history of a software evolution component based on the change from an old version to a new version. A secondary-input-driven path addresses the evolution rationale with a sequence of the software evolution components. Therefore, these two structures form the relational hypergraph which determines not only what to evolve but also how to evolve it. The relational hypergraph can be characterized as follows:

Definition 19. (Primary-input-driven Hypergraph) An evolutionary hypergraph $H = (N, E, I, O)$ is called a *primary-input-driven hypergraph* if and only if for every hyperedge e in H and every input node n in $I(e)$, e is primary-input-driven, and the input node n is called *primary input node*.

Definition 20. (Secondary-input-driven Hypergraph) An evolutionary hypergraph $H = (N, E, I, O)$ is called a *secondary-input-driven hypergraph* if and only if for every

hyperedge *e* in *H* and every input node *n* in *I(e)*, *e* is secondary-input-driven, and the input node n is called *secondary input node*.

Definition 21. (Relational Hypergraph) An evolutionary hypergraph $H = (N, E, I, O)$ is called a *relational hypergraph* if and only if for every hyperedge *e* in *H* and every input node *n* in *I(e)*, the dependency between *n* and *e* is *primary_input* or *secondary_input.*

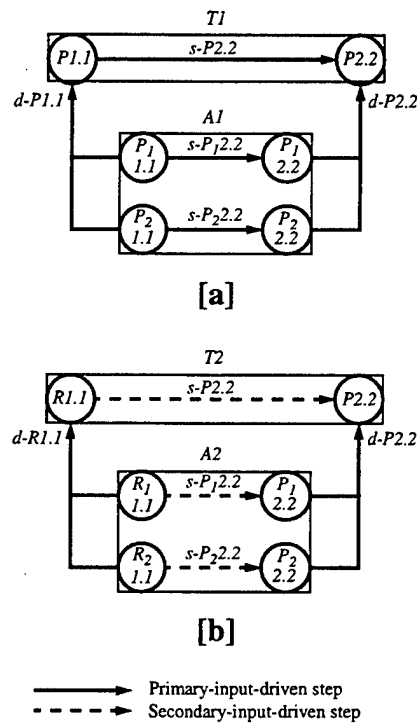


Figure 7: Primary-input-driven and secondary-input-driven hypergraph

Figure 7 [a] shows a primary-input-driven hypergraph since input node *P1.1* and output node *P2.2* to hyperedge *s-P2.2* are different versions of the same component *P*. Figure 7 [b] shows a secondary-input-driven hypergraph since input node *R1.1* and output node *P2.2* to hyperedge *s-P2.2* are different components. Hypergraphs *T1* and *T2* are top-level evolutionary hypergraphs with a top-level evolution step *s-P2.2*, and similarly

hypergraphs *A1* and *A2* are atomic evolutionary hypergraphs with two atomic evolution steps $s\text{-}P_1 2.2$ and $s\text{-}P_2 2.2$.

Primary-input-driven and secondary-input-driven hypergraphs in Figure 7 can be combined into a relational hypergraph shown in Figure 8. Steps *s-P2.2* at the top-level in different hypergraphs are the same step, and the inputs to step *s-P2.2* have two software evolution components, primary input node *P1.1* and secondary input node *R1.1*. Therefore, the relationships among nodes, subnodes, top-level steps and atomic steps can be established by this relational hypergraph.

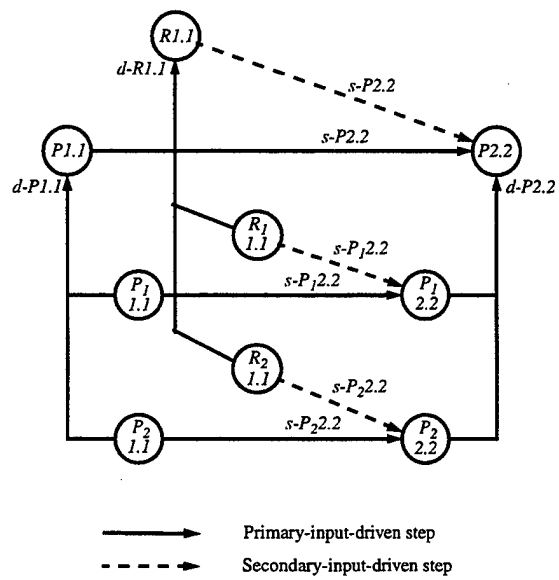


Figure 8: A relational hypergraph

C. SOFTWARE EVOLUTION STEPS

Actually, in the software evolution process, the related input components to each type of software evolution steps depends on real applications, but in order to formalize the software evolution process, we specify and extend software evolution steps and their related input components based on the Schematic Model of the Analysis Process modified

from the IBIS model in [IBRA96]. In that model the software evolution steps in software evolution process include: software prototype demo step, issue analysis step, requirement analysis step, and specification design step. We extend other four types into that model: module implementation step, program integration step, software product demo step, and software product implementation step. The software evolution components include the following ten types: *stakeholders, software prototype programs, software product programs, test scenarios, criticisms, issues, requirements, specifications, modules,* and *optimizations*. Stakeholders of a step can be regarded as virtual teams before the step (or job) is assigned to stakeholders.

Definition 22. (Software Prototype Demo Step) Let C_1 be a set of old-version criticisms, C_2 be a set of new-version criticisms, P be a set of software prototype programs, T be a set of software test scenarios, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *software prototype demo step* if and only if there exist C_1, C_2, P, T, and U, such that $I(s) = C_1 \cup P \cup T \cup U$ and $O(s) = C_2$.

In steps of definition 22, there are five software component sets, C_1, C_2, P, T, and U. C_1 is a primary input node. P, T, and U are secondary input nodes. C_2 is the result of this step.

Definition 23. (Issue Analysis Step) Let J_1 be a set of old-version issues, J_2 be a set of new-version issues, C be a set of criticisms, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *issue analysis step* if and only if there exist J_1, J_2, C, and U, such that $I(s) = J_1 \cup C \cup U$ and $O(s) = J_2$.

In steps of definition 23, there are four software component sets, J_1, J_2, C, and U. J_1 is a primary input node. C and U are secondary input nodes. J_2 is the result of this step.

Definition 24. (Requirement Analysis Step) Let R_1 be a set of old-version requirements, R_2 be a set of new-version requirements, J be a set of issues, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *requirement analysis step* if and only if there exist R_1, R_2, J, and U, such that $I(s) = R_1 \cup J \cup U$ and $O(s) = R_2$.

In steps of definition 24, there are four software component sets, R_1, R_2, J, and U. R_1 is a primary input node. J and U are secondary input nodes. R_2 is the result of this step.

Definition 25. (Specification Design Step) Let S_1 be a set of old-version specifications, S_2 be a set of new-version specifications, R be a set of requirements, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *specification design step* if and only if there exist S_1, S_2, R, and U, such that $I(s) = S_1 \cup R \cup U$ and $O(s) = S_2$.

In steps of definition 25, there are four software component sets, S_1, S_2, R, and U. S_1 is a primary input node. R and U are secondary input nodes. S_2 is the result of this step.

Definition 26. (Module Implementation Step) Let M_1 be a set of old-version specifications, M_2 be a set of new-version specifications, S be a set of specifications, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *module implementation step* if and only if there exist M_1, M_2, S, and U, such that $I(s) = M_1 \cup S \cup U$ and $O(s) = M_2$.

In steps of definition 26, there are four software component sets, M_1, M_2, S, and U. M_1 is a primary input node. S and U are secondary input nodes. M_2 is the result of this step.

Definition 27. (Program Integration Step) Let P_1 be a set of old-version software prototype programs, P_2 be a set of new-version software prototype programs, M be a set of modules, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *program integration step* if and only if there exist P_1, P_2, M, and U, such that $I(s) = P_1 \cup M \cup U$ and $O(s) = P_2$.

In steps of definition 27, there are four software component sets, P_1, P_2, M, and U. P_1 is a primary input node. M and U are secondary input nodes. P_2 is the result of this step.

Definition 28. (Software Product Demo Step) Let K_1 be a set of old-version optimizations, K_2 be a set of new-version optimizations, P be a set of software product programs, T be a set of software test scenarios, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *software product demo step* if and only if there exist K_1, K_2, P, T, and U, such that $I(s) = K_1 \cup P \cup T \cup U$ and $O(s) = K_2$.

In steps of definition 28, there are five software component sets, K_1, K_2, P, T, and U. K_1 is a primary input node. P, T, and U are secondary input nodes. K_2 is the result of this step.

Definition 29. (Software Product Implementation Step) Let P_1 be a set of old-version software product programs, P_2 be a set of new-version software product programs, K be a

set of optimizations, and U be a set of stakeholders. Let $H = (N, E, I, O)$ be a relational hypergraph where $N = I(s) \cup O(s)$ and $s \in E$. We say step s is a *software product implementation step* if and only if there exist P_1, P_2, K, and U, such that where $I(s) = P_1 \cup K \cup U$ and $O(s) = P_2$.

In steps of definition 29, there are four software component sets, P_1, P_2, K, and U. P_1 is a primary input node. K and U are secondary input nodes. P_2 is the result of this step.

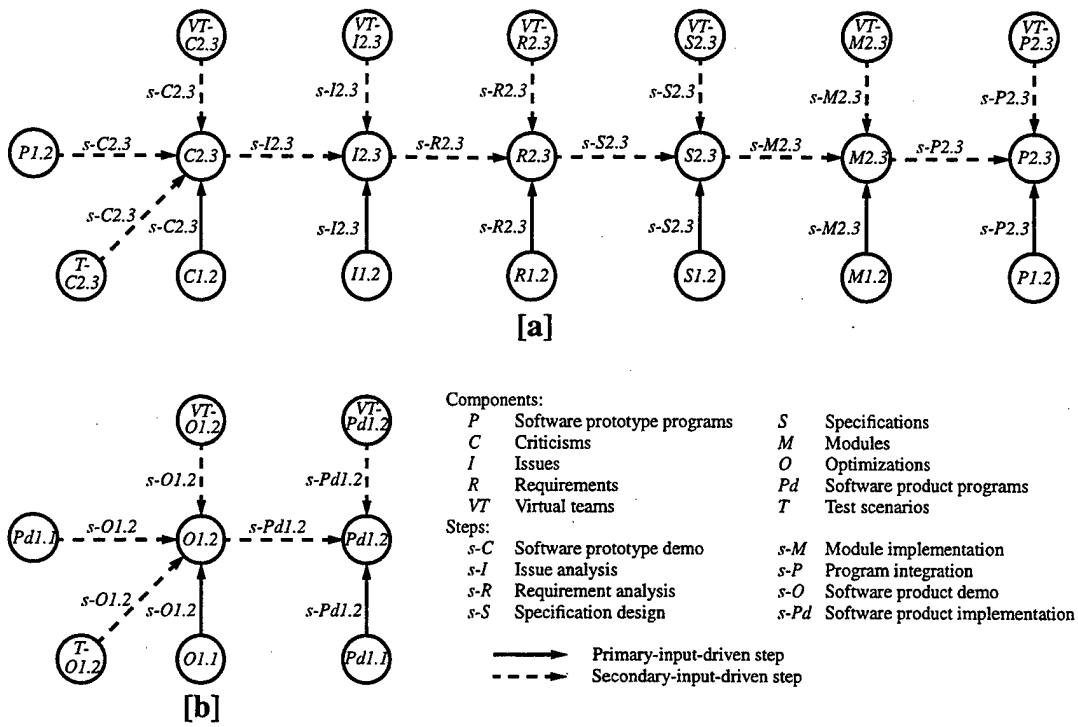


Figure 9: Software evolution steps with their related components

In Figure 9, [a] and [b] are secondary-input-driven hypergraphs that are formed by a series of software evolution steps with their related primary inputs and secondary inputs, such as virtual teams (or stakeholders), software test scenarios and so on. In [a], an old version of software prototype program *P1.2* will be upgraded into a new version of software prototype program *P2.3* via a series of software prototype evolution steps that are

software prototyping demo step *s-C2.3*, issue analysis step *s-I2.3*, requirement analysis step *s-R2.3*, specification design step *s-S2.3*, module implementation step *s-M2.3*, and program integration step *s-P2.3*. In [b], an old version of software product program *Pd1.1* is going to be modified into a new version of software product program *Pd1.2* via s series of software product evolution steps, such as product demo step *s-O1.2* and product implementation step *s-Pd1.2*. Arrows with the same name are the same step. For example, step *s-C2.3* has five software evolution components linked with arrows: three secondary input nodes *VT-C2.3*, *P1.2*, *T-C2.3*, one primary input node *C1.2*, and one output node *C2.3*.

D. SOFTWARE EVOLUTION PROCESS

The model of the software evolution process is based on the RH model and the IBIS model. The RH model provides a primary and secondary input driven mechanism to drive the software evolution process via a sequence of individual activities. The IBIS model relates the design rationale to the artifacts created during the systems development process [RAME95]. Therefore, the model of software evolution process can describe a secondary input driven mechanism in software evolution process well and provide another aspect from the original software evolution description based on the primary input driven mechanism [BADR94].

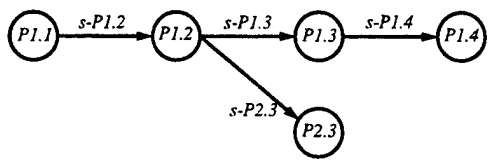


Figure 10: The software evolution graph of an object driven by primary input

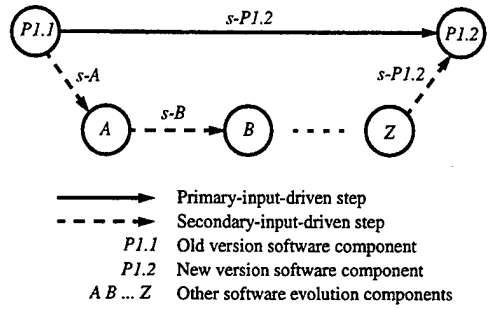


Figure 11: The software evolution graph driven by secondary input between two specified software components

A software component has to be modified from an old version to a new version due to social, political, or cultural factors. The software evolutionary hypergraph of a component driven by primary input can be shown as Figure 10. There are a lot of continuous software evolution activities driven by secondary input between two specified software components, as illustrated in Figure 11.

According to the Schematic Model of the Analysis Process modified from IBIS model in [IBRA96], we identify software prototype evolution process, software product generation process, and software evolution process by means of the links among eight types of software evolution steps and nine types of software evolution components as follows [HARN99f]:

Definition 30. (Software Prototype Evolution Process) Let $H = (N, E, I, O)$ be a relational hypergraph. Let m be the number of evolution times and path p be a sequence $p_1 \ldots p_t$ of paths driven by secondary input, where for each $m = 1, \ldots, t$ path p_m from a node n of an old version of prototyping software component to a node n' of a new version of prototyping software component is a sequence $e_1 \ldots e_6$ of hyperedges and a sequence $n_0 \ldots n_6$ of nodes such that $n_{i-1} \in I(e_i)$ and $n_i \in O(e_i)$ for $i = 1, \ldots, 6$, where $n = n_0$ and

$n' = n_6$. We say that hypergraph H is a *software prototype evolution process* if and only if there exists a path p such that the following assumptions satisfy:

1. Hyperedges $e_1 \ldots e_6$ are identifiers for the following steps: software prototype demo, issue analysis, requirement analysis, specification design, module implementation, and program integration, respectively.
2. Nodes $n_0 \ldots n_6$ are identifiers for the following components: old versions of programs, criticisms, issues, requirements, specifications, modules, and new versions of programs, respectively.
3. Let e_i^m be a hyperedge e_i, where $i = 1, \ldots, 6$, in the path p of the mth evolution. For $m = 1$ and each $i = 1, \ldots, 6$, there exist a hyperedge e_i^m, nodes n_{i-1}^m, n_i^m, and n_i^{m-1}, where $n_{i-1}^m \in I(e_i^m)$ and $n_i^m \in O(e_i^m)$, such that $n_0^m = n_6^{m-1} \in I(e_6^m)$. For each $m = 2, \ldots, t$ and $i = 1, \ldots, 6$, there exist a hyperedge e_i^m, nodes n_{i-1}^m, n_i^m, and n_i^{m-1}, where $n_{i-1}^m \in I(e_i^m)$ and $n_i^m \in O(e_i^m)$, such that $n_i^{m-1} \in I(e_i^m)$.

Definition 31. (Software Product Generation Process) Let $H = (N, E, I, O)$ be a relational hypergraph. Let m be the number of evolution times and path q be a sequence $q_1 \ldots q_t$ of paths driven by secondary input, where for each $m = 1, \ldots, t$ path q_m from a node n of a firm prototyping software component to a node n' of a software product component is a sequence $e_1 \ldots e_2$ of edges and a sequence $n_0 \ldots n_2$ of nodes such that $n_{i-1} \in I(e_i)$ and $n_i \in O(e_i)$ for $i = 1, 2$, where $n = n_0$ and $n' = n_2$. We say that hypergraph H is a *software product generation process* if and only if there exists a path q such that the following assumptions are satisfied:

1. Hyperedges e_1 and e_2 are identifiers for software product demo step and software product implementation step, respectively.

2. Nodes $n_0 \ldots n_2$ are identifiers for the following components: new versions of software prototyping programs or old versions of software product programs, optimizations, and new versions of software product programs, respectively.
3. Let e_i^m be a hyperedge e_i, where $i = 1, 2,$ in the path q of the mth evolution. For $m = 1$ and each $i = 1, 2$, there exist a hyperedge e_i^m, nodes n_{i-1}^m, n_i^m, and n_i^{m-1}, where $n_{i-1}^m \in I(e_i^m)$ and $n_i^m \in O(e_i^m)$, such that $n_0^m = n_2^{m-1} \in I(e_2^m)$. For each $m = 2, \ldots, t$ and $i = 1, 2$, there exist a hyperedge e_i^m, nodes n_{i-1}^m, n_i^m, and n_i^{m-1}, where $n_{i-1}^m \in I(e_i^m)$ and $n_i^m \in O(e_i^m)$, such that $n_i^{m-1} \in I(e_i^m)$.

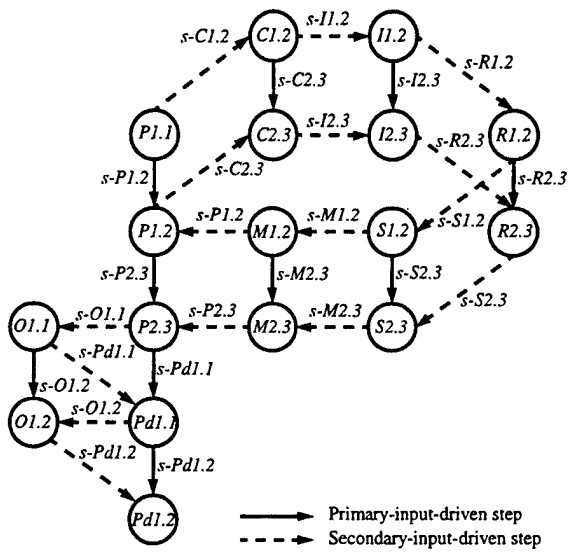
Definition 32. (Software Evolution Process) Let $H = (N, E, I, O)$ be a relational hypergraph. Let path p^k be a sequence $p_1 \ldots p_t$ of paths driven by secondary input in software prototyping evolution process and path q^k be a sequence $q_1 \ldots q_t$ of paths driven by secondary input in the software product generation process, where $k = 1, \ldots, n$. We say that hypergraph H is a *software evolution process* if and only if the following assumptions are satisfied:

1. The structure of hypergraph H combines software prototyping evolution process and software product generation process.
2. There is a path P such that $P = p^1 q^1 \ldots p^n q^n$.

Figure 12 shows an example of the software evolution process driven by primary input and secondary input. The first process from node *P1.1* to node *P2.3* is a software prototyping evolution process (evolution times m= 2) and the second process from node *P2.3* to node *Pd1.2* is a software product generation process (evolution times $m = 2$).

First of all, node *P1.2* is evolved in the first prototyping evolution from node *P1.1* and node *M1.2* via step *s-P1.2,* where node *M1.2* is the result of a serious steps, *s-C1.2, s-I1.2, s-R1.2, s-S1.2, s-M1.2*. Next, node *P2.3* is evolved in the second prototyping evolution

from node *P1.2* and node *M2.3* via step *s-P2.3,* where node *M2.3* is the result of serious steps, *s-C2.3, s-I2.3, s-R2.3, s-S2.3, s-M2.3.* Third, the node *Pd1.1* is evolved in the first software product evolution from node *P2.3* and node *O1.1* via step *s-Pd1.1,* where node *O1.1* is the result of a steps *s-O1.1.* Finally, the node *Pd1.2* is evolved in the second software product evolution from node *Pd1.1* and node *O1.2* via step *s-Pd1.2,* where node *O1.2* is the result of a steps *s-O1.2.*



Components:

P	Software prototype programs	*S*	Specifications
C	Criticisms	*M*	Modules
I	Issues	*O*	Optimizations
R	Requirements	*Pd*	Software product programs

Steps:

s-C	Software prototype demo	*s-M*	Module implementation
s-I	Issue analysis	*s-P*	Program integration
s-R	Requirement analysis	*s-O*	Software product demo
s-S	Specification design	*s-Pd*	Software product implementation

Figure 12: Software evolution process driven by primary and secondary inputs

E. RELATIONAL HYPERGRAPH NET

The relational hypergraph of a software evolution process is a very complicated structure, especially for using different kinds of inputs to a hyperedge. The secondary inputs to an atomic step usually make the relational hypergraph chaotic when the software system is huge and complex. We use the relational hypergraph net and its formal notation to describe and record the software evolution process.

The relational hypergraph net is a relational hypergraph which transfers a primary input hypergraph and secondary input hypergraphs into a top-level evolutionary hypergraph and an atomic evolutionary hypergraph. The relational hypergraph net is composed of a top-level step and a set of atomic steps together with their primary input node(s), related secondary input nodes and produced output node. Under a top-level evolutionary hypergraph, the input nodes to an atomic step can be shared by another atomic step to be input nodes, but the output node to an atomic step cannot be shared by any other atomic step. Therefore, the relational hypergraph net can be defined as follows:

Definition 33. (Top-level Relational Hypergraph Net) Let $H = (N, E, I, O)$ be a relational hypergraph. Let A be a set of primary input nodes, $B_1, ..., B_n$ be sets of secondary input nodes, and C be a set of output nodes to a top-level evolution step $s \in E$ in H, where $A, B_1, ..., B_n, C \subset N$ and $n \geq 0$. We say H is a *top-level relational hypergraph net* if and only if for each evolution step s, there exist a set A of primary input nodes, sets $B_1, ..., B_n$ of secondary input nodes, and a set C of output nodes to a top-level evolution step s such that $A \cup B_1 \cup ... \cup B_n = I(s)$ and $C = O(s)$. We call s together with $A \cup B_1 \cup ... \cup B_n = I(s)$ and $C = O(s)$ *top-level SPIDER s (Step Processed in Different Entrance Relationships).*

Definition 34. (Atomic Relational Hypergraph Net) Let $H = (N, E, I, O)$ be a relational hypergraph. Let C be a set of output nodes, A be a set of primary input nodes and $B_1, ..., B_n$ be sets of secondary input nodes to a top-level evolution step $s \in E$ in H, where

$A, B_1, ..., B_n, C \subset N$ and $n \geq 0$. We say H is an *atomic relational hypergraph net* if and only if for each top-level evolution step s and each atomic evolution step $s_i \in s$ where $i \geq 1$, there exist a set $A' \subseteq A$ of primary input nodes, sets $B_1' \subseteq B_1, ..., B_n' \subseteq B_n$ of secondary input nodes and an output node $c \in C$ to an atomic evolution step s_i such that $A' \cup B_1' \cup ... \cup B_n' = I(s_i)$ and $c \in O(s_i)$. We call s_i together with $A' \cup B_1' \cup ... \cup B_n' = I(s_i)$ and $c \in O(s_i)$ *atomic SPIDER s_i (Step Processed in Different Entrance Relationships).*

A software evolution step with its input and output components forms a SPIDER. Each top-level, refined, and atomic level SPIDER is connected into a relational hypergraph net. An atomic level SPIDER is a minimal task unit that can be assigned to developers.

Definition 35. (Relational Hypergraph Net) Let $H = (N, E, I, O)$ be a relational hypergraph. Let $H^t = (N^t, E^t, I, O)$ be a top-level relational hypergraph net and $H^a = (N^a, E^a, I, O)$ be an atomic relational hypergraph net. We say that H is a *relational hypergraph net* if and only if $H^t \cup H^a$.

To avoid the chaotic lines shown in relational hypergraph and make the complicated relationships readable, we illuminate the relational hypergraph with relational hypergraph net which combines the two separated hypergraphs, top-level relational hypergraph net and atomic relational hypergraph net. The top-level relational hypergraph net describes the relationships not only among each top-level step and its input and output nodes but also among each composite node and its subnodes. The atomic relational hypergraph net describes the relationship among each atomic step and its input and output nodes. Each input node in the relational hypergraph net can be easily comprehended as well as how many steps and what steps use the input node.

The top-level SPIDER and atomic SPIDER is a minimal unit of which the relational hypergraph net is comprised. We can link each SPIDER together to weave the relational

hypergraph nets and separate each SPIDER to record the relationships among the step and its input and output components.

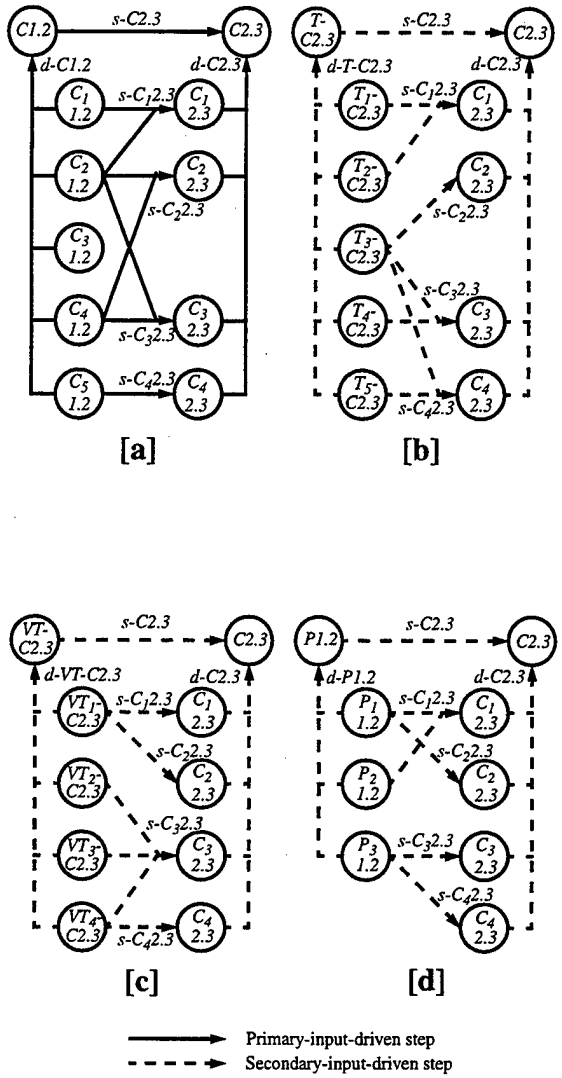


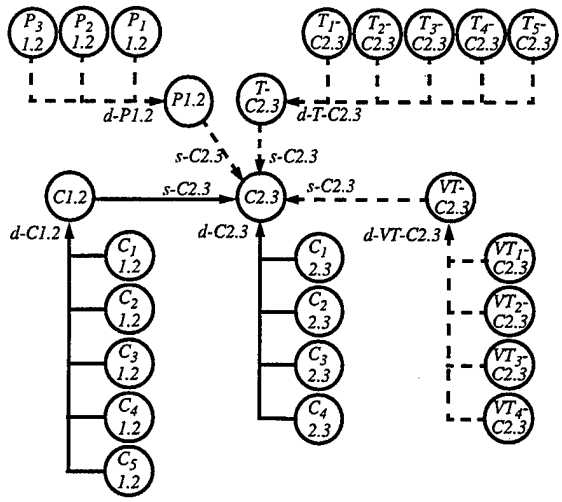
Figure 13: Separated relational hypergraphs

Figure 13 shows the four different separated relational hypergraphs: [a] is a primary-input-driven hypergraph from the evolution of the old version criticism *C1.2* to the new version criticism *C2.3* via step *s-C2.3*. Figure 13 [b], [c], and [d] are secondary-input-driven hypergraphs from the evolution of the test scenario *T-C2.3*, the virtual team

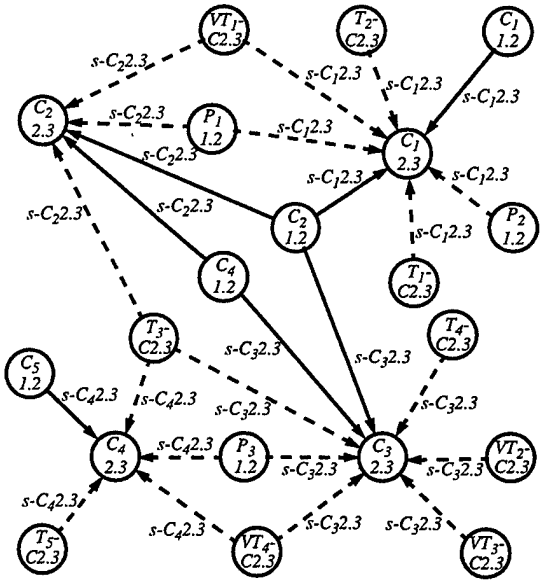
VT-C2.3, and the prototyping program *P1.2*, respectively, to the criticism *C2.3* via step *s-C2.3*. The old version criticism *C1.2* is a set of the old version atomic criticisms $C_1 1.2$, $C_2 1.2$, $C_3 1.2$, $C_4 1.2$, and $C_5 1.2$. New version criticism *C2.3* is a set of new version atomic criticisms $C_1 2.3$, $C_2 2.3$, $C_3 2.3$, and $C_4 2.3$. Test scenario *T-C2.3* is a set of atomic test scenarios T_1-*C2.3*, T_2-*C2.3*, T_3-*C2.3*, T_4-*C2.3*, and T_5-*C2.3*. Virtual team *VT-C2.3* is a set of atomic virtual teams VT_1-*C2.3*, VT_2-*C2.3*, VT_3-*C2.3*, and VT_4-*C2.3*. Prototyping program *P1.2* is a set of atomic prototyping programs $P_1 1.2$, $P_2 1.2$, and $P_3 1.2$. Step *s-C2.3* is a set of atomic step *s-*$C_1 2.3$, *s-*$C_2 2.3$, *s-*$C_3 2.3$, and *s-*$C_4 2.3$ as well as decomposition steps *d-C1.2*, *d-T-C2.3*, *d-VT-C2.3*, and *d-P1.2*.

The decomposition structure of separated relational hypergraphs is very easy to be understood but the relationships to an atomic step among different parts of relational hypergraph are difficult to be visualized due to the complicated input and output links. Two or more inputs to an atomic step can be briefly presented by an arrow with two or more tails, like input nodes $C_1 1.2$ and $C_2 1.2$ to atomic step *s-*$C_1 2.3$ in Figure 13 [a].

Figure 14 shows another way to use the relational hypergraph net to present the relational hypergraph in Figure 13. Figure 14 [a] is a top-level relational hypergraph net that has a top-level SPIDER *s-C2.3*, including step *s-C2.3* and refinement of input and output nodes to step *s-C2.3*. Figure 14 [b] is an atomic relational hypergraph net which has four atomic SPIDER*s* *s-*$C_1 2.3$, *s-*$C_2 2.3$, *s-*$C_3 2.3$, and *s-*$C_4 2.3$. Atomic SPIDER *s-*$C_1 2.3$ includes step *s-*$C_1 2.3$ as well as its input nodes VT_1-*C2.3*, T_2-*C2.3*, $C_1 1.2$, $P_2 1.2$, T_1-*C2.3*, $C_2 1.2$, $P_1 1.2$, and output node $C_1 2.3$. Atomic SPIDER *s-*$C_2 2.3$ includes step *s-*$C_2 2.3$ as well as its input nodes VT_1-*C2.3*, $P_1 1.2$, $C_2 1.2$, $C_4 1.2$, T_3-*C2.3*, and output node $C_2 2.3$. Atomic SPIDER *s-*$C_3 2.3$ includes step *s-*$C_3 2.3$ as well as its input nodes T_4-*C2.3*, VT_2-*C2.3*, VT_3-*C2.3*, VT_4-*C2.3*, $P_3 1.2$, T_3-*C2.3*, $C_4 1.2$, $C_2 1.2$, and output node $C_3 2.3$. Atomic SPIDER *s-*$C_4 2.3$ includes step *s-*$C_4 2.3$ as well as its input nodes VT_4-*C2.3*, T_5-*C2.3*, $C_5 1.2$, T_3-*C2.3*, $P_3 1.2$, and output node $C_4 2.3$.



[a]



[b]

Primary-input-driven step

Secondary-input-driven step

Figure 14: Relational hypergraph net

The advantage of the relational hypergraph net is that both the refinement structure of the relational hypergraph and the relationships between steps and nodes are easy to comprehend. The trivial disadvantage of the relational hypergraph net is that it is hard to condense the two or more same steps into one arrow with two or more tails and arrange a SPIDER at the appropriate position to avoid the links tangling together, due to the space limitation in the diagram. But SPIDER is worth being applied because it makes the representation of hypergraph refinement clear.

The formal notation of the relational hypergraph is another aspect to describe the software evolution process. We use *production formula* to record the top-level and atomic relational hypergraph nets.

The information of a relational hypergraph net can be illustrated by the following three basic production formulae:

$SPIDER(s) = O(s) \leftarrow s(I(s))$,

$DECOMPOSITION(d(n)) = n \leftarrow d(n)(I(d(n)))$, and

$SPIDER(s(i)) = O(s(i)) \leftarrow s(i)(I(s(i)))$,

where s is a top-level step, $d(n)$ is a decomposition edge to node n, and $s(i)$ is an atomic step for each nodes i in output node set $O(s)$. This definition is presented as follows:

Definition 36. (Formal Notation of Relational Hypergraph Net) Let $RHN = (N, E, I, O)$ be a relational hypergraph net, $T\text{-}RHN = (N^t, E^t, I, O)$ be a top-level relational hypergraph net, and $A\text{-}RHN = (N^a, E^a, I, O)$ be an atomic relational hypergraph net, where $RHN = T\text{-}RHN \cup A\text{-}RHN$. Let s be a top-level evolution step in RHN which has a set of atomic steps $s(i)$ for $i \geq 1$, and $d(n)$ be a decomposition edge where $n \in (I(s) \cup O(s))$. We say that $RHN(s)$ is a *formal notation of relational hypergraph net of step s* where

- $RHN(s) = T\text{-}RHN(s) \cup A\text{-}RHN(s)$
- $T\text{-}RHN(s) = SPIDER(s) \cup$

$$\cup_{n \in (I(s) \cup O(s))} DECOMPOSITION(d(n))$$

- $A\text{-}RHN(s) = \cup_{s(i) \in s}\ SPIDER(s(i))$
- $SPIDER(s) = O(s) \leftarrow s(I(s))$
- $DECOMPOSITION(d(n)) = n \leftarrow d(n)(I(d(n)))$
- $SPIDER(s(i)) = O(s(i)) \leftarrow s(i)(I(s(i)))$

For example, Figure 14 can be described and recorded as following formal notations:

1. Relational hypergraph net of *s-C2.3* is

   $RHN(s\text{-}C2.3) = T\text{-}RHN(s\text{-}C2.3) \cup A\text{-}RHN(s\text{-}C2.3)$

2. Top-level relational hypergraph net of *s-C2.3* is

   $T\text{-}RHN(s\text{-}C2.3) = \{$

   $(C2.3 \leftarrow s\text{-}C2.3(C1.2, P1.2, T\text{-}C2.3, VT\text{-}C2.3)),$

   $(C2.3 \leftarrow d\text{-}C2.3(C_1 2.3, C_2 2.3, C_3 2.3, C_4 2.3)),$

   $(C1.2 \leftarrow d\text{-}C1.2(C_1 1.2, C_2 1.2, C_3 1.2, C_4 1.2, C_5 1.2)),$

   $(P1.2 \leftarrow d\text{-}P1.2(P_1 1.2, P_2 1.2, P_3 1.2)),$

   $(T\text{-}C2.3 \leftarrow d\text{-}T\text{-}C2.3(T_1\text{-}C2.3, T_2\text{-}C2.3, T_3\text{-}C2.3, T_4\text{-}C2.3, T_5\text{-}C2.3)),$

   $(VT\text{-}C2.3 \leftarrow d\text{-}VT\text{-}C2.3(VT_1\text{-}C2.3, VT_2\text{-}C2.3, VT_3\text{-}C2.3, VT_4\text{-}C2.3))\}.$

3. Atomic relational hypergraph net of *s-C2.3* is

   $A\text{-}RHN(s\text{-}C2.3) = \{$

   $(C_1 2.3 \leftarrow s\text{-}C_1 2.3(C_1 1.2, C_2 1.2, P_1 1.2, P_2 1.2, T_1\text{-}C2.3, T_2\text{-}C2.3, VT_1\text{-}C2.3)),$

   $(C_2 2.3 \leftarrow s\text{-}C_2 2.3(C_2 1.2, C_4 1.2, P_1 1.2, T_3\text{-}C2.3, VT_1\text{-}C2.3)),$

   $(C_3 2.3 \leftarrow s\text{-}C_3 2.3(C_2 1.2, C_4 1.2, P_3 1.2, T_3\text{-}C2.3, T_4\text{-}C2.3, VT_2\text{-}C2.3, VT_3\text{-}C2.3,$
   $VT_4\text{-}C2.3)),$

   $(C_4 2.3 \leftarrow s\text{-}C_4 2.3(C_5 1.2, P_3 1.2, T_3\text{-}C2.3, T_5\text{-}C2.3, VT_4\text{-}C2.3))\}.$

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPUTER-AIDED SOFTWARE EVOLUTION

A. COMPUTER-AIDED SOFTWARE EVOLUTION SYSTEM

This chapter describes a computer-aided software evolution tool that is based on the RH model. Due to different software application domains, development environments, and methodologies of requirements engineering, software evolution is currently not well understood. In our experience, completely formalizing a software evolution process for a large scale and complex software system, especially if one tries to include social, political, and cultural factors, is extremely difficult [SEIT98] [SOMM96].

There has been no efficient and standard software evolution process to support software system development in the past decade [SOMM96]. This study proposes a RH model with primary-input-driven and secondary-input-driven dependency approaches to illustrate the software evolution process. We give a standard software evolution process in developing a prototype system as well as a production software system. The RH model can also describe an informal software evolution process that developers explore under different software development and evolution environments.

In order to perform the automated software evolution, we have constructed a tool called CASES. CASES is a set of Java software that performs the following main functions: control, management, formation, refinement, traceability, and assignment. The structure of CASES is based on the RH model and interfaces to CAPS [HARN99d] to manage and control the software evolution process for iterative software prototyping [LUQI88a].

1. Software evolution description

There is no standard software evolution process which adapts to different developers' needs, especially for developing real-time embedded software systems. Generally, a software evolution process consists of a series of software evolution steps with their related input and output components. These software evolution steps and components

are called *software evolution objects*. Software evolution components include not only pure software components consisting of software codes, but also include other components, such as text, data, hypermedia, and so on. In the different development methods and environments, software evolution object types and numbers are extremely uncertain; therefore, the architecture of software evolution processes depends on different developers' needs. In the software evolution of a large, complex, and real-time embedded system, the prototyping method can be applied to grasp the users' requirements [LUQI88a] [LUQI89]. However, in different development environments, the prototyping process has various descriptions. For example, the evolution processes of C4I (command, control, communication, computer, and intelligence) systems using CAPS are different from those of MIS (management information systems) using CASE (computer-aided software engineering) tools. Making a general discipline of software evolution processes is not quite easy since a system assigned to different developers or software development companies have distinctive development processes. Therefore, it is difficult to construct a tool with general applicability. For developing a real-time embedded system, we have formalized an appropriate software evolution process that includes a software prototype evolution process and a software product evolution process. This formalization is extended from a modified IBIS model [BERZ97] [CONK88].

2. Preliminary software evolution process

Basically, the *software evolution processes* using CAPS are two iterative processes: a *software prototype evolution process* and a *software production generation process*. The software prototype evolution process repeats a guess/check/modify cycle until the users agree that the demonstrated behavior is acceptable [LUQI88a] [LUQI88b] [LUQI89] [LUQI90] [BERZ93]. The software production generation process repeats a cycle that optimizes and implements production codes from the final results of the software prototype evolution process. These two processes can be depicted by two iterative loops shown in Figure 15. Figure 15 [a] is a software prototype evolution process built by a series

of software evolution steps: *s-R, s-S, s-M,* and *s-P* with related software evolution components *R* (requirements), *S* (specifications), *M* (modules), and *P* (software prototype programs), where *s-R, s-S, s-M,* and *s-P* represent the software prototype demo step, the specification design step, the module implementation step, and the program integration step respectively. Figure 15 [b] is a software production generation process built by a series of software evolution steps: *s-O, s-Pd* with related software evolution components *O* (optimizations) and *Pd* (software production programs), where *s-O* and *s-Pd* represent the software product demo step and the software product implementation step respectively. Figure 15 [c] is a software evolution process that is comprised of Figure 15 [a] and [b]. During the software evolution processes, the final version of *P* in Figure 15 [a] can trigger the iterative loop in Figure 15 [b] if *P* needs to be transformed into a software product; and the final version of *Pd* in Figure 15 [b] can trigger the iterative loop in Figure 15 [a] if *Pd* needs to be evolved to another generation.

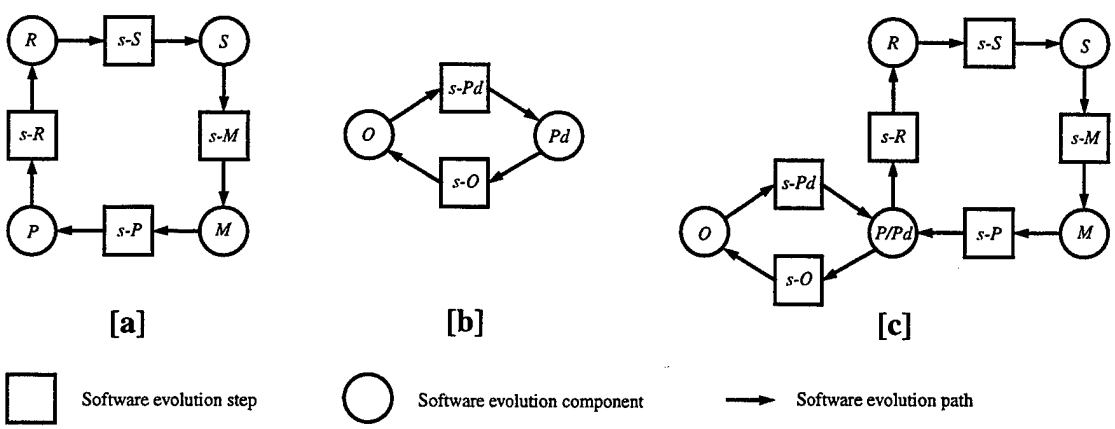


Figure 15: Different software evolution processes

In software evolution processes, software evolution objects are associated with unique version identifiers. Version identifiers contain a variant and a version number. The current version of software evolution objects is evolved from an older version. For example

the first iteration of software evolution starts from *s-R* via *R, s-S, S, s-M, M,* and *s-P* to *P* with version *1.1*, which has the variant *1* and the version number *1*. If the variant in the second iteration of software evolution is the same as in the first iteration, then the version identifiers of the software evolution objects are *1.2*.

3. Formalized software evolution process

In the above preliminary software evolution process, it seems that stakeholders cannot immediately obtain requirements efficiently from the software prototype demo step. We have to add some software evolution steps and components to Figure 15 [c] to enhance the capacity to grasp user requirements. According to the interactive, evaluation-centered user interaction development process [HIX93] and the Schematic Model of the Analysis Process [BERZ97] modified from the IBIS model [CONK88], we have identified eight types of *top-level steps* in the software evolution process to address this drawback: software prototype demo, issue analysis, requirement analysis, specification design, module implementation, program integration, software product demo, and software product implementation. We have also identified eight different types of top-level components in the software evolution process: criticisms, issues, requirements, specifications, modules, software prototype programs, optimizations, and software product programs. Each top-level object can be decomposed into a set of atomic objects, either directly or indirectly. This formalization of the modified software evolution process [HARN99f] obtains user requirements via software prototype demo, issue analysis, and requirement analysis steps as shown in Figure 16. Typically, the software evolution processes can be changed according to development environment needs.

4. CASES functions

A computer-aided software evolution system, CASES, can manage and control all of the activities that change a software system and the relationships among these activities. The relationship among CASES and the whole process for software evolution is shown in Figure 16.

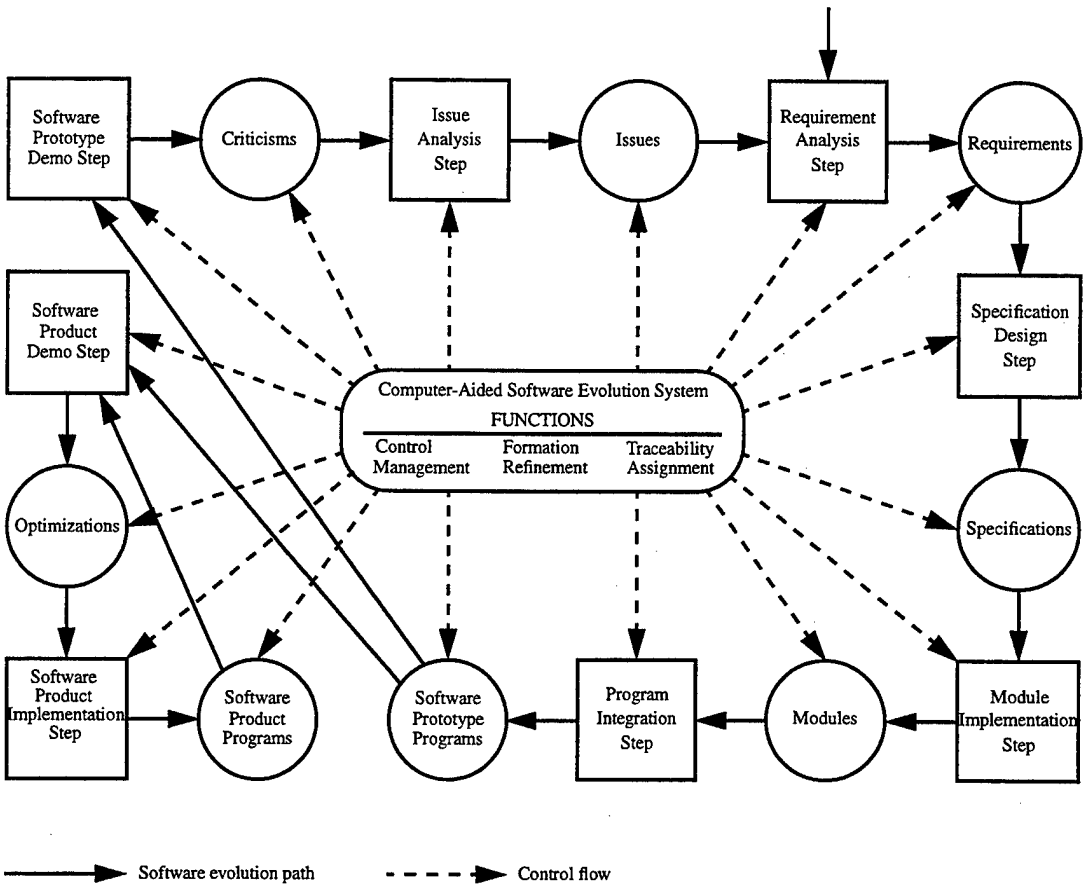


Figure 16: Software evolution processes with CASES

Based on the definitions of the RH model in Chapter III, an appropriate design of CASES at least includes the following basic functions: control, management, formation, refinement, traceability, and assignment. We conduct CASES functions according to the above directions.

CASES has five common functions related to the software evolution activities: step refinement, project evaluation, constraint management, personnel management and step management. CASES has five functions that are related to the software evolution components: component management, component traceability, version control and configuration management, dependency management, and inference rule management.

The following description focuses on some top-level software evolution steps:

a. Step refinement

The software evolution top-level step can be refined into a set of atomic steps. Atomic steps are the basic job execution unit of a software evolution step. The principle of atomic step is that every atomic step has a group of input components and only one output component. Based on this principle, the software evolution components under a top-level step can be classified into several groups of the input components to atomic steps by project organizers.

b. Project evaluation

After project organizers propose an evolution step as a project, project evaluators will evaluate this project according to the risk assessment of executing this software evolution step. The project evaluators perform cost, benefit and impact analysis, and evaluate the validity of the deficiency report that motivates the proposed step.

The evaluator enters evaluation results into CASES to aid management when they decide whether the step should be approved. The whole evaluation process is realized via an interactive user interface.

c. Constraint management

The project organizer sets constraints that affect the scheduling of steps, such as predecessors, priorities, deadlines, estimated duration, earliest start times, finish times, and constraints that affect personnel assignments, such as security level and skill requirements for a step. CASES manages constraints on atomic steps via inference rules and aggregate decisions made by project organizers.

d. Personnel management

Project organizers control the current status of the project personnel such as skill, skill level, security level, on-hand jobs, and so forth. The personnel data would be adjusted after system analysts or designers finish an atomic step by the project team leader.

The performance of software engineers is the fundamental reference for job assignment and promotion.

e. ***Step management***

The content of the top-level step can be automatically generated, refined, and queried. The content of the atomic step can also be automatically generated, combined, and queried. Stakeholders can manage and trace any step of software evolution component under the whole software evolution process.

f. ***Component management***

Stakeholders can enter, delete, retrieve, modify, and query the attributes of atomic component from the hypertext database or software library (including software base and design database).

g. ***Component traceability***

The stakeholder can trace an atomic component generated by its source atomic step via the following two components: primary input components and secondary input components.

h. ***Version control and configuration management***

The version and variation number of output components of a step are automatically determined by a labeling function of CASES. The software evolution process loops of CASES automatically construct the configuration management.

i. ***Dependency management***

The dependencies among atomic components to an atomic step can be identified and managed. CASES generates some dependencies, like *affect/used_by* associated with the relationship of primary and secondary input as well as *part_of* associated with the relationship of hyperedge and node refinement and stakeholders manually identify some dependencies.

j. Inference rule management

The stakeholders can specify and adjust inference rules related to SPIDER formation, scheduling and assignment constraints, policies, special assignments, and so on, to help them resolve the design and management issues of the software development process.

5. Evolution process using CASES and CAPS

At the beginning of the evolution process, users or system designers design the first version of a prototype of their proposed system using CAPS according to the user requirements managed by CASES. The steps of this process follow:

a. Software prototype demo step

After running the current version of the software prototype or product demo, customers record and send their criticisms to CASES via the network which can directly connect to the CASES hypertext database. This can be done using the Front Loaded Accurate Requirements Engineering (FLARE) system [LEON97].

b. Issue analysis step

System analysts collect and classify the criticisms and associate them with the related issues with the help of browsing and search capabilities of the hypertext database and communication with stakeholders as needed to clarify the intent of recorded criticisms.

c. Requirement analysis step

System analysts collect and classify the issues and refine the related requirements with database and communication support. This is a creative process that involves proposing and assessing plausible alternatives for responding to open issues.

d. Specification analysis step

System designers modify the PSDL corresponding to new requirements via the graphic user interface and editor of CAPS.

e. Module implementation step

System designers modify or implement the modules corresponding to the new PSDL via the graphic user interface and editor of CAPS.

f. Program integration step

System designers modify or implement the programs corresponding to the new modules and integrate them using CAPS. After the program integration step, the system prototype has evolved to the next version and can be demonstrated again to customers to assess its acceptability.

g. Software product demo step

The evolution process of prototyping systems can continue for many iterations until the final version of the prototype matches the customer's real requirements. After the demo step for the final prototyping system, software engineers evaluate the target operating environment and obtain the optimizations for the proposed software products.

h. Software product implementation step

System designers carry out proposed optimizations to design the deliverable software products.

After the software product implementation step, the software products can be delivered to customers and be used by users. During the lifecycle of the software system, users may develop more criticisms, which are submitted according to appropriate policies. If the criticisms exceed some threshold, the project office may consider a maintenance upgrade. If a maintenance project is authorized, the process of software evolution will be started from the demo step again.

6. Dynamic state model of evolution steps

The dynamic state model of evolution steps in [BADR93] and [BADR94] includes six states for a software evolution step: *Proposed, Approved, Scheduled, Assigned, Completed,* and *Abandoned.* The state transition diagram for software evolution steps can be shown as Figure 17. In the Proposed state, a proposed evolution step is subjected to both cost and benefit analysis. This analysis also includes identifying the software objects comprising the input set of the step. In the Approved state, the proposed step has been approved but not yet scheduled, and the input set of the step is not bound to particular versions. Approval of a proposed step by the management triggers the decomposition process to create an atomic step for each primary or secondary input component for the step. In the Scheduled state, the approved step has been scheduled but not assigned to a designer. The scheduling mechanism produces an updated schedule containing the newly-scheduled step. In the Assigned state, the scheduled step is assigned to the scheduled designer automatically from the scheduled state. When a designer is available, the schedule is used to determine his or her next assignment. In the Completed state, the outputs of the step have been verified and approved for release. This is the final state for each successfully completed step. In the Abandoned state, the step has been cancelled before it has been completed. The outputs of the step do not appear as components in the evolution history graph.

This initial model lacks the means to decompose input software evolution components if they are determined to be composite components. If we do not modify the dynamic states, the proposed decomposition of the software evolution step in dynamic state model shown in Figure 17 can be described as follows:

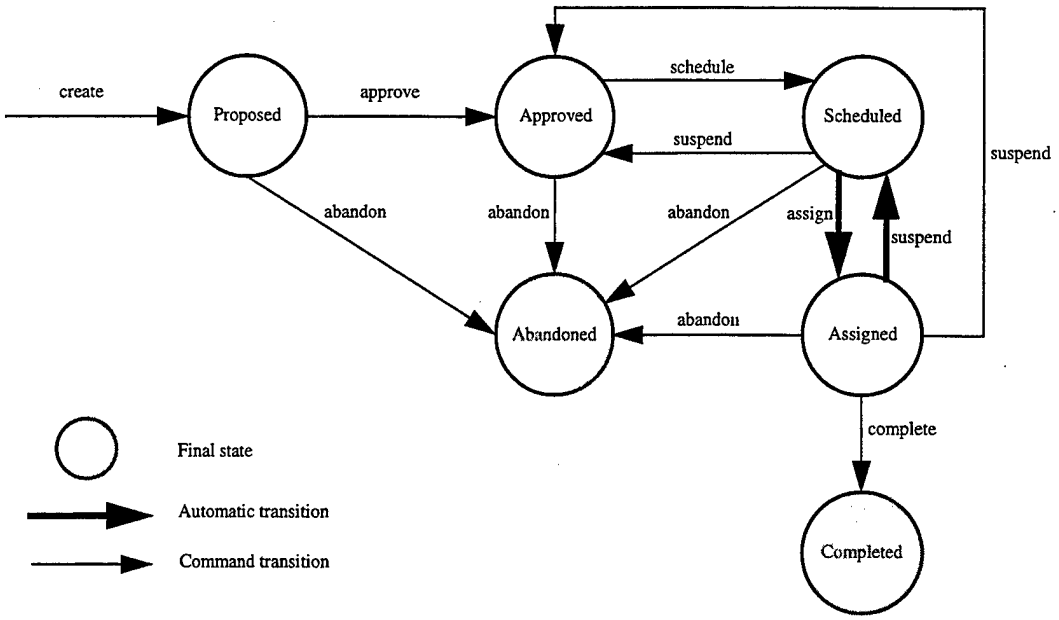


Figure 17: State transition diagram for software evolution steps

When a step reaches the Assigned state, the designer may determine that the step should be decomposed into two or more refined steps. The assigned designer may complete the decomposition steps or determine that the decomposition must be evaluated and reassigned. If the designer completes the decomposed software evolution step without reassignment, then the transition form Assigned to Completed occurs and the Completed state retains that same meaning. If the designer proposes decomposition with reassignment, then the software evolution step transitions to the Completed state with a new issue submitted to evaluate the proposal. The Completed state represents the multiple result possibilities for a software evolution step to reach its completion; therefore, the transition from Assigned to Completed is now labeled complete. The previous label for the transition was commit, implying that the development was finished, and then the output software evolution component is approved for release.

In order to capture the management decision, the dynamic model is extended into seven states. We add a *Decomposed* state and the events: decompose, approve, and

disapprove, for the proposed decomposition. The proposed decomposition of the software evolution step in the dynamic state model shown in Figure 18 can be described as follows:

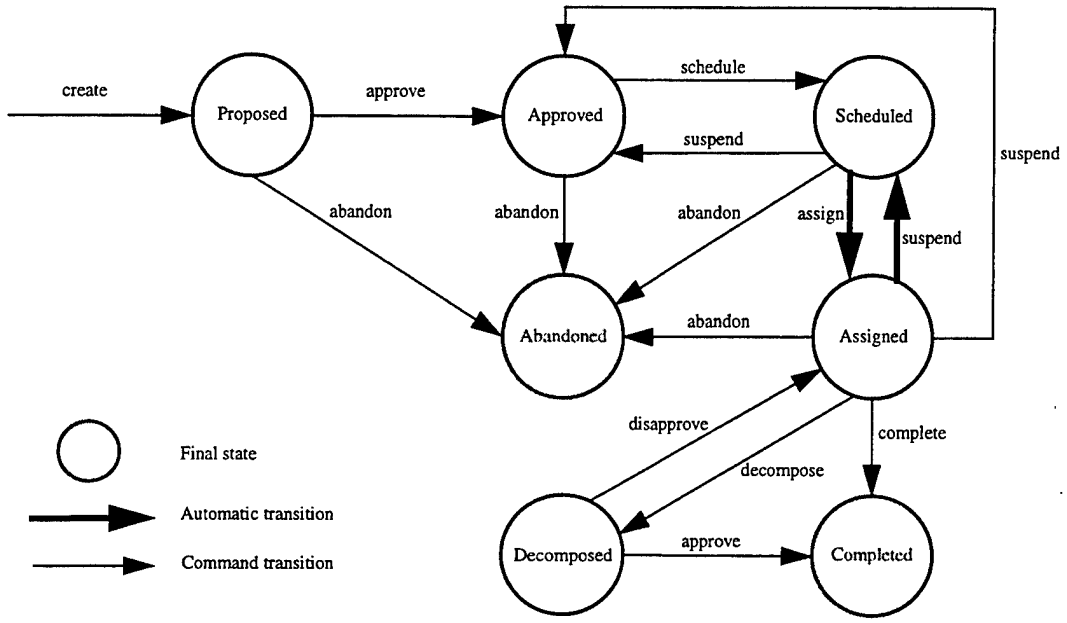


Figure 18: State transition diagram for software evolution steps (extended)

The Decomposed state is reached when an assigned designer determines that a step must be composite, and that new decomposed atomic steps are required for reapproval, rescheduling, and reassigning. At this time, the designer suspends development of the composite step and a decision must be made to determine if the decomposition of the step is warranted. If the decomposition of the step is disapproved, then the designer has to complete this step. If the decomposition of the step is approved, then the composite step is transferred to the Completed state and the decomposed atomic steps are transferred into the Proposed state.

Actually, software evolution steps in a dynamic state model are atomic SPIDERs. The atomic SPIDER is the basic unit to the task assignment. The CASES

automatically assigns the atomic SPIDER to a system analyst or a system designer. The manager decides the level of difficulty for each skill in the atomic SPIDER.

7. Project team organization

a. Classification of stakeholder roles

In the software evolution process, customers and software engineers are the primary stakeholders. We have identified two roles of *customers* and five roles of *software engineers* shown in Figure 19. The roles of customers include *system owners* and *end users*. The roles of software engineers include *project team leaders/managers, project organizers, project evaluators, system analysts,* and *system designers*.

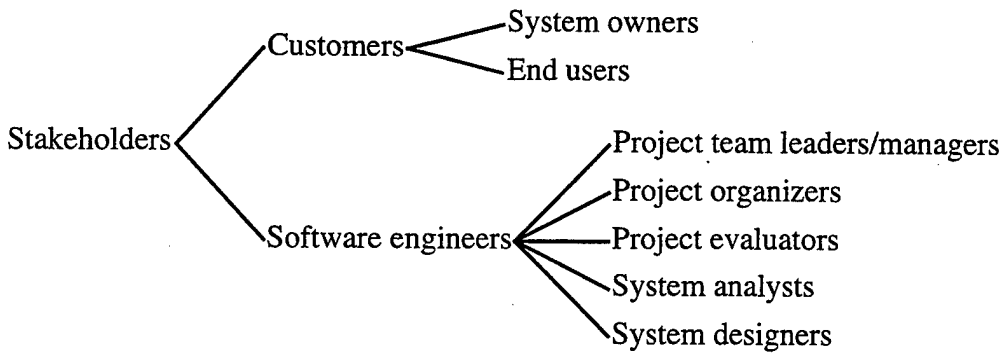


Figure 19: Stakeholder classification

The system owner is a sponsor who supports the software development project and owns the result of the developed software. The end user is a person who uses the software product and manipulates the software system. The project team leaders/ managers lead the members of the project team: project organizers, project evaluators, system analysts and system designers, and they manage the progress of the evolution steps.

The project organizers are responsible for organizing a software project including the following activities:

- create a project and define software evolution object types under a specific

software project,

- modify definitions of software evolution object types under a specific software project,
- create or modify software evolution processes under a specific software project,
- define or modify dependencies among software evolution objects,
- create a new step version under a specific software project.
- explore and manage required skills of projects,
- manage the required skills and levels, and the security level of a stakeholder,
- organize an atomic SPIDER as a job and propose the job with scheduling, skill, and security constraints to a project evaluation team, and
- schedule and assign a job to a project analysis team or a project design team.

The project evaluators are responsible for evaluating the software project including the following activities:

- evaluate and modify software evolution processes under a specific software project,
- evaluate and upgrade security levels, required skills and levels for stakeholders,
- evaluate the formation of an atomic SPIDER with the scheduling, skill, and security constraints proposed by project organizers or system designers,
- make the risk assessment and the failure impact evaluation for a job.

The system analysts assume the responsibility of completing the analysis steps of software evolution, such as the criticism analysis step, the issue analysis step, and the requirements analysis step. The system designers complete the design step of software evolution, such as the specification design step, the module implementation step, the program integration step, the software prototype demo step, the software product implementation step, and the software product demo step.

Figure 20: Organization structure of project teams

b. Organization structure of project teams

Generally, there are many project teams in a software development department. This depends on the scale of the organizational structure of the software development department. The *chief information officer (CIO)* is a top level leader/manager who monitors the entire software development process and manages the administration of project teams. In CASES, there are four kinds of project teams: *the project organization team, the project evaluation team, the system analysis team* and *the system design team* shown in Figure 20. Each project team has a project team leader/manager and members. One person could be in different teams because he or she could be a project organizer, project evaluator, a system analyst, and a system designer simultaneously. Typically, the project organization teams include project team leaders/managers and project organizers. The project evaluation teams include project team leaders/managers and project evaluators. The system analysis teams include project team leaders/managers and system analysts. The system design teams include project team leaders/managers and system designers.

B. REUSABLE ARCHITECTURE

This section explores the idea of component-based reuse of software development architecture. It includes: (1) an analysis of a domain-specific software development architecture, (2) the development of a component base (repository) that is robust with respect to system evolution, and (3) the implementation of a lightweight inference engine for automated decision support.

The study is aimed at gaining a framework for component-based reuse of software architecture, where a family of software systems sharing the same architecture is produced using common components. This embraces a component base (repository) equipped with a lightweight inference engine for software evolution and automated decision support for processes, i.e. component retrieval, version control, project management, and task decomposition.

1. Repositories of software evolution steps and components

The architecture of software evolution component reuse is based on the relational hypergraph net. The relational hypergraph net is structured after the stakeholders complete the related software evolution activities and stored it in the RH model base. The relationships among the software evolution steps and components are recorded and stored in the RH model base via the data entry of each SPIDER.

The relational hypergraph net shows the basic architecture of software evolution but some of the attributes of the software evolution steps are not described in the RH model base. We design a step database to store the attributes of the software evolution steps.

An output node of an atomic SPIDER shows the following information from the RH model base: the version and variant number and the source atomic step. Because each output node of an atomic SPIDER has different types of content, we design component content links to connect different content in component content repositories. The component content links are stored in the component content link database. The component

content links of the output node of an atomic SPIDER can be retrieved by indexing the version and variant number in the related component content link database.

A source atomic step of an atomic SPIDER shows the following information from the RH model base: the version and variant numbers, the top-level step, the input components, and the output component. The attributes about evaluations, scheduling constraints, and assignment constraints can be retrieved by the index of the version and variant number in the step database.

Based on the three categories of components, text, software code, and data, the content of a new output component to a step are stored as files in different component content repositories as follows:

- text component base: criticisms, issues, requirements, optimizations, and test scenarios,
- software component base: specifications and programs, and
- personnel database: stakeholders.

2. Lightweight inference engines and input component search engine

The lightweight inference engine assists software evolution and automated decision support for processes, namely component retrieval, version control, project management, and task decomposition. The stakeholder can easily assure and obtain the information associated with software evolution by using the inference rules. Due to specific decision support domains, the inference engine is designed with a lightweight scale [BERZ98] [HARN99].

The preliminary dependency rules of automated decision support for processes are created as follows:

$$ALL(s : S, c : C :: c\ output\ s \Leftrightarrow c \in output(s)) \quad (1)$$

$$ALL(c1\ c2 : C :: c1\ same\_objcet\ c2 \Leftrightarrow c1.version.object\text{-}id = c2.version.object\text{-}id) \quad (2)$$

$$ALL(s : S, c : C :: c\ input\ s \Leftrightarrow c \in input(s)) \quad (3)$$

$$ALL(s : S, c1\ c2 : C :: c1\ primary\_input\ s \Leftrightarrow c1\ input\ s\ \&\ c2\ output\ s\ \&\ c1\ same\_object\ c2) \quad (4)$$

$$ALL(s : S, c1\ c2 : C :: c1\ secondary\_input\ s \Leftrightarrow c1\ input\ s\ \&\ c2\ output\ s\ \&\ \neg(c1\ same\_object\ c2)) \quad (5)$$

An atomic SPIDER is a posterior result in the software evolution process. Before carrying out the atomic SPIDER, the stakeholders have to seek and reuse the related input components to the atomic SPIDER by the input component search engine that can be executed with a lightweight inference engine for inferring dependencies. The more input components to an atomic SPIDER obtained by means of the dependencies among the software evolution objects, the less the efforts of the stakeholders to construct an atomic SPIDER in the project plan.

After stakeholders record and save the data of an atomic SPIDER in the repository, the relationships among the software evolution objects in this atomic SPIDER can be structured automatically by inference rules. When a software evolution component is changed, new software evolution steps may have to be induced because a change in a component of a software evolution process may require changes in other components to maintain the consistency of the software system [LUQI90]. To seek and reuse the input components associated with the induced step for stakeholders, we trace the dependencies among the software evolution components with the inference rules to find the input scope of the induced step. The stakeholders use and refer to the input components automatically generated by the input component search engine to achieve a development step of a new version of a component.

3. Attribute and content retrieval engines

After getting the input component list from the input component search engine, the stakeholders can retrieve the step attributes with the attribute retrieval engine, and content of input components via the text component retrieval engine, software component retrieval engine, or personnel component retrieval engine.

The step attribute retrieval engine can access the basic attributes of software evolution steps from the step database.

The content of a text or software component occupies a large amount of space in the repository, so it is suitable to save as a file instead of database attributes. However, the content of a personnel component is represented by attributes of stakeholders.

The text component retrieval engine can access the contents of text components, such as criticisms, issues, requirements, optimizations, and test scenarios, from the text component base according to the component content links of a specified text component. Similarly, the software component retrieval engine can access the contents of the software components, that is, specifications and programs, from the software component base according to the component content links of a specified software component.

The personnel component retrieval engine can access the content of the personnel components, regarded as virtual teams or stakeholders, from the personnel database according to the version and variant number of a specified personnel component.

Before an atomic SPIDER is assigned to stakeholders, the step attribute retrieval engine will access the data of the steps, which include the needed security level, skills and skill levels, from the attributes of this atomic SPIDER. After that, the personnel component retrieval engine will access the attributes of the stakeholders from the personnel database to perform the job assignment.

4. Job assignment engine

The job assignment engine can be executed with a lightweight inference engine that is designed for job assignment. The function of the job assignment engine is to search a group of people who can achieve the software evolution activities in a specified atomic SPIDER. Based on the needed skills of an atomic SPIDER combined with skill levels, the job assignment engine can automatically search the appropriate stakeholders to carry out the software evolution activities of an atomic SPIDER. Basically, the security level and skills with skill levels of a stakeholder have to be recorded in the personnel database in advance. The needed security level and skills of an atomic SPIDER with skill levels can be stipulated by evaluators and saved in the step database. The job assignment search engine

obtains a group of candidates via the matching algorithms. The job assignment engine provides two approaches to choose the person from the candidates. First, the manager can interactively specify some people to achieve the atomic SPIDER. Second, the job assignment engine can automatically assign people to achieve the atomic SPIDER via the inference rules, which provide the job assignment knowledge and are driven by a lightweight inference engine.

5. Software evolution search functions

The software evolution search functions are designed as an interpreter to get the software evolution objects and their dependencies, to compute the number of objects in a net or step, and to evaluate properties in a relational hypergraph net. There are many kinds of search functions that can be used to understand the inside relational hypergraph net. Some of software evolution search functions related to the relational hypergraph net are as follows:

a. Relational hypergraph net search functions

- rhnet(n) : get the relational hypergraph net n
- tnet(n) : get the top-level relational hypergraph net n
- anet(n) : get the atomic relational hypergraph net n
- spider(c) : get a SPIDER of an output component c
- decomp(c) : get a decomposition of a composite component c

b. Component search functions

- icom(s) : get the input components to a step s
- picom(s) : get the primary input components to a step s
- sicom(s) : get the secondary input components to a step s
- ocom(s) : get the output component to a step s

c. Step search functions

- sstep(c) : get the source step of a component c
- rstep(c) : get the related steps regarding a component c as an input component
- astep(s) : get the atomic steps of a composite step s
- cstep(s) : get the composite step of an atomic step s
- acom(c) : get the atomic components of a composite component c
- ccom(c) : get the composite component of an atomic component c

- status(s) : get the current status of a step s

d. Dependency search functions

- dep(a, b) : get the dependency between objects a and b
- his(a, b) : get the history between objects a and b

e. Object number functions

- tnum(t) : get the number of steps in the top-level relational hypergraph net t
- anum(a) : get the number of steps in the atomic relational hypergraph net a
- inum(s) : get the number of input components to a step s
- pinum(s) : get the number of primary inputs to a step s
- sinum(s) : get the number of secondary inputs to a step s

f. Property evaluation functions

- step(s) : evaluate s is a step
- component(c) : evaluate c is a component
- input(c, s) : evaluate component c is an input component to a step s
- pinput(c, s) : evaluate component c is a primary input component of a step s
- sinput(c, s) : evaluate component c is a secondary input component of a step s
- output(c, s) : evaluate component c is an output component to a step s
- member(o1, o2) : evaluate object o1 is a member of a composite object o2.

Take Figure 14 in Chapter III for example. Before executing the search function, we should create the relational hypergraph net *RHN(s-C2.3)* in the RH model base. The following interpretations are part of the search functions:

$ spider($C_2 2.3$)

$(C_2 2.3 \leftarrow s\text{-}C_2 2.3(C_2 1.2,\ C_4 1.2,\ P_1 1.2,\ T_3\text{-}C2.3,\ VT_1\text{-}C2.3))$

$ sicom($s\text{-}C_2 2.3$)

$(P_1 1.2,\ T_3\text{-}C2.3,\ VT_1\text{-}C2.3)$

$ rstep($C_2 1.2$)

$(s\text{-}C_1 2.3,\ s\text{-}C_2 2.3,\ s\text{-}C_3 2.3)$

$ dep($C_2 1.2,\ C_3 2.3$)

primary_input

$ inum($s\text{-}C_2 2.3$)

5

$ input($C_2 2.3$, s-$C_2 2.3$)

F

The main contribution of software evolution via reusable architecture is that we have built a component reusable architecture to resolve the essential issues of software evolution: rapid requirements changing and component reuse. The RH model with a multi-dimensional architecture is constructed to be a basis of dependency-computing and management rules inferring via a lightweight inference engine. Particularly, input component search with attribute and content retrieval to an atomic SPIDER and software evolution job assignment can be automatically realized in our component reusable architecture.

V. DEPENDENCY-COMPUTING MODEL

The dependency-computing model integrates the fundamental software evolution model, like the hypergraph model, the evolutionary hypergraph model, and the RH model, with the dependency rules that are driven by a lightweight inference engine [HARN99b] [HARN99c]. The lightweight inference engine is suitable to compute small scale and domain-specific inference rules [HARN99a].

Domain-specific inference often requires large numbers of similar rules. These systems are generally not very modular, and consequently they are highly difficult to extend and refine. In many cases, inference rules are implicitly encoded in complex algorithms. Even if the rules are explicit, they may not be systematically organized. In order to achieve adequate efficiency, we keep inference chains short. This leads to large numbers of very specific rules and algorithms that are coupled to the structure of the problem space [BERZ98].

There are two kinds of *dependency rules* in the dependency-computing model: *dependency generation rules* and *dependency action rules*. According to data existence, the lightweight inference engine computes the dependencies among the software evolution objects via the dependency generation rules (stated by $\Leftrightarrow$). The specific combination of dependencies can automatically support software evolution via the dependency action rules (stated by $\Rightarrow$). In the dependency rules, there are four sources from which input data can be obtained: functions, the result of inference, database and stakeholders.

A. SOFTWARE EVOLUTION

1. Preliminary

a. Object types

The set of software evolution steps and components is structured as a class architecture.

According to the RH model of the software evolution process, based on the schematic model of the analysis process modified from the IBIS model in [CONK88] [IBRA96], there are eight types of software evolution steps: *software prototype demo, issue analysis, requirement analysis, specification analysis, module implementation, program integration, software product demo,* and *software product implementation.* The set S of software evolution steps contains at least these subtypes.

In the RH model there are ten types of software evolution components: *criticisms, issues, requirements, specifications, modules, software prototype programs, software product programs, optimizations, test scenarios,* and *virtual teams (or stakeholders).* Therefore, the set C of software evolution components contains at least these subtypes.

Finally, N is the set of natural numbers, which is used to represent variant and version numbers.

b. Object attributes and functions

The following version attributes of an object are used to defined dependency action rules. The attributes can bind data via dependency action rules and record data to database. The version attributes of an object are described as follows:

- *o.version*: an attribute of an object *o* that identifies the version of an object *o* and consists of the three parts: *o.version.object-id*, *o.version.variant-number*, and *o.version.version-number*;
- *o.version.object-id*: a subattribute of *o.version* that represents the object identifier of an object *o*;
- *o.version.variant-number*: a subattribute of *o.version* that represents a unique

identifier of a variant of an object *o*; and

- *o.version.version-number*: a subattribute of *o.version* that represents a unique identifier of a version of a specific variant of an object *o*.

The following functions are used to obtain the output from the relational hypergraph net and to define dependency rules:

- *primary-input(s)*: the set of primary input components to the step *s*;
- *secondary-input(s)*: the set of secondary input components to the step *s*;
- *input(s)*: the set of input components to the step *s*;
- *output(s)*: the set of output component from the step *s*;
- *type(s)*: a type indicator for step *s* that has two possible values: *"s"* (step) or *"d"* (decomposition);
- *highest-variant-number(c.version.object-id)*: the highest variant number of the object denoted by *c.version.object-id* in the current state; and
- *max(c1.version.version-number, c2.version.version-number)*: the maximum version number in *c1.version.version-number* and *c2.version.version-number*.

The following functions are used to evaluate logical value from the relational hypergraph net:

- *primitive-component(c)*: true if component *c* is a primitive-component;
- *atomic(c)*: true if component *c* is an atomic component;
- *current(s)*: true if step *s* is a current step; and
- *current(c)*: true if component *c* is a current component.

c. *Dependencies*

The dependencies among software evolution objects are classified into four types: component-to-step, step-to-component, component-to-component, and step-to-step dependencies.

In component-to-step dependencies, there are two types of dependencies: *primary_input* and *secondary_input*, among a step and its input nodes. In step-to-component dependency there is only one type of dependency: *output*, among a step and its output node.

In component-to-component dependencies, there is one dependency: *used_by*, between two components.

- *used_by*: between the components of a given configuration.

In component-to-component and step-to-step dependencies, there are five types of dependencies: *part_of, same_object, same_variation, null,* and *unknown*, among the software evolution objects as follows:

- *part_of*: between a substep of a composite step and the composite step or between a subcomponent of a composite component and the composite component,
- *same_object*: between the two objects of the same object identifier,
- *same_variation*: between the two objects of the same variation number,
- *null*: between the two objects that have no relationship after inference, and
- *unknown*: between the two objects that have no relationship before inference.

There are four subclasses of *used_by* dependencies: *used_by.test, used_by.handle*, *used_by.produce, used_by.merge*, and *use_by.merge*. These apply among different software evolution components as follows:

- *used_by.test*: between program and test scenario components,
- *used_by.handle*: between stakeholder (or virtual team) and the following components: criticism, issue, requirement, specification, model, program, optimization, and test scenario,
- *used_by.produce*: between two components one of which is an input component to a step and the other of which is an output component to this step,
- *used_by.split*: between two components that have the same identifier but the different variation number, and
- *used_by.merge*: between two components that have the same identifier and the different variation number.

There are two subclasses of *used_by.produce* dependencies: *used_by.produce.directly* and *used_by.produce.indirectly*, between two components one of which is an input component to a step and the other of which is an output component to this step as follows:

- *used_by.produce.directly*: between two components one of which is an input component to a step and the other of which is an output component to the same step, and
- *used_by.produce.indirectly*: between two components one of which is an input component to one step and the other of which is an output component to another

step.

There are two subclasses of *used_by.merge* dependencies: *used_by.merge.new_variation* and *used_by.merge.old_variation,* between two components that have the same identifier and the different variation number as follows:

- *used_by.merge.new_variation*: between two components that have the same identifier and the different variation number, but whose output component to a merge step has the new variation number from them after merging, and
- *used_by.merge.old_variation*: between two components that have the same identifier and the different variation number, but whose output component to a merge step has the old variation number from them after merging.

There are two subclasses of *part_of* dependencies: *part_of.step* and *part_of.component*, between a subobject of a composite object and this composite object as follows:

- *part_of.step*: between a substep of a composite step and this composite step, and
- *part_of.component*: between a subcomponent of a composite component and this composite component.

2. Dependency rules

A number of dependency rules have been developed for automated decision support and software evolution processes, i.e. version control, task decomposition, component retrieval, and so forth.

a. Object identifiers

A version of an object is one of the attributes of this object that can be represented as a string type containing the concatenation of an object identifier, a variant number, and a version number.

An object identifier can be a component identifier or a step identifier. We represent component identifiers with *"C", "I", "R", "S", "M", "P", "O"*, and *"Pd"*, and the step identifiers with *"s-C", "s-I", "s-R", "s-M", "s-S", "s-P", "s-O"*, and *"s-Pd"*. The generation of the object identifiers are inferred via the following rules.

$$ALL(s : S, c : C :: c\ output\ s \Leftrightarrow c \in output(s)) \qquad (1)$$

$ALL(s : S.software\text{-}prototype\text{-}demo, c : C.criticism :: c\ output\ s \Rightarrow c.version.object\text{-}id = "C" \ \&\ s.version.object\text{-}id = "s\text{-}C")$ (2)

$ALL(s : S.issue\text{-}analysis, c : C.issue :: c\ output\ s \Rightarrow c.version.object\text{-}id = "I" \ \&\ s.version.object\text{-}id = "s\text{-}I")$ (3)

$ALL(s : S.requirement\text{-}analysis, c : C.requirement :: c\ output\ s \Rightarrow c.version.object\text{-}id = "R" \ \&\ s.version.object\text{-}id = "s\text{-}R")$ (4)

$ALL(s : S.specification\text{-}design, c : C.specification :: c\ output\ s \Rightarrow c.version.object\text{-}id = "S" \ \&\ s.version.object\text{-}id = "s\text{-}S")$ (5)

$ALL(s : S.module\text{-}implementation, c : C.module :: c\ output\ s \Rightarrow c.version.object\text{-}id = "M" \ \&\ s.version.object\text{-}id = "s\text{-}M")$ (6)

$ALL(s : S.program\text{-}integration, c : C.software\text{-}prototype\text{-}program :: c\ output\ s \Rightarrow c.version.object\text{-}id = "P" \ \&\ s.version.object\text{-}id = "s\text{-}P")$ (7)

$ALL(s : S.software\text{-}product\text{-}demo, c : C.optimization :: c\ output\ s \Rightarrow c.version.object\text{-}id = "O" \ \&\ s.version.object\text{-}id = "s\text{-}O")$ (8)

$ALL(s : S.software\text{-}product\text{-}implementation, c : C.software\text{-}product\text{-}program :: c\ output\ s \Rightarrow c.version.object\text{-}id = "Pd" \ \&\ s.version.object\text{-}id = "s\text{-}Pd")$ (9)

b. Object variants and versions

Variants represent alternative formulations of a software object with different objectives, for instance running on different operating systems or serving different user communities. Successive versions of the same variant represent refinements or improvements to the component [BADR93] [BADR94].

The primitive component that is a source component cannot be produced by any step. The primitive component could have more than one variant. The variants are assigned successive numbers. The variants of a primitive component are less than those of a nonprimitive component. Versions along each variant are assigned successive numbers, starting with 1 at the root version of the initial variant.

$ALL(c : C :: primitive\text{-}component(c) \Leftrightarrow \neg EXISTS(s : S :: c \in output(s)))$ (10)

$ALL(c1\ c2 : C :: primitive\text{-}component(c1)\ \&\ \neg\ primitive\text{-}component(c2) \Rightarrow$
$(c1.version.variant\text{-}number < c2.version.variant\text{-}number)\ \&\ c1.version.version\text{-}number = 1)$ (11)

c. Primary and secondary inputs

The dependencies *same_object* and *same_variant* are defined by the attributes *version.object-id* and *version.variant-number* as follows.

$ALL(c1\ c2 : C :: c1\ same\_objcet\ c2 \Leftrightarrow c1.version.object\text{-}id = c2.version.object\text{-}id)$ (12)

$ALL(c1\ c2 : C :: c1\ same\_variant\ c2 \Leftrightarrow c1.version.variant\text{-}number = c2.version.variant\text{-}number)$ (13)

The primary and secondary input concept can be formalized by the attributes *version.object-id* and *version.version-number*. An input to a step is primary if and only if it is the previous version of the same object as the output of the step. An input to a step is secondary if and only if it is not the same object as the output of the step. The dependencies *primary_input* and *secondary_input* are defined by the following dependency rules.

$ALL(s : S, c : C :: c\ input\ s \Leftrightarrow c \in input(s))$ (14)

$ALL(s : S, c1\ c2 : C :: c1\ primary\_input\ s \Leftrightarrow c1\ input\ s\ \&\ c2\ output\ s\ \&\ c1\ same\_object\ c2)$ (15)

$ALL(s : S, c1\ c2 : C :: c1\ secondary\_input\ s \Leftrightarrow c1\ input\ s\ \&\ c2\ output\ s\ \&\ \neg\ (c1\ same\_object\ c2))$ (16)

d. Evolution history splitting and merging

Creating a new component in a variant different from the original variant is called software evolution history splitting. The dependency between the input component and the output component in the evolution history splitting is represented by *used_by.split*.

$ALL(c1\ c2 : C, s : S :: c1\ used\_by.split\ c2 \Leftrightarrow c1\ primary\_input\ s\ \&\ c2\ output\ s\ \&\ \neg\ (c1\ same\_variant\ c2))$ (17)

Creating a new component based on two primary input components is called software evolution history merging. The dependency between the two primary input components in the software evolution history merging is represented by *used_by.merge* which is divided into two subclasses: *used_by.merge.new_variant* and *used_by.merge.old_variant*. In the situation of software evolution history merging, if the variant of the output component is not the same as that of the two primary input components, the dependency between the two primary input components is denoted by *used_by.merge.new_variant*. Additionally, if the variant of the output component is the same as that of one of the two primary input components, the dependency between the two primary input components is denoted by *used_by.merge.old_variant*.

ALL(c1 c2 : C :: c1 used_by.merge.new_variant c2 ⇔ EXISTS(c3 : C :: c1 used_by.split c3 & c2 used_by.split c3)) (18)

ALL(c1 c2 : C :: c1 used_by.merge.old_variant c2 ⇔ EXISTS(c3 : C, s : S :: c3 output s & ((c1 used_by.split c3 & c2 primary_input s & c2 same_variant c3) or (c2 used_by.split c3 & c1 primary_input s & c1 same_variant c3)))) (19)

ALL(c1 c2 : C :: c1 used_by.merge c2 ⇔ (c1 used_by.merge.new_variant c2) or (c1 used_by.merge.old_variant c2)) (20)

e. *Variant and version numbering*

The successive variant and version numbers rely on the above dependencies among the input components and output component to a specified step. They are defined as follows.

ALL(c1 c2 : C, s : S :: c1 primary_input s & c2 output s & c1 same_variant c2 ⇒ c2.version.version-number = c1.version.version-number + 1) (21)

ALL(c1 c2 : C, s : S :: c1 primary_input s & c2 output s & c1 used_by.split c2 ⇒ c2.version.version-number = c1.version.version-number + 1 & c2.version.variant-number = highest-variant-number(c1.version.object-id) + 1) (22)

ALL(c1 c2 c3 : C, s : S :: c1 primary_input s & c2 primary_input s & c3 output s & c1 used_by.merge.new-variant c2 ⇒ c3.version.version-number = max(c1.version.version-number, c2.version.version-number) + 1 & c3.version.variant-number = highest-variant-number(c1.version.object-id) + 1) (23)

ALL(c1 c2 c3 : C, s : S :: c2 primary_input s & c3 output s & c1 used_by.merge.old_variant c2 & c1 used_by.split c3 ⇒ c3.version.version-number = max(c1.version.version-number, c2.version.version-number) + 1) (24)

ALL(c1 c2 c3 : C, s : S :: c1 primary_input s & c3 output s & c1 used_by.merge.old_variant c2 & c2 used_by.split c3 ⇒ c3.version.version-number = max(c1.version.version-number, c2.version.version-number) + 1) (25)

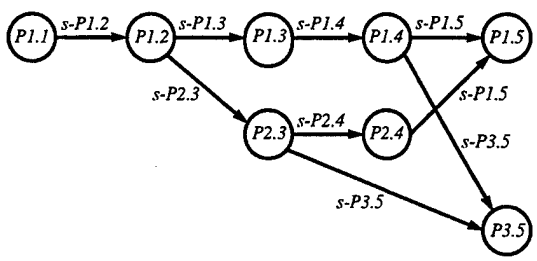


Figure 21: The variant and version numbering in the case of software evolution history splitting and merging

Figure 21 shows the variant and version numbering in the case of software evolution history splitting and merging. The node *P1.1* is the primitive component of the software program. The software evolution history is represented by a path in the hypergraph model. The variant number of each object in the path, from *P1.1* to *P1.5* via *s-P1.2, P1.2, s-P1.3, P1.3, s-P1.4, P1.4,* and *s-P1.5*, is specified by *1*. After the node *P1.2*, the step *s-P1.2* yields a software evolution history split and a new variant whose number of each object in the path, from *s-P2.3* to *P2.4* via *P2.3* and *s-P2.4*, is specified by *2*. After the nodes *P1.4* and *P2.3*, the step *s-P3.5* yields a software evolution history merge and a new variant whose number of each object in the path, from *s-P3.5* to *P3.5*, is specified by *3*. After nodes *P1.4* and *P2.4*, the step *s-P1.5* yields a software evolution merging but the

variant number is specified by *1*. The version number of each object in case of software evolution history splitting or merging is specified by the above dependency action rules.

f. Object decomposition

In the hypergraph model, software evolution objects can be decomposed into lower level objects. There are a *part_of.component* dependency between a composite component and its subcomponents, and a *part_of.step* dependency between a composite step and its substeps:

$$EXISTS(c1\ c2 : C,\ s : S :: c2\ part\_of.component\ c1 \Leftrightarrow \neg\ atomic(c1)\ \&\ c2\ input\ s\ \&\ c1\ output\ s\ \&\ type(s) = "d"\ ) \quad (26)$$

$$EXISTS(s1\ s2 : S,\ c1\ c2\ c3\ c4 : C :: s2\ part\_of.step\ s1 \Leftrightarrow \neg\ atomic(s1)\ \&\ c3\ part\_of.component\ c1\ \&\ c4\ part\_of.component\ c2\ \&\ c1\ input\ s1\ \&\ c2\ output\ s1\ \&\ c3\ input\ s2\ \&\ c4\ output\ s2\ \&\ type(s1) = "s"\ \&\ type(s2) = "s") \quad (27)$$

We concatenate the prefix symbol "*d-*" and the object identifier of a composite component to denote the decomposition step between the composite component and its subcomponents in order to distinguish the notation of activity step with "*d-*". The subcomponents and substeps are denoted with a string concatenating an object identifier and a natural number. The examples are shown in Figure 13 and Figure 14. The version control action rules of dependencies *part_of.component* and *part_of.step* are defined as follows:

$$EXISTS(c1\ c2 : C,\ s : S,\ n : N :: c2\ part\_of.component\ c1 \Rightarrow s.version.object\text{-}id \leftarrow string("d\text{-}",\ c1.version.object\text{-}id)\ \&\ c2.version.object\text{-}id = string(c1.version.object\text{-}id,\ string(n)) \quad (28)$$

$$EXISTS(s1\ s2 : S,\ n : N :: s2\ part\_of.step\ s1 \Rightarrow s2.version.object\text{-}id = string(s1.version.object\text{-}id,\ string(n))) \quad (29)$$

g. *Test scenarios and the virtual teams*

A test scenario is created to test a software prototype program or a software product program. The dependency between a program component and its test scenario component is *used_by.test* that is defined as follows:

ALL(p : C.software-prototype-program ∪ C.software-product-program, t : C.test-scenario, c : C.criticism, s : S :: t used_by.test p ⇔ t secondary_input s & p secondary_input s & c output s) (30)

A virtual team is formed to handle the input components and produce the output component to a specified step, but is not assigned to specific system developers (system analysts or system designers) yet. The dependency between a virtual team component and the other input components are *used_by.handle* that is defined as follows.

ALL(c : C - C.virtual-team, h : C.virtual-team, s : S :: h used_by.handle c ⇔ c input s & h input s) (31)

The object identifiers of a test scenario component and a virtual team component is related to the output component that they produce and to a specified step. We concatenate the prefix symbols "*T-*" and the object identifier of an output component to a specified step to denote the object identifier of a test scenario component, and the prefix symbols "*VT-*" and the object identifier of an output component to a specified step to denoted the object identifier of a virtual team component. The version action control rules of a test scenario component and a virtual team component are defined as follows:

ALL(p : C.software-prototype-program ∪ C.software-product-program, t : C.test-scenario, c : C.criticism :: t used_by.test p & t used_by.directly_produce c ⇒ t.version.object-id = string("T-", c.version.object-id)) (32)

ALL(c1 c2 : C, h : C.virtual-team :: h used_by.handle c1 & h used_by.directly_produce c2 ⇒ h.version.object-id = string("VT-", c2.version.object-id)) (33)

h. Component generation

A software evolution history can be regarded as a path in the hypergraph model. The final version of a component is evolved by a series of evolution processes from the previous versions of the same component with the primary input driven mechanism or from the different components with the secondary input driven mechanism. The dependency *used_by.directly_produce* is defined between an input component and an output component to a specified step. The dependency *used_by.indirectly_produce* is defined between two components but a component between them exists in their software evolution path:

ALL(c1 c2 : C, s : S :: c1 used_by.directly_produce c2 ⇔ c1 input s & c2 output s)(34)

ALL(c1 c3 : C :: c1 used_by.indirectly_produce c3 ⇔ c1 used_by.directly_produce c3 or EXISTS(c2 : C :: c1 used_by.indirectly_produce c2 & c2 used_by.directly_produce c3)) *(35)*

Figure 22 shows the new component generation in the software prototype demo step. The components *P1.2, T-C2.3*, and *VT-C2.3* can be used to directly produce the component *C2.3*. The component *C1.2* can be used to directly produce component *C2.3* via step *s-C2.3* or indirectly produce component *C2.3* via a software evolution path form component *C1.2* to component *C2.3* via *s-I1.2, I1.2, s-R1.2, R1.2, s-S1.2, S1.2, s-M1.2, M1.2, s-P1.2, P1.2,* and *s-C2.3*.

i. Component retrieval: SPIDER formation

SPIDER formation in preparation for executing a software evolution step is a component retrieval process via dependency rules and a lightweight inference. SPIDER formation can form either a top-level or a refined SPIDER. In the software evolution process, two successive versions of software are indirectly manipulated via a path of secondary-input-driven steps. Following the secondary-input-driven mechanism, we can use the current component and its related components as input components of a current step to generate a new version component. The question is how to obtain automatically the

related input components to form a SPIDER based on only known current components. The dependencies between this current component and its related components can be inferred to find the input components of a current step by the dependency rules and a lightweight inference engine.

$$ALL(s : S :: current(s) \Leftrightarrow \neg EXISTS(c : C :: c\ output\ s)) \quad (36)$$

$$ALL(c1 : C :: current(c1) \Leftrightarrow \neg EXISTS(c2 : C :: c1\ used\_by.directly\_produce\ c2)) \quad (37)$$

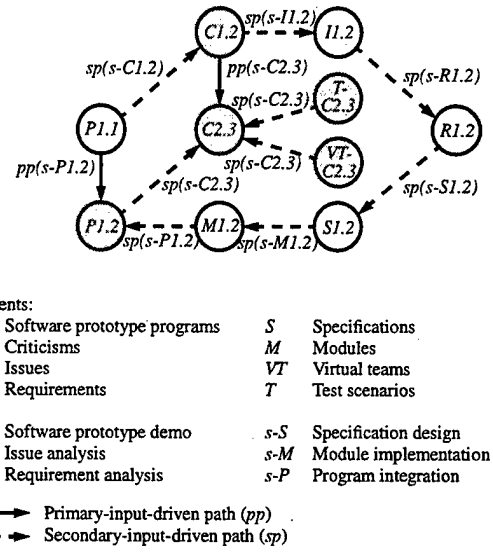


Figure 22: The new component generation in the software prototype demo step

The current component is regarded as a filter to slice the relational hypergraph into a software evolution path. In Figure 22, if step *s-C2.3* is a current step and component *P1.2* is a current component, before the step *s-C2.3* is executed, the components *T-C2.3, VT-C2.3, C1.2,* and *C2.3* are unknown, except for the current component *P1.2*. We can find all of the primary and secondary input components based on the current component *P1.2* via dependency inferring and produce the component *C2.3* via executing step *s-C2.3* with its input components.

The input component to a current step is a set that combines a primary input component set and a secondary input component set. The members of the primary and secondary input component sets depend on different software evolution steps. The primary

input component(s) in the software evolution path can be retrieved by the dependencies *used_by.directly_produce* and *used_by.indirectly_produce*. The secondary input component can be found by the dependencies *used_by.directly_produce* and *used_by.handle*.

Take the software prototyping demo step for example. The type of the primary input component and output component is criticism and the types of the secondary input components are software prototype program, test scenario, and virtual team.

primary-input(s : S.software-prototyping-demo) = {c1 : C.criticism | EXISTS(p1 p2 : C.software-prototype-program, c2 : C.criticism :: p2 used_by.directly_produce c2 & p1 used_by.directly_produce p2 & p1 used_by.directly_produce c1 & c1 used_by.indirectly_produce p2)} (38)

secondary-input(s : S.software-prototyping-demo) = {t : C.test-scenario, p : C.software-prototype-program, h : C.virtual-team | EXISTS(c : C.criticism :: p used_by.directly_produce c & t used_by.test p & h used_by.handle p)} (39)

input(s : S.software-prototyping-demo) = primary-input(s) ∪ secondary-input(s) (40)

The test scenario is created for the software prototype program and the dependency between them is *used_by.test*. The test scenario can be used to directly produce a new criticism if and only if it can be used to test the current component of the software prototype program.

EXISTS(t : C.test-scenario, c : C.criticism, p : C.software-prototype-program :: t used_by.directly_produce c ⇔ p used_by.directly_produce c & t used_by.test p) (41)

The virtual team can be used to directly produce a new criticism if and only if it can be used to handle the current component of the software prototype program.

EXISTS(h : C.virtual-team, c : C.criticismp, p : C.software-prototype-program :: h used_by.directly_produce c ⇔ p used_by.directly_produce c & h used_by.handle p) (42)

j. Unknown dependencies

The unknown dependency between two components can be used to infer the dependency by the lightweight inference engine. The *part_of* dependency combines *part_of.component* and *part_of.step* dependencies. The *used_by* is a dependency that combines the following subclass dependencies: *used_by.split, used_by.merge, used_by.test, used_by.handle, used_by.directly_produce,* and *used_by.indirectly_produce*. The dependency *unknown* states that the dependency between two arbitrary components at the current time is unknown and needs to be inferred from the dependency rules. We can not guarantee that the inferring result of two unknown dependency components can be found, but normally it can be determined by the following dependencies: *same_object, same_variant, part_of, used_by* or *null* (that means there is no dependency between two components). They can be defined as follows:

$$EXIST(o1\ o2 : C \cup S :: o1\ part\_of\ o2 \Leftrightarrow o1\ part\_of.component\ o2\ or\ o1\ part\_of.step\ o2) \quad (43)$$

$$EXISTS(c1\ c2 : C :: c1\ used\_by\ c2 \Leftrightarrow c1\ used\_by.split\ c2\ or\ c1\ used\_by.merge\ c2\ or\ c1\ used\_by.test\ c2\ or\ c1\ used\_by.handle\ c2\ or\ c1\ used\_by.directly\_produce\ c2\ or\ c1\ used\_by.indirectly\_produce\ c2) \quad (44)$$

$$EXISTS(c1\ c2 : C :: c1\ null\ c2 \Leftrightarrow \neg(c1\ same\_object\ c2\ and\ c1\ same\_variant\ c2\ and\ c1\ part\_of\ c2\ and\ c1\ used\_by\ c2)) \quad (45)$$

$$EXISTS(c1\ c2 : C :: c1\ unknown\ c2 \Leftrightarrow c1\ same\_object\ c2\ or\ c1\ same\_variant\ c2\ or\ c1\ part\_of\ c2\ or\ c1\ used\_by\ c2\ or\ c1\ null\ c2) \quad (46)$$

The complicated relationship among the software evolution objects is the important key to inferring the unknown dependencies between two objects and to execute actions for the needs of the software evolution management.

B. JOB SCHEDULING

1. Job scheduling model

The job scheduling model is based on the heuristic mechanism that provides the features to rearrange/cancel requests in the step priority queue, and to change step priorities dynamically. This model can do synchronization and rendezvous scheduling via scheduling dependency rules.

a. Scheduling constraints

A step in the CASES can be regarded as a job or a task [BADR93] [BADR94] [EVAN97]. The task set in the CASES scheduling problem is a variable set of evolution steps $S = \{s_1, s_2, ..., s_n\}$, where n varies with time. This set of tasks needs to be scheduled to a set of m stakeholders $D = \{d_1, d_2, ... , d_m\}$. The stakeholders are of E different skills with L different skill levels. Tasks as used in the CASES are independent, nonperiodic and non-preemptive. They can be charactered by the follows:

- Skills and skill levels: $T_{skill} = \{(e_1, l), (e_2, l), ... , (e_k, l)\}$ where k is different skills and $l \in L = \{0, 1, 2, 3\}$, requested by a task s or a designer d;
- Security level: $T_{security} = \{0, 1, 2, 3, 4, 5\}$;
- Predecessors: $T_{predecessor}$;
- Priority: $T_{priority} = \{1, 2, 3, 4, 5\}$;
- Deadline: $T_{deadline}$;
- Estimated duration: $T_{estimated\_duration}$;
- Earliest start time: $T_{earliest\_start\_time}$;
- Finish time: T_{finish_time};

The skill and its skill level are attributes of a step as well as attributes of a stakeholder. The project organizers or evaluators determines the skill and its skill level of steps and store them to the step database. The project evaluators or the project managers of the stakeholders assess the capability of a stakeholder with the skill and its skill level.

CASES assign jobs to designers by the matching of job skill requirements and capacities of stakeholders.

The skill can be any related techniques of the software evolution, such as the understanding of UNIX system, Ada, TAE + [CENT93], and so on. The skill levels are none, low, middle, and high that are denoted by *0, 1, 2* , and *3* respectively. The skill set, the skill level set, the job-skill set, and the stakeholder set are defined as follows:

- Skill $K = \{k1, \ldots, kn\}$,
- Skill level $L = \{0, 1, 2, 3\}$,
- $Job\text{-}skill(s : S) = \{(k, l) \mid k \in K, l \in L\}$, and
- $Stakeholder\text{-}skill(h : H) = \{(k, l) \mid k \in K, l \in L\}$.

Skills and skill levels as well as security level are used to search a feasible schedule of stakeholders. Predecessors, priority, deadline, estimated duration and earliest start time are used to sort a set of tasks. Finish time is recorded when a task has been carried out.

The skill level *0* is the lowest and the skill level *3* is the highest in the skill level set *L*. The security level *0* is no security consideration and the security level *5* is the highest security consideration in the security level set $T_{security}$. The priority *1* is the lowest priority and the priority *5* is the highest priority in the priority set $T_{priority}$.

Each task is associated to a predecedence constraint given in the form of a directed acyclic graph $G = \{S, E\}$ such that $(s_i, s_j) \in E$ implies that s_j cannot start until s_i has completed.

The priority, $T_{priority}$, is a small positive integer that is assigned to each task to reflect the importance of its deadline. The priorities of different tasks should be compatible with the precedence constraints between the steps; i.e. no lower priority step can precede a higher priority step:

$$\text{if } (s_2, s_1) \in E \Rightarrow T_{priority}(s_2) \quad T_{priority}(s_1) \text{ and}$$

$$\text{if } (s_2, s_1) \in E \;\&\; T_{priority}(s_1) \quad T_{priority}(s_3) \Rightarrow T_{priority}(s_2) >= T_{priority}(s_3).$$

Project organizers give the deadline $T_{deadline}$ and estimated duration $T_{estimated\_duration}$. CASES calculates the start time $T_{earliest\_start\_time}$ after scheduling. Stakeholders give the finish time T_{finish_time} after they finish an assigned task.

b. Scheduling heuristics

Scheduling a set of tasks to find a fully feasible schedule is a variant of the sorting and searching problem. We have to sort a set of tasks and then search for a feasible schedule according to the scheduling constraints and the scheduling heuristics.

The scheduling constraint function $C(T)$ and the heuristic function $H(T)$ order the set of tasks ready to be scheduled.

The related scheduling constraints are as follows [BADR93] [BADR94] [EVAN97]:

- Predecessors: $C(T) = T_{predecessor}$;
- Priority: $C(T) = T_{priority}$;

The candidate heuristics are as follows:

- Minimum deadline first (Min_D): $H(T) = T_{deadline}$;
- Minimum estimated duration first (Min_E): $H(T) = T_{estimated\_duration}$;
- Minimum earliest start time first (Min_S): $H(T) = T_{earliest\_start\_time}$;
- Minimum laxity first (Min_L): $H(T) = T_D - (T_{earliest\_start\_time} + T_{estimated\_duration})$;
- Min_D + Min_E: $H(T) = W \times T_{dead\_line} + (1 - W) \times T_{estimated\_duration}$;
- Min_D + Min_S: $H(T) = W \times T_{dead\_line} + (1 - W) \times T_{earliest\_start\_time}$;

The weight $W\ (0 \leq W \leq 1)$, used to combine the two simple heuristics Min_D and Min_E or Min_D and Min_S, can be tuned according to how critical the deadlines of the available steps are.

c. Scheduling algorithms

The job scheduling algorithms is integrated by *JobSchedule* and *JobAssign* algorithms as follows:

JobSchedule

Step 1. Create lists J_1 and J_2 for jobs.

Step 2. Put all the jobs under a specified project into list J_1.

Step 3. If the jobs of list J_1, whose status is *approved,* have no predecessors, put the jobs into list J_2.

Step 4. If the jobs of list J_1, whose status is *approved,* have predecessors and the status of all predecessors is *completed*, put the jobs into list J_2.

Step 5. If the status of a job in list J_1 is *scheduled*, put the job into list J_2.

Step 6. If the status of a job in list J_2 is *approved*, change the status of the job in list J_2 into *scheduled.*

Step 7. Sort the jobs of list J_2 based on the heuristic chosen by users.

Step 8. Pop and assign the first job j of list J_2 to stakeholders by *JobAssign.*

Step 9. Go to Step 1 until the jobs in list J_1 have no *approved* status and *scheduled* status.

JobAssign

Step 1. Create lists S_1 and S_2 for stakeholders and a two-element list $T(m)$ for a stakeholder m.

Step 2. Get the security level l of job j.

Step 3. Put all the stakeholders into list S_1.

Step 4. Put the stakeholders of list S_1, whose security level is no less than the security level l of job j, into list S_2.

Step 5. If list $T(m)$ of each stakeholder in list S_2 is full then return.

Step 6. Count the skill matching number n for each stakeholder of list S_2 based on the dependency rule that the required skill level of job j is no less than the associated skill level of stakeholders.

Step 7. Sort the stakeholders of list S_2 by the skill matching number n.

Step 8. Pop the first stakeholder m of list S_2 out.

Step 9. If list $T(m)$ is not full then assign job j to the stakeholder m as the major job by the following substeps: (1) append job j into list $T(m)$. (2) change the status of job j into *assigned.* (3) return.

Step 10. If list $T(m)$ is full then assign job j to the stakeholder m as the minor job and go to Step 8.

2. Dependency rules

a. Step attributes and dependency functions

The following attributes of a step that can be entered by manager or calculated by CASES are used to defined dependency action rules. The attributes can bind data via dependency action rules and record data to step database. The attributes of a step are described as follows:

- *s.predecessor*: the predecessor of a step *s*;
- *s.priority*: the priority of a step *s*;
- *s.deadline*: the deadline of a step *s*;
- *s.estimated-duration-time*: the estimated duration time of a step *s*;
- *s.earliest-start-time*: the start time of a step *s*; and
- *s.finish-time*: the finish time of a step *s*.

The following functions are used to obtain the step from the step content and to define dependency rules:

- *predecessor(s)*: get the predecessor of the step *s*;
- *deadline(s)*: get the deadline of the step *s*;
- *estimated_duration(s)*: get the estimated duration of the step *s*;
- *earliest_start_time(s)*: get the earliest start time of the step *s*;
- *laxity(s)*: get the laxity of the step *s*;
- *deadline_and_estimated_duration*: get the sum of deadline and estimated duration of the step *s* with a weight *W*; and
- *deadline_and_earliest_start_time*: get the sum of deadline and earliest start time of the step *s* with a weight *W*.

b. Scheduling dependency rules

The dependencies in the job scheduling model is step-to-step dependencies. There are three types of dependencies between two steps as follows:

- *precedes*: between two steps;
- *concurs_with*: between two steps;
- *precedes_by*: between two steps.

There are two subclasses of *precedes* dependencies between two steps as follows:

- *precedes.immediately*: between two steps; and
- *precedes.remotely*: between two steps

There are six subclasses of *precedes_by* dependencies between two steps as follows:

- *precedes_by.deadline*: between two steps;
- *precedes_by.estimated_duration*: between two steps;
- *precedes_by.earliest_start_time*: between two steps;
- *precedes_by.laxity*: between two steps;
- *precedes_by.deadline_and_estimated_duration*: between two steps; and
- *precedes_by.deadline_and_earliest_start_time*: between two steps.

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes immediately *s2* if and only if *s1* is a predecessor of *s2*. The dependency *prededes.immediately* is defined as follows:

$$ALL(s1\ s2 : S :: s1\ precedes.immediately\ s2 \Leftrightarrow s1 = predecessor(s2)) \quad (47)$$

Let *s, s1* and *s2* denote three different steps. For all *s1* and *s2*, it is the case that *s1* is precedes remotely *s2* if and only if there is a s such that *s1* precedes immediately *s*, and either *s* precedes immediately or remotely *s2*. The dependency *precedes.remotely* is defined as follows:

$$ALL(s1\ s2 : S :: s1\ precedes.remotely\ s2 \Leftrightarrow EXIST(s : S :: s1\ precedes.immediately\ s\ \&\ (s\ precedes.immediately\ s2\ or\ s\ precedes.remotely\ s2))) \quad (48)$$

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* if and only if *s1* precedes immediately or remotely *s2*. The dependency *precedes* is defined as follows:

$$ALL(s1\ s2 : S :: s1\ precedes\ s2 \Leftrightarrow s1\ precedes.immediately\ s2\ or\ s1\ precedes.remotely\ s2) \quad (49)$$

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* concurs with *s2* if and only if *s1* does not precede *s2* and *s2* does not precede *s1*. The dependency *concurs_with* is defined as follows:

$$ALL(s1\ s2 : S :: s1\ concurs\_with\ s2 \Leftrightarrow \neg(s1\ precedes\ s2)\ and \neg(s2\ precedes\ s1))(50)$$

c. *Scheduling policy rules*

The scheduling decision of stakeholders will effect the scheduling. In concurrent steps of a scheduling dependency list, we can reschedule the steps by the different policies. The current elements in the scheduling dependency list will be adjusted after the scheduling policy rules are executed.

(1) Policy 1: Minimum deadline first between two steps

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the minimum deadline first policy if and only if the deadline of *s1* is no greater than that of *s2*. The dependency *precedes_by.deadline* is charactered as follows:

$$ALL(s1\ s2 : S :: s1\ precedes\_by.deadline\ s2 \Leftrightarrow \neg(deadline(s1) > deadline(s2)))(51)$$

(2) Policy 2: Minimum estimated duration first between two steps

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the minimum estimated duration first policy if and only if the estimated duration of s1 is no less than that of s2. The dependency *precedes_by.estimated_duration* is charactered as follows:

$$ALL(s1\ s2 : S :: s1\ precedes\_by.estimated\_duration\ s2 \Leftrightarrow \neg(estimated\text{-}duration(s1) < estimated\text{-}duration(s2)))\quad(52)$$

(3) Policy 3: Minimum earliest start time first

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the minimum earliest start time first policy if and only if the earliest start time of *s1* is no greater than that of *s2*. The dependency *precedes_by.earliest_start_time* is charactered as follows:

ALL(s1 s2 : S :: s1 precedes_by.earliest_start_time s2 ⇔ ¬ (earliest-start-time(s1) > earliest-start-time(s2))) (53)

(4) Policy 4: Minimum laxity first

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the minimum laxity first policy if and only if the laxity of *s1* is no greater than that of *s2*. The dependency *precedes_by.laxity* is charactered as follows:

ALL(s1 s2 : S :: s1 precedes_by.laxity s2 ⇔ ¬ (laxity(s1) > laxity(s2))) (54)

(5) Policy 5: Min_D + Min_E

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the Min_D + Min_E policy if and only if the sum of deadline and estimated duration of *s1* is no greater than that of *s2*. The dependency *precedes_by.deadline_and_estimated_duration* is charactered as follows:

ALL(s1 s2 : S :: s1 precedes_by.deadline_and_estimated_duration s2 ⇔ ¬ (deadline-and-estimated-duration(s1) > deadline-and-estimated-duration(s2))) (55)

(6) Policy 6: Min_D + Min_S

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the Min_D + Min_S policy if and only if the sum of deadline and earliest start time of *s1* is no greater than that of *s2*. The dependency *precedes_by.deadline_and_earliest_start_time* is charactered as follows:

ALL(s1 s2 : S :: s1 precedes_by.deadline_and_earliest_start_time s2 ⇔ ¬ (deadline-and-earliest-start-time(s1) > deadline-and-earliest-start-time(s2))) (56)

Let *s1* and *s2* denote two different steps. For all *s1* and *s2*, it is the case that *s1* precedes *s2* based on the heuristic policy if and only if *s1* precedes *s2* by deadline, estimated duration, earliest start time, laxity, deadline and estimated duration, or deadline and earliest start time. The dependency *precedes_by* is charactered as follows.

ALL(s1 s2 : S :: s1 precedes_by s2 ⇔ s1 precedes_by.deadline s2 or s1 precedes_by.estimated_duration s2 or s1 precedes_by.earliest_start_time s2 or s1

precedes_by.laxity s2 or s1 precedes_by.deadline_and_estimated_duration s2 or s1 precedes_by.deadline_and_earliest_start_time s2) *(57)*

Automation of software evolution with formal methods is one of the best ways to handle large and complex software system development. The contribution of this chapter is to propose a dependency-computing model that can automate parts of software evolution with dependency inference rules and resolve issues of rapid requirement changes and software evolution component reuse.

VI. DESIGN OF CASES

CASES has been designed by Java JDK1.1.7 under the *VisualCafé* environment [FLAN97] [SYMN98]. The Java code of CASES is attached in Appendix D [LE99]. Based on the RH model and dependency-computing model, we develop a three-tier object-oriented architecture for CASES, shown in Figure 23:

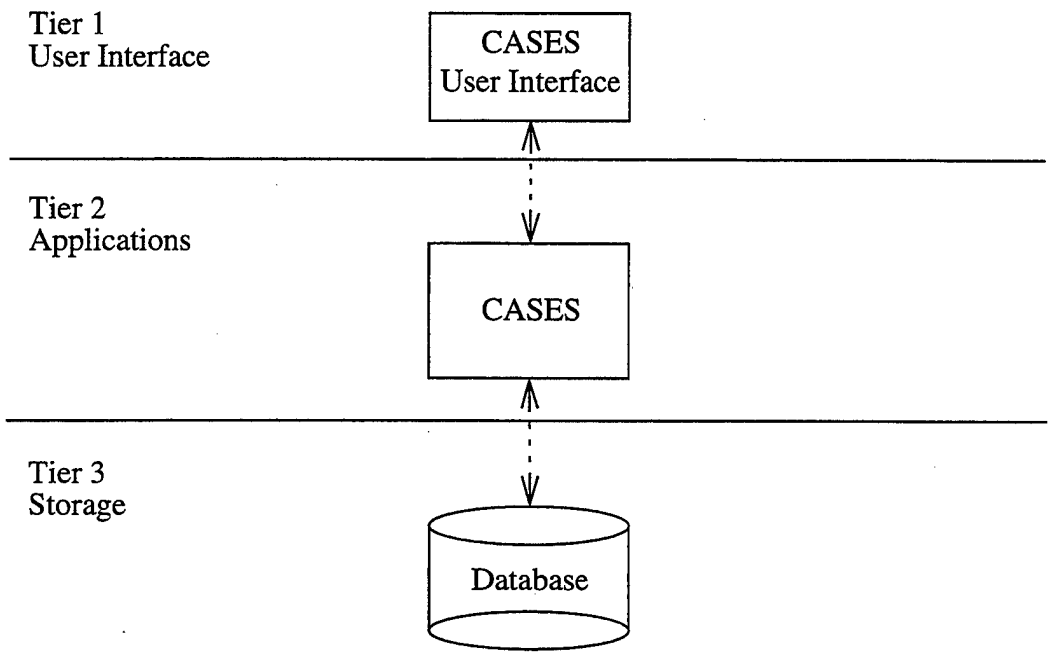


Figure 23: Three-tier object-oriented architecture of CASES

Figure 24 shows the system context diagram of CASES that includes two external objects: a stakeholder and a tool. CASES is used by stakeholders, who are project team leaders/managers, project organizers, project evaluators, system analysts, and system designers. CASES provides the interface to connect external tools, currently Text Editor, MS Word, MS Excel, Netscape, and CAPS.

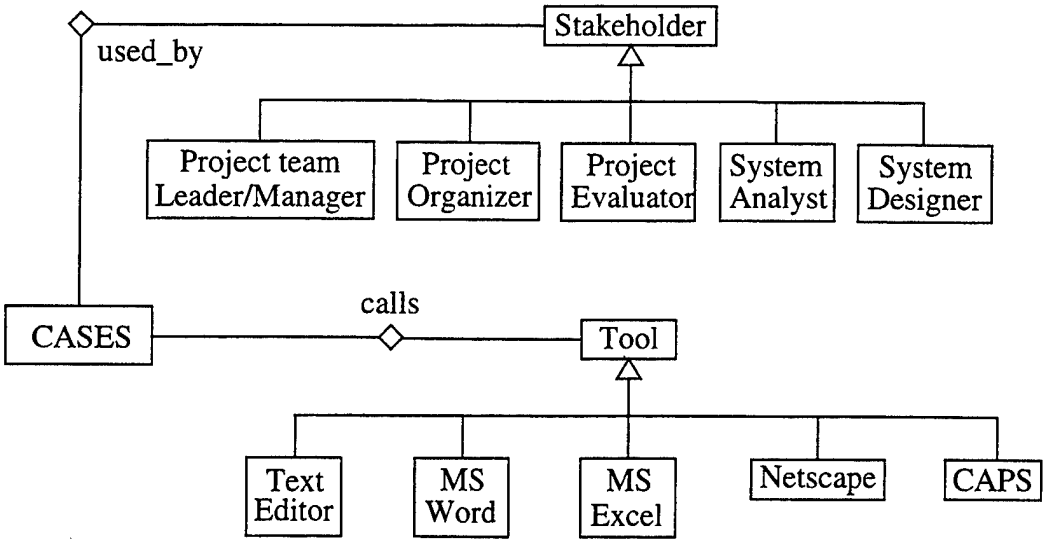


Figure 24: System context diagram of CASES

A. CLASS STRUCTURE

The CASES class structure is based on the hypergraph shown in Figure 25, and described as follows:

- The class *Hypergraph* can be decomposed into lower level structure.
- The class *Evolutionary Hypergraph* is part of the class Project.
- The classes *Evolutionary Hypergraph* and *Process Hypergraph* inherit the class *Hypergraph.*
- The classes *Edge* and *Node* are parts of the class *Hypergraph* and inherit the class *Decomposable Entity* in which there is the relationship *part_of.*
- There are two relationships: *output* and *input* between the classes *Edge* and *Node.*
- The relationships: *primary input* and *secondary input* inherit the relationship *input.*
- The class *Component* is part of the class *Evolutionary Hypergraph* and inherits the classes *Node* and *Versionable Entity.*
- The class *Component Type* is part of the class *Process Hypergraph* and the class *Component*, and inherits the class *Node.*
- The class *Step* is part of the class *Evolutionary Hypergraph* and inherits the class *Edge.*

- The class *Step Type* is part of the class *Process Hypergraph* and the class *Step*, and inherits the class *Edge*.
- The class *Component Type ID* is part of the class *Component Type* and inherits the class *Dynamic Enumeration.*
- The class *Step Type ID* is part of the class *Step Type* and inherits the class *Dynamic Enumeration.*
- The classes *Security ID* and *Skill ID* also inherit the class *Dynamic Enumeration.*
- The class *Dynamic Enumeration* is part of the class *Level Map* that is part of the class *Step* and the class *Virtual Team.*

B. FILE STRUCTURE

Many CASES files have been created in near isolation and incorporated onto the system as a whole. The hierarchical file structure of CASES includes five levels: a CASES directory, project names, step identifiers and related files, version numbers, and a recursive SPIDER construction, shown in Figure 26.

In the CASES directory level, CASES creates a directory: *<cases>* to construct CASES repositories.

In the project names level, CASES creates project subdirectories under directory *<cases>* to construct projects, such as subdirectories: *<c4i>* and *<c3i>*.

In the step identifiers and related files level, CASES creates object databases, working memory files, and configuration management files. In the object databases, CASES creates several subdirectories under project names to store related data of steps and components by step identifiers, such as subdirectories: *<s-C>*, *<s-I>*, ..., *<s-Pd>*. In the working memory files, CASES creates the files: *current.vsn* to store working data. In the configuration management files, CASES creates the files: *loop.cfg, step.cfg, component.cfg*, and *dependency.cfg* to store object configurations.

In the version numbers level, CASES creates several subdirectories under object databases to store objects by version numbers, such as subdirectories: *<1.1>*, *<1.2>*, ..., *<2.3>*. These subdirectories represent the SPIDER name that the step identifiers in the Step Identifiers and Related Files level combine with the version number of subdirectories.

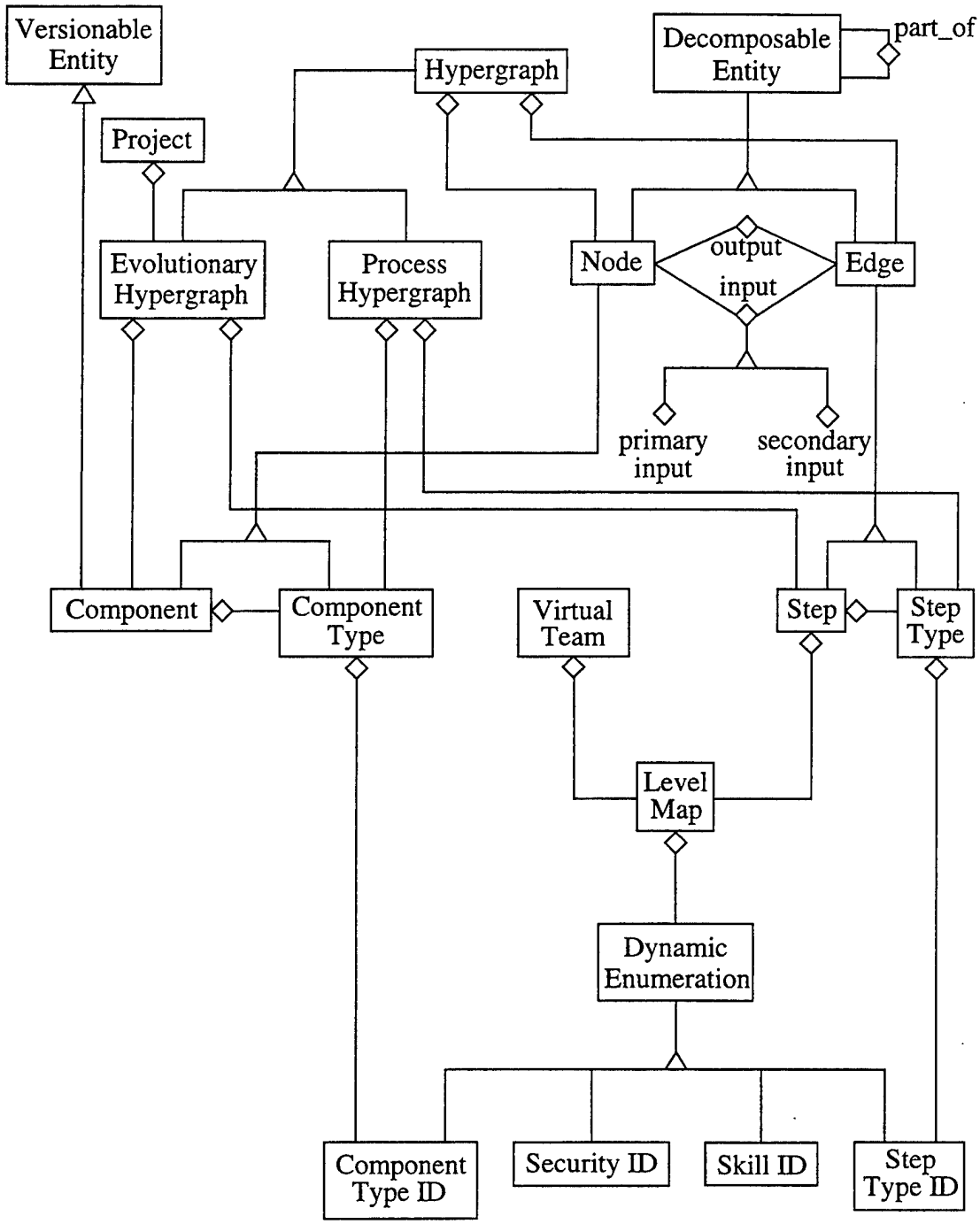


Figure 25: CASES class structure

In the recursive SPIDER construction level, CASES creates the primary input file: *input.p*, the secondary input file: *input.s*, the step content file: *step.cnt*, the component link subdirectory: *<component>*, the component subdirectories that are not in the process loop *loop.cfg,* and refined SPIDER subdirectories by extended version numbers, such as *<1>*, *<2>*, *<3>*, and *<4>*. In the component link subdirectory: *<component>* and the component subdirectories that are not in the process loop, CASES creates six files: *text.link, word.link, excel.link, data.link, url.link*, and *caps.link* to store component text files, MS Word files, MS Excel files, data files, URLs and CAPS files respectively. Under the refined SPIDER subdirectories, the file structure is also the recursive SPIDER construction level.

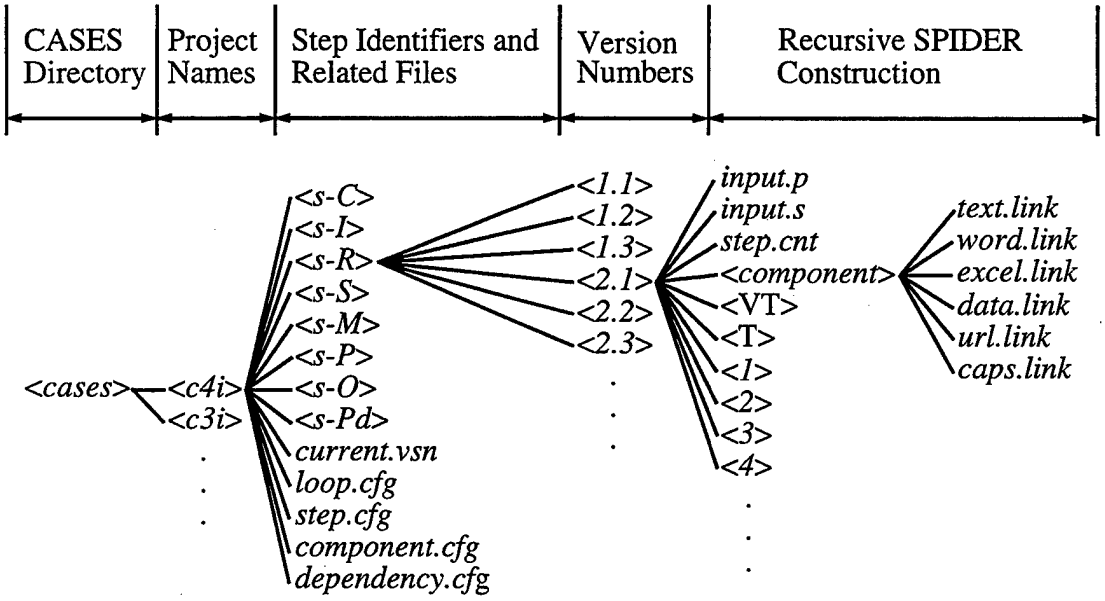


Figure 26: CASES file structure

The following describes the individual files as outlined in Figure 26 above:

- CASES creates the files: *step.cfg* and *component.cfg*, to store step types and component types respectively. Attributes of the file: *step.cfg* are *stepID, stepName,* and *stepDescription,* described in Table 1 of Appendix A. Attributes of the file: *component.cfg* are *componentId, componentName* and *componentDescription,*

described in Table 2 of Appendix A.

- CASES creates the files: *loop.cfg* and *dependency.cfg*, to store evolution processes and dependency related data respectively. Attributes of the file: *loop.cfg* are *EHLName,* and *EHLPath* described in Table 3 of Appendix A. Attributes of the file: *dependency.cfg* are *step, loopName, outputComponent, primaryInput* and *secondaryInput,* described in Table 4 of Appendix A.
- CASES creates the files: *current.vsn* to store current loop and step related data. Attributes of the file: *current.cfg* are *currentStep, currentLoop, currentVariant* and *currentVersion,* described in Table 5 of Appendix A.
- CASES creates the files: *step.cnt* to store step related data. Attributes of the file: *step.cnt* are *stepVersion, status, skill, skillLevel, securityLevel, evaluation, evaluator, organizer, predecessor, priority, estimatedDuration, deadline, earliestStartTime, finishTime* and *manager* described in Table 6 of Appendix A.
- CASES creates the files: *input.p* and *input.s* to store primary and secondary input components of a step respectively. Due to only one attribute in the files: *input.p* and *input.s,* we do not specify any attribute name in the files, described in Table 7 and 8 of Appendix A.
- CASES creates the files: *text.link, data.link, url.link,* and *caps.link* to store connections of text files, stakeholder data files, URLs, and CAPS files respectively. Due to only one attribute in the files: *text.link, wrod.link, excel.link, data.link, url.link* and *caps.link,* we do not specify any attribute name in the files, described in Table 9, 10, 11, 12, 13 and 14 of Appendix A.
- CASES creates a directory: *<stakeholder>* that is the same directory level as *<cases>* to construct stakeholder data repositories. CASES creates several stakeholder data files to store stakeholder data under the directory: *<stakeholder>*. Attributes of the files are *ID, name, skill, skillLevel, securityLevel, email, telephone, fac, address, majorJobs* and *minorJobs,* described in Table 15 of Appendix A.

C. USER INTERFACE

The user interface of CASES, shown in Figure 27, includes five portions: project, automated version control, SPIDER, tools, and job schedule. CASES' functions are embedded in the following menu bars, shown as Figure 28, 29, 30, 31, 32, 33 and 34 in Appendix B: *Project, Automated Version Control, SPIDER, Tools,* and *Job Schedule.*

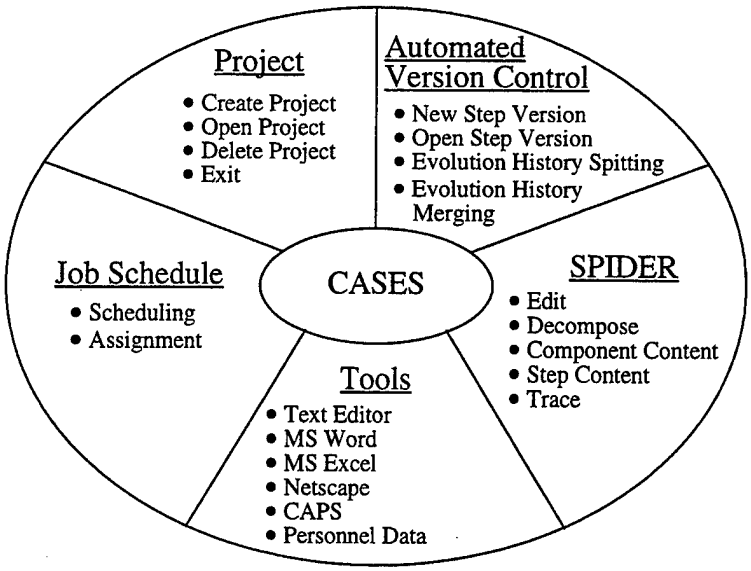


Figure 27: CASES user interfaces

1. Project

There are four menu items: *Create Project, Open Project, Delete Project,* and *Exit*, under the *Project* menu bar in the *CASES* main frame, shown as Figure 30 in Appendix B. When CASES users (briefly users) select a menu item *Create Project* in the menu bar *Project* of *CASES* main frame, it means they are creating a new project. When the users select a menu item *Open Project* in the menu bar *Project* of *CASES* main frame, it means they are proceeding an undergoing project or opening a previous project.

CASES provides functions to decide object types of a hypergraph and relationships among objects. The users have to define step and component types with identifiers and build their dependencies in a SPIDER in advance.

We create a subdirectory with project name in directory: <*cases*> to deal with software evolution processes and to record different software evolution activities for each project. At the beginning, if directory: <*cases*> is empty (no subdirectory), menu items:

Open Project and *Delete Project* are grayed. Menu items: *Open Project* and *Delete Project* are available if a project subdirectory is created in directory: <*cases*>.

The menu items: *New Project, Open Project, Delete Project* and *Exit* under the menu bar *Project* can be designed as follows:

a. Create project

We design a *Project* menu bar, shown as Figure 30 in Appendix B to get a new project name from *Project Name* item, shown as Figure 35 in Appendix B. CASES creates a subdirectory with the new project name under the directory: <*cases*> after the users press the *OK* button of the *Create Project* frame, and CASES ignores the new project name after the users press the *Cancel* button of the *Create Project* frame. If the new project name has already existed, the error messages will show up. After the users finish creating a project, the *Project Schema* frame appears up as Figure 39 in Appendix B and provides four frames: *Step Type, Component Type, Evolution Process,* and *Dependency*, for the users to enter related data.

(1) Step type

The *Step Type* in the *Project Schema* frame, shown as Figure 39 and 40 in Appendix B, includes four data items: *Step Type Id, Step Type Name, Existing Step Types*, and *Step Type Description*. If *step.cfg* file under a specified project directory does not exist, CASES will create it and append the step type data to it while the *Add* button is pressed. If *step.cfg* file under a specified project directory has already existed, the users can add a record by pressing the *Add* button, edit a record by pressing the *Edit* button, and delete a record by pressing the *Delete* button. The following are the requirements of the users' manipulations for creating step types:

- When the users add a record, the processes are as follows: typing data in the *Step Type Id,* the *Step Type Name* and the *Step Type Description* and pressing the *Add* button;
- When the users edit a record, the processes are as follows: getting the data of the *StepType Id* from the *Existing Step Types* combo box, shown as Figure 40 in Appendix B, making some modification in the *Step Type Name* or the *Step Type*

Description and pressing the *Edit* button;

- When the users delete a record, the processes are as follows: getting the data of the *Step Type Id* from the *Existing Step Types* combo box and pressing the *Delete* button;
- When the users press the *Clear* button, the data of the items in the *Step Type* frame will be cleared;
- When the users press the *Save* button, the data of the items will be saved in *step.cfg*;
- When the users press the *Finish* button the data of the items will be saved in *step.cfg* and the screen will return to the *CASES* main frame, shown as Figure 29 in Appendix B.

(2) Component type

The *Component Type* in *Project Schema* frame, shown as Figure 41 and 42 in Appendix B, includes four data items: *Component Type Id, Component Type Name, Existing Component Type*, and *Component Type Description*. If *conponent.cfg* file under a specified project directory does not exist, CASES will create it and append the component type data to it while the *Add* button is pressed. If *conponent.cfg* file under a specified project directory has already existed, the users can add a record by pressing the *Add* button, edit a record by pressing the *Edit* button, and delete a record by pressing the *Delete* button. The following are the requirements of the users' manipulations for creating component types:

- When the users add a record, the processes are as follows: typing data in the *Component Type Id,* the *Component Type Name* and the *Component Type Description* and pressing the *Add* button;
- When the users edit a record, the processes are as follows: getting the data of the *Component Type Id* from the *Existing Component Types* combo box, shown as Figure 42 in Appendix B, making some modification in the *Component Type Name* or the *Component Type Description* and pressing the *Edit* button;
- When the users delete a record, the processes are as follows: getting the data of the *Component Type Id* from the *Existing Component Type* combo box and pressing the *Delete* button;
- When the users press the *Clear* button, the data of the items in the *Component Type* frame will be cleared;
- When the users press the *Save* button, the data of the items will be saved in *component.cfg*; and

- When the users press the *Finish* button the data of the items will be saved in *component.cfg* and the screen will return to the *CASES* main frame, shown as Figure 29 in Appendix B.

(3) Evolution process

The *Evolution Process* in the *Project Schema* frame, shown as Figure 43 and 44 in Appendix B, includes three data items: *Evolution Process Name, Evolution Process,* and *Existing Evolution Process*. If *loop.cfg* file under a specified project directory does not exist, CASES will create it and append the evolution process data to it while the *Add* button is pressed. If *loop.cfg* file under a specified project directory has already existed, the users can add a record by pressing the *Add* button, edit a record by pressing the *Edit* button, and delete a record by pressing the *Delete* button. The following are the requirements of the users' manipulations for creating software evolution processes:

- When the users add a record, the processes are as follows: typing data in the *Evolution Process Name* and the *Evolution Process* and pressing the *Add* button;
- If the users edit a record, the processes are as follows: getting the *Evolution Process Name* from the *Existing Evolution Process* combo box, shown as Figure 44 in Appendix B, modifying the *Evolution Process Name* or *Evolution Process* and pressing the *Edit* button;
- When the users delete a record, the processes are as follows: getting the *Evolution Process Name* from the *Existing Evolution Process* combo box and pressing the *Delete* button;
- When the users press the *Clear* button, the items in the *Evolution Process* frame will be cleared;
- When the users press the *Done* button, the item data will be saved in *loop.cfg* and the screen will return to the *Dependency* frame, shown as Figure 45 in Appendix B; and;
- When the users press the *Finish* button, the item data will be saved in *loop.cfg* and the screen will return to the *CASES* main frame, shown as Figure 29 in Appendix B.

(4) Dependency

The *Dependency* in the *Project Schema* frame, shown as Figure 45, 46, 47, 48, and 49 in Appendix B includes four data items: *Evolution Process, Step Types, Output Component Type, Primary Input Component Type,* and *Secondary Input*

Component Type(s). The following are the requirements of the users' manipulations for editing dependencies:

- When the users edit a record in the *Secondary Input Component Type* data item, the processes are as follows: getting the *Evolution Process* from the *Evolution Process* combo box, shown as Figure 46 in Appendix B, getting the *Step Type* from the *Step Type* combo box, shown as Figure 47 in Appendix B, and making some modifications through the *Component Type List* panel shown as Figure 48 in Appendix B, after pressing the *Secondary Input Component Type(s)* button shown as Figure 47 in Appendix B.
- In the *Evolution Process* frame, when the users press the *Cancel* button, the data of items will be ignored; when the users press the *OK* button, the data of items will be saved in *dependency.cfg*; and when the users press *Finish* button, the data of items will be saved in *dependency.cfg* and the screen will return to the *CASES* main frame, shown as Figure 29 in Appendix B.

b. Open project

The following are the requirements of the users' manipulations for opening a project:

- When the users select the menu item *Open Project* in the menu bar *Project,* the *Open Project* file chooser shows up as Figure 36 and 37 in Appendix B. A list of project names in the file chooser is obtained from the project subdirectories under the directory: <*cases*>;
- After the users select one of the project subdirectories in the directory: <*cases*> and press *Open* button, the confirmation message shows up as Figure 38 in Appendix B: *Do you want to open Project Schema?*
- If the *Yes* button is pressed, the *Project Schema* frame shows up as Figure 39 in Appendix B; and
- If the *No* button is pressed, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

c. Delete project

The following are the requirements of the users' manipulations for deleting a project:

- When the users select the menu item *Delete Project* in the menu bar *Project,* the *Delete Project* panel shows up as Figure 50 and 51 in Appendix B. There is a combo box of project names in the *Delete Project* panel;
- When the users select a project name in the combo box and press the *OK* button,

all the files under the selected project directory will be deleted; and

- After the *OK* or *Cancel* button is pressed, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

d. Exit

If the *Exit* button is pressed, the *CASES* main frame disappears.

2. Automated version control

There are four menu items: *Create Step Version, Open Step Version, Evolution History Splitting,* and *Evolution History Merging* under the *Automated Version Control* menu bar, shown as Figure 31 in Appendix B. We create a one-record file *current.vsn* whose attributes are *current_loop, current_step (null is default), current_variant,* and *current_version* described in Table 5 of Appendix A for opening a step version.

After the users create, split, and merge a new step version by pressing *Create Step Version, Evolution History Splitting*, or *Evolution History Merging* respectively, CASES has to create new subdirectories under the steps in a software evolution process and save related primary input components in the file: *input.p* and related secondary input components in the file: *input.s* under the new subdirectories. The related primary input components in the file: *input.p* and secondary input components in the file: *output.p* can be automatically generated by CASES.

a. Create step version

The following are the requirements of the users' manipulations for creating a step version:

- When the users press the *Create Step Version* menu in the *Automated Version Control* menu bar, shown as Figure 31 in Appendix B, the *Create Step Version* frame shows up as Figure 52, 53, 54, and 55 in Appendix B which includes three data items: *Evolution Process, Current Variant Number,* and *New Step Version*;
- The users create a new step version in the data item: *New Step Version* by selecting the data items: *Evolution Process* and *Current Variant Number* from combo boxes and pressing the *New Step Version* button;
- When the users press the *OK* button, CASES instantiates step versions and creates step version subdirectories for each step in the specified software evolution process

and two files: *input.p* and *output.p* under the new subdirectories; and

- When the users press *Cancel* button, the data of items in the *Create Step Version* frame will be ignored and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

b. Open step version

The following are the requirements of the users' manipulations for opening a step version:

- When the users press the *Open Step Version* menu in the *Automated Version Control* menu bar, shown as Figure 31 in Appendix B, the *Open Step Version* frame shows up as Figure 56, 57, 58, and 59 in Appendix B. The *Open Step Version* frame includes three data items: *Evolution Process, Step Type,* and *Version*;
- The users open a step version in the data item: *Open Step Version* by selecting the data items: *Evolution Process, Step Type* and *Version* from combo boxes and pressing the *OK* button; and
- When the users press the *Cancel* button, the data of items in the *Open Step Version* frame will be ignored and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

c. Evolution history splitting

The following are the requirements of the users' manipulations for splitting a software evolution history:

- When the users press the *Evolution History Splitting* menu in the *Automated Version Control* menu bar, shown as Figure 31 in Appendix B, the *Evolution History Splitting* frame shows up as Figure 60, 61, 62, and 63 in Appendix B that includes three data items: *Evolution Process, Current Step Version,* and *New Step Version*;
- The users split a new step version in the data item: *New Step Version* by selecting the data items: *Evolution Process* and *Current Step Version* from combo boxes and pressing the *New Step Version* button;
- When the users press the *OK* button, CASES instantiates step versions and creates step version subdirectories for each step in the specified software evolution process and two files: *input.p* and *output.p* under the new subdirectories; and
- As the users press the *Cancel* button, the data of items in the *Evolution History Splitting* frame will be ignored and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

d. Evolution history merging

The following are the requirements of the users' manipulations for merging software evolution histories:

- When the users press the *Evolution History Merging* menu in the *Automated Version Control* menu bar, shown as Figure 31 in Appendix B, the *Evolution History Merging* frame shows up as Figure 64, 65, 66, 67, 68, 69, and 70 in Appendix B that includes three data items: *Evolution Process, Current Step Version, Merged Step Version, Variant Type,* and *New Step Version*;
- The users merge two step versions into a new step version in the data item: *New Step Version* by selecting the data items: *Evolution Process, Current Step Version, Merged Step Version* and *Variant Type* from combo boxes and pressing the *New Step Version* button;
- When the users press the *OK* button, CASES instantiates step versions and creates step version subdirectories for each step in the specified software evolution process and two files: *input.p* and *output.p* under the new subdirectories; and
- When the users press the *Cancel* button, the data of items in the *Evolution History merging* frame will be ignored and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

3. SPIDER

There are five menu items: *Edit, Decompose, Component Content, Step Content,* and *Trace* under the *SPIDER* menu bar, shown as Figure 32 in Appendix B. The following are the requirements of the users' manipulations for editing, decomposing and tracing a SPIDER, and reviewing the component content and the step content:

a. Edit

A file chooser shows up as Figure 71, 72, and 73 in Appendix B, when the users click the menu item *Edit* under the *SPIDER* menu bar. The file chooser lists all the steps including top-level steps and refined steps (containing atomic steps).

If the users select one of steps in the file chooser, the *SPIDER-Edit* panel shows up as Figure 74 in Appendix B. There are four data items in the panel: *Step Version, Output Component, Primary Input Component(s),* and *Secondary Input Component(s).* All the data items with their data will automatically show up in this panel. The data item *Step*

Version comes from the previous selection in the file chooser. The data item *Output Component* comes from the component part of the *Step Version*. The data item *Primary Input Component(s)* retrieves from the file: *input.p* and the data item *Secondary Input Component(s)* retrieves from the file: *input.s* in the directory of the opened step.

The users can modify and save all the data items in the *SPIDER-Edit* panel. Simultaneously, the users can open a window by pressing menu item *Trace* under the *SPIDER* menu bar to trace the SPIDERs and to browse the step and component content of the SPIDERs, and then decide how to modify SPIDER data in the *SPIDER-Edit* panel. The data in the *SPIDER-Edit* panel can be saved by pressing *Save* button and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

All the data items in the *SPIDER-Edit* panel can be deleted by pressing the *Delete* button. If users press the *Delete* button, it means that all the data related with the step should be deleted. So, CASES needs to clean up and delete all files: *input.p*, *input.s* and *step.cnt* and subdirectory *component.cnt.* After the users press the *Delete* button, the *Warning Message* panel will show up. If the users press the *Yes* button in the *Warning Message* panel, CASES removes this step directory and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B; otherwise if the users press the *No* button in the *Warning Message* panel, CASES does nothing and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

All the data items in the *SPIDER-Edit* panel can be cancelled by pressing *Cancel* button. This means that CASES will ignore the data modification in the *SPIDER-Edit* panel and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

b. Decompose

As the users click the menu item *Decompose* under the *SPIDER* menu bar, a file chooser of steps shows up as Figure 75 and 76 in Appendix B. This file chooser lists all the steps including a top-level step and refined steps (containing atomic steps).

When users select one of the steps in the file chooser, this selected step will be decomposed into a substep by the *SPIDER-Decompose* panel, shown as Figure 77 in Appendix B. There are four data items in the panel: *Step Version, Output Component, Primary Input Component(s),* and *Secondary Input Component(s).* All the data items with their data will automatically show up in this panel. The data item *Step Version* is combined by the higher level step version coming from the previous selection in the file chooser and a new substep number numbered by order. If the higher level step has been decomposed, the new substep number is the highest step number in the substeps plus one. If the higher level step has not been decomposed, the new substep number is 1. If this higher level step is a top-level step, there is a hyphen "-" in the *Step Version* between the higher level step version and the new substep number; otherwise, there is a dot "." between the higher level step version and the new substep number in the *Step Version.* The data item *Output Component* comes from the component part of the *Step Version.* The data item *Primary Input Component(s)* retrieves from the file: *input.p* and the data item *Secondary Input Component(s)* retrieves from the file: *input.s* in the higher level step directory. If this higher level step is a top-level step, CASES puts a hyphen "-" behind the higher level step version in the *Step Version* and waits for the users to put the step number in; otherwise, CASES puts a dot "." behind the higher level step version in the *Step Version* and waits for the users to put the step number in.

The users can modify and save all the data items in the *SPIDER-Decompose* panel. The users can open a window by menu item *Trace* under the *SPIDER* menu bar to trace the SPIDERs and to browse the step and component content of the SPIDERs and decide how to modify data in the *SPIDER-Decompose* panel. The data in the *SPIDER-Decompose* panel can be saved by pressing *Save* button and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B. After the users press the *Save* button, CASES creates a new subdirectory under the higher step and saves primary input data to the new file: *input.p* and secondary input data to the new file: *input.s* under this new

subdirectory. If the saved data in the new substep needs to be modified, the users can click menu item *Edit* to select this substep version and modify the data in the *SPIDER-Edit* panel.

There is no *Delete* button in the *SPIDER-Decompose* panel. If the users would like to delete a step, they need to go back to the *SPIDER-Edit* panel. The users can press the menu item *Edit* under the *SPIDER* menu bar, type the *Step Version* in and press the *Delete* button in *SPIDER Edit* panel to delete a step.

All the data items in the *SPIDER-Decompose* panel can be cancelled by pressing the *Cancel* button. This means that CASES will ignore the modification in the *SPIDER- Decompose* panel and return to the *CASES* main frame, shown as Figure 29 in Appendix B.

c. Component content

When the users select the menu item *Component Content* under the *SPIDER* menu bar, a file chooser of steps shows up as Figure 78, 79, and 80 in Appendix B. This file chooser lists all the steps including a top-level step and refined steps (containing atomic steps).

After the users select one of the steps in the file chooser, the *SPIDER-Component Content* panel shows up as Figure 81, 82, and 83 in Appendix B. In this panel there are one component combo box, *Available Components,* and six combo boxes of component content links: *Text Files, MS Word Files, MS Excel Files, Data Files, URLs,* and *CAPS Files*. The *Available Component* combo box includes primary input, secondary input and output components of the selected step, shown as Figure 82 in Appendix B. The components in the *Available Component* combo box are coverage of the components from the files: *input.p* and *input.s* and from the component part of the selected step in the file chooser.

When the users select one of the components in the *Available Component* combo box and press the *Add, Edit,* or *Delete* button in the *SPIDER-Component Content*

panel, CASES provides the *Connection Link Editor* frame for adding, editing, deleting, or browsing component content links, shown as Figure 84 to Figure 98 in Appendix B.

There is a *Component Content Type* combo box and a *Connection Links* list in the *Connection Link Edit* frame. If the users select one of the component content types: *txt.link, word.link, excel.link, data.link, url.link,* and *caps.link* in the *Component Content Type* combo box of the *Connection Link Edit* frame, the connection links of the component content shows up in the *Connection Links* list as Figure 85 in Appendix B. If the selected component is the newly created or decomposed component whose content has not been specified, there is nothing in the *Connection Links* list. If the selected component has been specified, there are connection links of the component content in the *Connection Links* list. It is important that CASES allow its users to specify more than one connection link to describe the component content of a component object. Therefore, it could happen that one component object has more than one connection link in the *Connection Links* list. After the users press the *Update* or *Exit* button in the *Connection Links* list, the screen returns to the *SPIDER-Component Content* panel.

When users press the *Save* button in the *SPIDER-Component Content* panel, CASES saves the connection links into one of the following connection link files under the directory of the selected component and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B:

- *txt.link* if the type of the connection link is *Text*,
- *word.link* if the type of the connection link is *MS Word*,
- *excel.link* if the type of the connection link is *MS Excel*,
- *data.link* if the type of the connection link is *Stakeholder*,
- *url.link* if the type of the connection link is *URL*, or
- *caps.link* if the type of the connection link is *CAPS*.

When the users press the *Cancel* button, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

d. Step content

A file chooser of the steps shows up as Figure 99, 100, and 101 in Appendix B, when the users select the menu item *Step Content* under the *SPIDER* menu bar. This file chooser lists all the steps including a top-level step and refined steps (containing atomic steps).

The users may select one of the steps in the file chooser and the *SPIDER-Step Content* panel appears as Figure 102 to Figure 110 in Appendix B. There are fifteen data items in the panel: *Step Version, Status, Skill, Security Level, Evaluation, Evaluator, Organizer, Predecessors, Priority, Estimated Duration, Deadline, Earliest Start Time, Finish Time,* and *Manager.* The data item *Step Version* automatically shows up in the *SPIDER-Step Content* panel after the users select a step in the file chooser. The data item *Step Version* comes from the previous selection in the file chooser. The data items *Evaluation* and *Estimated Duration* are values entered by project evaluators. The data items: *Deadline, Earliest Start Time,* and *Finish Time* are time values entered via a *Calendar* frame by project organizers, shown as Figure 115 in Appendix B. The rest of the data items in the *SPIDER-Step Content* panel are entered via combo boxes. If the data in the *SPIDER-Step Content* panel have already existed in the file: *step.cnt* under the directory of the step version, the data can be retrieved from this file. If the file: *step.cnt* does not exist under the directory of the step version, CASES needs to obtain the related data in the *SPIDER-Step Content* panel. The content of each combo box in the *SPIDER-Step Content* panel is described as follows:

- *Status*: lists *Proposed, Approved, Scheduled, Assigned, Decomposed, Abandoned* and *Completed* in a combo box to represent the step current status, shown as Figure 103 in Appendix B.
- *Skill*: lists a series of skill numbers, names, skill level numbers: 0, 1, 2, and 3, to represent skills and levels. the users need to select skills and skill levels in the *Skill List* table, shown as Figure 111, 112, and113 in Appendix B, and put them into the skill and level combo box, shown as Figure 104 in Appendix B.
- *Security Level*: lists a series of security level numbers: 0, 1, 2, 3, 4, and 5, in a combo box to represent the security levels, shown as Figure 105 and Appendix B.

- *Predecessors*: lists all atomic steps for allowing the users to select multiple atomic steps in the file chooser, shown as Figure 114 in Appendix B, and put them into the predecessor combo box, shown as Figure 108 in Appendix B.
- *Priority*: lists a series of priority numbers: 1, 2, 3, 4, and 5, in a combo box to represent the priority, shown as Figure 109 in Appendix B.
- *Organizer*: lists identifiers of stakeholders in a combo box, shown as Figure 107 in Appendix B.
- *Evaluator*: lists identifiers of stakeholders in a combo box, shown as Figure 106 in Appendix B.
- *Manager*: lists identifiers of stakeholders in a combo box, shown as Figure 110 in Appendix B.

All the data items in the *SPIDER-Step Content* panel allow users to modify data and save them under the directory of the step version. the users can open a window by menu item *Trace* to trace the SPIDERs and to browse the step and component content of the SPIDERs and to decide how to modify data in the *SPIDER-Step Content* panel. When the users press the *Save* button in this panel, CASES saves data into the file: *step.cnt* and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

All the data items in the *SPIDER-Step Content* panel can be deleted by pressing the *Delete* button. If the users press the *Delete* button, it means that all the data in the *SPIDER-Step Content* panel should be deleted. Therefore, CASES needs to delete the file: *step.cnt.* After the users press the *Delete* button, the *Warning Message* panel will show up: *All the data in the Step Content Panel will be deleted.* If users press the *OK* button in the *Warning Message* panel, CASES remove the file: *step.cnt* and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B, or if the users press the *Cancel* button in the *Warning Message* panel, CASES does nothing and the screen returns to the *CASES* main frame as in Figure 29 in Appendix B.

All the data of the items in the *SPIDER-Step Content* panel can be cancelled by pressing *Cancel* button. This signifies that CASES will ignore the modification in the *SPIDER-Step Content* panel and the screen returns to the *CASES* main frame, as shown in Figure 29 in Appendix B.

e. Trace

The users can select the menu item *Trace* under the *SPIDER* menu bar, and a file chooser of steps shows up as Figure 116 and 117 in Appendix B. This file chooser lists all the steps including a top-level step and refined steps (containing atomic steps).

There are seven menu buttons on the top of the panel: *Home, Forward, Backward, Trace, Decompose, Component Content,* and *Step Content* in the *SPIDER-Trace* panel, as in Figure 118 in Appendix B. There are four SPIDER data items in the panel: *Step Version, Output Component, Primary Input Component(s),* and *Secondary Input Component(s).* All the data items with their data will automatically appear up in this panel. The data item *Step Version* comes from previous selection in the file chooser. The data item *Output Component* comes from the component part of the *Step Version.* The data item *Primary Input Component(s)* retrieves from the file: *input.p* and the data item *Secondary Input Component(s)* retrieves from the file: *input.s* in the directory of the selected step.

(1) Trace

Whenever the users press the *Trace* button in the *Trace* panel, a *Component* table shows up as Figure 121 in Appendix B. The *Component* table lists all the primary and secondary input components of the selected step. If the users click one of the components in the *Component* table, CASES will obtain a new SPIDER whose output component is this clicked component, the step version will be put into a stack and all the data items with their data will automatically show up in the *Trace* panel. The *Step Version* is the combination of a string "s-" and the identifier of the previous selected component in the *Component* combo box, shown as Figure 122 in Appendix B. The data item *Output Component* comes from the previous selected component in the *Component* table. The data item *Primary Input Component(s)* retrieves from the file: *input.p,* and the data item *Secondary Input Component(s)* retrieves from the file: *input.s* in the directory of the selected component. The users can unlimitedly navigate another SPIDER by selecting a

component in the *Component* combo box until the selected component has no other input components.

(2) Decompose

When the users select the *Decompose* button in the *Trace* panel, a *Component* table appears as in Figure 123 in Appendix B. The *Component* table lists all the primary and secondary input components and the output component of the selected step. If the users select one of the components in the *Component* table and this component is an atomic component, an *error message panel* shows up: *This is an atomic component.* When the users press the *OK* button in the *Error Message* panel, the screen returns to the *SPIDER-Trace* panel. Otherwise, if the users click one of the components in the *Component* table and this component is not an atomic component, the lower level *Component* table appears as Figure 124 in Appendix B. The lower level *Component* table lists all the decomposed components of the selected component in the higher level *Component* table.

If the users click one of the steps in the lower level *Component* table, the *SPIDER-Trace* panel will obtain a new refined SPIDER of this selected step and the step version will be put into a stack. All the data items with their data will automatically show up in the *SPIDER-Trace* panel. The data item *Step Version* comes from the previous selection in the lower level *Component* table. The data item *Output Component* comes from the component part of the *Step Version*. The data item *Primary Input Component(s)* retrieve from the file: *input.p* and *Secondary Input Component(s)* retrieve from the file: *input.s* in the directory of the selected step. The users can unlimitedly navigate another decomposed SPIDER via the selected components of the *Component* combo box and steps of the file chooser until the selected component is an atomic component and the selected step is an atomic step.

(3) Component content

The users may select the *Component Content* button in the *SPIDER-Trace* panel, and a *Component* table shows up as Figure 126 in Appendix B. The

Component table lists all the primary and secondary input components and the output component of the selected step. If the users click one of the components in the *Component* table of the *SPIDER-Trace* panel, a *Review Component Content* panel shows up as Figure 127 in Appendix B. There is a *Component Content Type* combo box and a *Available Links* table in the *Review Component Content* panel. What component content types in the *Component Content Type* combo box are coverage of depends on what the component content files: *text.link, word.link, excel.link, data.link, url.link,* and *caps.link* exists in the directory of the selected component in the *Component* table of the *SPIDER-Trace* panel. If the users click one of the component content types in the *Component Content Type* combo box of the *Review Component Connection* panel, connection links of the component content will show up in the *Available Links* table as Figure 127 to Figure 130 in Appendix B.

If the selected component content file is *text.link*, the text files may show up in the *Available Links* table as Figure 129 and 130 in Appendix B. When the users press the *Exit* button in the *Review Component Content* panel, the screen returns to the *SPIDER-Trace* panel. The users can select one of the text files in the *Available Links* table and press the *Connect* button, and the *Notepad* text editor with the selected text shows up as Figure 131 in Appendix B. the users can read and modify text via the *Notepad* text editor. When the users exit the *Notepad* text editor, the screen returns to the *Review Component Content* panel, shown as Figure 130 in Appendix B.

If the selected component content file is *word.link*, the *MS Word* document files may show up in the *Available Links* table. When the users press the *Exit* button in the *Review Component Content* panel, the screen returns to the *SPIDER-Trace* panel. The users can select one of the *MS Word* document files in the *Available Links* table and press the *Connect* button, and the *MS Word* with the selected document appears. the users can read and modify text via the *MS Word*. The screen returns to the *Review Component Content* panel, shown as Figure 130 in Appendix B, when the users exit the *MS Word*.

If the selected component content file is *excel.link*, the *MS Excel* table files may show up in the *Available Links* table as. When the users press the *Exit* button in the *Review Component Content* panel, the screen returns to the *SPIDER-Trace* panel. When a user selects one of the *MS Excel* table files in the *Available Links* table and presses *Connect* button, the *MS Excel* with the selected table shows up. the users can read and modify data via the *MS Excel*. When the users exit the *MS Excel*, the screen returns to the *Review Component Content* panel, shown as Figure 130 in Appendix B.

If the selected component content file is *data.link*, the personnel data files may show up in the *Available Links* table. When the users press the *Exit* button in the *Review Component Content* panel, the screen returns to the *SPIDER-Trace* panel. When the users select one of the personnel data files in the *Available Links* list and press *Connect* button, the *Personnel Data* panel with personnel data shows up as Figure 144 in Appendix B. There are nine data items in the *Personnel Data* panel: *Id, Name, Skill, Security Level, E-mail Address, Telephone Number, Fax Number, Address,* and *On-hand Jobs.* There are three fields: *Job Name, Real Start Time,* and *Estimated Duration* in the *On-hand Jobs* table whose data are listed by pressing the two buttons: *Major Jobs* and *Minor Jobs*. the users only can read personnel data in the *Personnel Data* panel but can not modify any data in the panel. When the users press the *Exit* button in the *Personnel Data* panel, the screen returns to the *Review Component Content* panel, shown as Figure 130 in Appendix B.

If the selected component content file is *url.link*, the *URLs* may show up in the *Available Links* table. When the users press the *Exit* button in the *Review Component Content* panel, the screen returns to the *SPIDER-Trace* panel. When the users select one of the *URLs* in the *Available Links* list and press *Connect* button, *Netscape* shows up with the selected *URL*. The users can read data via *Netscape*. Upon exiting the *Netscape*, the screen returns to the *Review Component Content* panel, shown as Figure 130 in Appendix B.

If the selected component content file is *caps.link*, the *CAPS* code files may show up in the *Available Links* table. When the users press the *Exit* button in the

Review Component Content panel, the screen returns to the *SPIDER-Trace* panel. Then if the users select one of the *CAPS* code files in the *Available Links* table and press the *Connect* button, the *CAPS* shows up as Figure 140 in Appendix B. Users can read and modify code via the *CAPS*. The screen returns to the *View Component Content* panel, shown as Figure 130 in Appendix B, when the users exit the *CAPS*.

If the selected component is the newly created or decomposed component whose content has not been specified, there is nothing in the *Available Links* table. If the component has been specified, there are connection links of the component content in the *Available Links* table. It is important that CASES allow the users to specify more than one connection link to describe the component content in a component object. Therefore, one component object possibly has more than one connection link in the *Available Links* table.

(4) Step content

When users click the *Step Content* button in the *SPIDER-Trace* panel, the *SPIDER-Step Content* panel shows up as Figure 132 in Appendix B. the users only can read data in the *SPIDER-Step Content* panel but cannot modify any data in the text. After the users press the *Exit* button in the *SPIDER-Step Content* panel, the screen returns to the *SPIDER-Trace* panel, shown as Figure 120 in Appendix B.

(5) Home, forward, and backward

Buttons *Home, Forward,* and *Backward* are three functions to navigate SPIDERs that have already been traced as Figure 133,134, and 135 in Appendix B. If the users press the *Backward* button, the stack pointer will go back one step location in the stack. When the users press the *Forward* button, the stack pointer will go forward one step location in the stack. Upon pressing the *Home* button, the stack pointer will go to the first step location in the stack. According to the step location, all the data items with their data will automatically show up in the *Trace* panel and the components in the *Trace* and *Decompose* combo box will be updated.

If the users press the *Close* button in the *SPIDER-Trace* panel the screen returns to the *CASES* main frame, as in Figure 29 in Appendix B.

4. Tools

CASES provides six tool links to manage step and component content files: *Text Editor, MS Word, MS Excel, Netscape, CAPS,* and *Personnel Data,* illustrated in Figure 33 in Appendix B. The following are the requirements of the users' manipulations for using the tools:

a. Text editor

When the users select the menu item *Text Editor* under the *Tools* menu bar, the *Notepad* text editor appears as Figure 136 in Appendix B. The users can then open, modify, and save a text file by means of the *Notepad* text editor. If the users exit the *Notepad* text editor, the screen returns to the *CASES* main frame, as in Figure 29 in Appendix B.

b. MS Word

If the users select the menu item *MS Word* under the *Tools* menu bar, *MS Word* shows up as Figure 137 in Appendix B. The users can then open, modify, and save a document file by means of *MS Word.* If the users exit *MS Word*, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

c. MS Excel

When the users select the menu item *MS Excel* under the *Tools* menu bar, *MS Excel* shows up as Figure 138 in Appendix B. The users can then open, modify, and save a table file by means of *MS Excel.* If the users exit *MS Excel*, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

d. Netscape

Netscape shows up as Figure 139 in Appendix B if the users select the menu item *Netscape* under the *Tools* menu bar. The users can then navigate the related web

site by means of *Netscape* with *URLs*. If the users exit *Netscape*, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

e. CAPS

As the users select the menu item *CAPS* under the *Tools* menu bar, *CAPS* shows up as Figure 140 in Appendix B. The users can then open the PSDL editor, retrieve and edit *CAPS* module files, and execute integrated programs by means of *CAPS*. If the users close the *CAPS*, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

f. Personnel data

When the users select the menu item *Personnel Data* under the *Tools* menu bar, the submenu items *Edit, Add,* and *Delete* show up as Figure 141 in Appendix B.

If the users select the submenu item *Add* under the menu item *Personnel Data*, the *Personnel Data* panel appears up as Figure 142 in Appendix B. There are eight data items in this *Personnel Data* panel: *Id, Name, Skill, Security Level, E-mail Address, Telephone Number, Fax Number, Address,* and *On-hand Jobs.* The data items: *Id, Name, E-mail Address, Telephone Number, Fax Number,* and *Address* are entered in the text fields The data item *Skill* is entered by the Skill table, shown as Figure 143 in Appendix B. The data item *Security level* is entered by a combo box. The content of skill and security level in the *Personnel Data* panel is described as follows:

- *Skill* lists a series of skill numbers, names, and skill level numbers 0, 1, 2, and 3, to represent skills and levels. the users need to select skills and skill levels in the *Skill List* table and put them into the skill and level combo box.
- *Security Level* lists a series of security level numbers 0, 1, 2, 3, 4, and 5, in a combo box to represent the security levels.

If the users finish adding data and press the *Save* button in the *Personnel Data* panel, the personnel data will save in a file by the stakeholder's identifier under the directory <stakeholder> and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B. If the users press the *Clear* button in the *Personnel Data* panel, the data

in the *Personnel Data* panel will be cleaned up. If the users press the *Exit* button in the *Personnel Data* panel, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

When the users select the submenu item *Edit* under the menu item *Personnel Data*, a file chooser of stakeholder identifiers shows up as Figure 145 in Appendix B. This file chooser lists all the stakeholder identifiers that have already been created by adding personnel data. The users can edit or modify personnel data in the *Personnel Data* panel.

After the users select the submenu item *Delete* under the menu item *Personnel Data*, the *Delete Personnel Data* panel shows up as Figure 147 and 149 in Appendix B. When the users press the *Browse* button in the *Delete Personnel Data* panel, a file chooser of stakeholders shows up as Figure 148 in Appendix B. This file chooser lists all the stakeholders that have already been created by adding personnel data. the users can delete the selected personnel data file after pressing the *OK* button in the file chooser. After the *OK* or *Cancel* button is pressed, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

5. Job schedule

CASES provides two menu items under the *Job Schedule* menu bar to schedule and to assign jobs: *Scheduling* and *Assignment,* shown as Figure 34 in Appendix B. The following are the requirements of the users' manipulations for scheduling and assigning the jobs:

a. Scheduling

When the users select the menu item *Scheduling* under the *Job Schedule* menu bar, the *Job Scheduling* panel appears as Figure 150 to Figure 155 in Appendix B. The *Job Scheduling* panel includes a *Job Scheduling Policy* combo box and a *Job Scheduling* table. There are seven scheduling policy menu items in the Job Scheduling Policy combo box: *High Priority First, Minimum Deadline First (Min_D), Minimum*

*Estimated Duration First (Min_D), Minimum Earliest Start Time First (Min_S), Minimum Laxity First (Min_L), Min_D * W + Min_E * (1 - W),* and *Min_D * W + Min_S * (1 - W),* shown as Figure 151 in Appendix B. There are eight fields in the Job Scheduling table: *Job ID, Priority, Deadline, Estimated Duration, Earliest Start Time, Laxity, Min_D * W + Min_E * (1 - W),* and *Min_D * W + Min_S * (1 - W).* The *Job Scheduling* table will update the sorting result of jobs after the users select a job scheduling policy in the *Job Scheduling Policy* combo box. The data in the *Job Scheduling* table are retrieved or calculated from the data in the related step content file: *step.cnt.*

When the users select *Min_D * W + Min_E * (1 - W),* and *Min_D * W + Min_S * (1 - W)* menu items, the *Weight* panel shows up as Figure 154 in Appendix B. the users should enter a weight value to the text field whose interval is from 0.0 to 1.0. If the users type invalid value, the error message *Invalid Value* will show up. After the users press the *Done* button in the *Weight* panel, the *Job Scheduling* table will update the sorting result of jobs and the screen returns to the *Job Scheduling* panel as Figure 155 in Appendix B. As the users press the *Exit* panel in the *Job Scheduling* panel, the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B.

b. Assignment

When the users select the menu item *Assignment* under the *Job Schedule* menu bar, the *Job Assignment* panel shows up as Figure 156 to Figure 160 in Appendix B. The *Job Assignment* panel includes five data items *Job ID, Security Level, Deadline, Estimated Duration,* and *Required Skills,* three assignment buttons *Filter by Security Level, Filter by Required Skills,* and *Assign the Job*, and a *Stakeholder List* table.

The data item *Job ID* comes from the first job identifier of the sorting result in the *Job Scheduling* panel after the users finish selecting a job scheduling policy in the *Job Scheduling Policy* combo box as Figure 155 in Appendix B. The data items *Deadline, Estimated Duration,* and *Required Skills* are retrieved from the related step content file, *step.cnt.* When the users press the *Required Skills* button in the *Job Assignment* panel, the

Required Skills panel shows up as Figure 157 in Appendix B. The required skills are listed in the table of the *Required Skills* panel that includes the following fields: *Skill ID, Skill Name,* and *Skill Level.* When the users press the *Exit* button, the screen returns to the *Job Assignment* panel as Figure 158 in Appendix B.

The users must sequentially press three assignment buttons, *Filter by Security Level, Filter by Required Skills,* and *Assign the Job*, to assign the job shown in the *Job Assignment* panel to a stakeholder. When a CASES user presses the *Filter by Security Level* button, CASES compares the security level of the assigning job and the stakeholders in the directory <stakeholder>, and filters the stakeholders in the *Stakeholder List* table that includes two fields: *Stakeholder ID* and *Security Level*, shown as Figure 158 in Appendix B. After the users press *Filter by Required Skills* button, CASES compares the skill levels of the assigning job and the stakeholders in the directory <stakeholder>, and filters the stakeholders in the *Stakeholder List* table that includes two fields: *Stakeholder ID* and *Match Number of Required Skills,* shown as Figure 159 in Appendix B. After CASES user press the *Assign the Job* button, CASES assigns the job to the first stakeholder in the *Stakeholder List* table and the screen returns to the *CASES* main frame, shown as Figure 29 in Appendix B. Whenever the users press the *Exit* panel in the *Job Scheduling* panel, the screen also returns to the CASES main frame, but CASES does not assign the job to any stakeholders.

VII. EVOLUTION OF C4I SYSTEMS

This chapter formalizes the evolution of C4I systems via a relational hypergraph model with primary-input-driven and secondary-input-driven dependencies to validate the CASES. The evolution of C4I system is modeled by a multidimensional architecture containing successive software evolution steps and related software evolution components. We analyze a domain-specific software development architecture and give a standard software evolution process in developing a prototype system as well as a production software system. This model is applied in several real-time prototyping systems especially for Command and Control applications [HARN99d].

C4I systems are extremely complex systems. A variety of factors influence the performance of C2 architectures — technology, environmental stressors, rules of engagement and so on. In order to improve the performance of alternative organizational structures in a simulated joint operational environment, an A2C2 (Adaptive Architectures for Command and Control) program has been ongoing for approximately four years [KEMP98]. Systematic approaches to C4I systems evaluation, including experimental design, scenario modification, planning and developing training materials, conducting player training, managing execution, and data collection, etc., work well to elicit evolution issues with new requirements.

A. FEATURES OF C4I SYSTEMS

C4I systems include the following preliminary features [LUQI92]:

- Their use in strategic defense applications makes accuracy and reliability critical.
- They are influenced by many people, by organizations, and by policies, so their requirements are complex and difficult to determine.
- Their design depends on techniques to guarantee that hard real-time constraints will be met both in large distributed systems connected by long-haul networks and in local distributed systems with many hardware structures.
- Their complex, dynamic interfaces make it almost impossible to deal with changes

in requirements.

Computer hardware and software enhances the feasibility and functionality of C3I systems. Computers within C4I systems not only play an interface role with external platforms, but also provide real-time embedded software and intelligent software to support commanders in decision making, routine program processing, data computing, etc. C4I computer software is too large, complex, dedicated, intractable, and mutable to meet mission needs under the development circumstances [LEE98]. As with any large systems, their development is costly, and the current low productivity of software development aggravates the problem [SOMM96].

B. APPROACHES

1. Prototyping a C4I system

Due to rapid requirement changes in the evolution environment, we use the CAPS to help develop C4I systems. CAPS is an easy-to-use, visual, integrated tool that can be used to rapidly design real-time applications utilizing its PSDL editor, reusable software database, program generator, real-time scheduler, and so on.

A generic C4I system includes the following external interfaces [LUQI92]:

- C4I users: could be a composite warfare commander, officer in tactical command, warfare area commander, tactical action officer, communication officer, etc.
- Communication links: any digital communication system capable of transmitting and receiving digital messages.
- Platform sensors: any locally-mounted device capable of identifying azimuth, elevation, velocity, and/or heading if a contact or track is considered to be a platform.
- Navigation system: a system that provides a platform with its own positioning, course, velocity, and time data.
- Weapons system: this interface, if it exists, makes the weapons status information available to the battle manager.

The software, with related data repositories, of a generic C4I system is mounted in workstations. Software requirements are based on different and specific C4I systems.

2. Automating evolution processes

CASES is a system that manages and controls all the activities that change a software system and the relationships among these activities. The whole process for software evolution is described in Figure 16.

C. C4I/MD SYSTEM EVOLUTION

A C4I system called the Missile Defense (MD) system has been developed by CAPS. The MD system provides defense functions to a specified area or a nation so that it can be extended to a TMD (Theater Missile Defense) system and a NMD (National Missile Defense) system. TMD and NMD systems are extremely complicated. In order to develop an MD system, we need to know how system requirements are obtained. The first prototype MD system is very simple and immature, but it gradually gets feasible as the MD system goes through the evolution process several times.

Initially, the assumptions of a MD system are as follows:

- There are ten bases where certain kinds of land-to-air missiles are deployed.
- Radar systems can accurately detect target objects.
- The radar coordinate system is 360 degrees increasing clockwise.
- The coordinate system is two-dimensional.
- The path of target objects is straight to an attack destination.
- All the attack destinations of target objects are on the center of a map which is the origin in the coordinate system.
- In one execution process, the MD system simulates only one target object and one defending missile.
- The speed of target objects and defending missiles is constant.
- The safety point is specified on a safety ring.
- The intersection point of target objects and defending missiles is on the safety point.
- Defending missiles can accurately destroy target objects at the safety point.
- There is no contingency plan in the case the missile fails to destroy the target object.
- Commanders take time making decisions with the computer.
- The position of target objects, the speed of target objects, the position of missile

bases, and decision delay time are manually manipulated by users.

Assumptions can be generalized and specialized according to requirements from users. Users provide criticisms according to their own view with different generalization and specialization ideas. The issue analysts collect criticisms into different generation and specialization issues. The requirement analysts provide project managers some information about requirements, like cost-benefit analysis and resource constraints. The decisions made by managers to new requirements of next generation prototype decides whether assumptions are generalized or specialized.

In the first prototype of MD systems, some assumptions are transferred to requirements. In the second prototype of MD systems, some other assumptions will be released into new requirements.

In the evolution processes of MD systems, criticism, issue, and requirement components can be described as hypermedia types, such as text, pictures, video, etc.; specification components can be described as data flow diagrams in PSDL editor and software programs; module and program components can be described as software programs.

1. Initial prototype of MD systems

a. Requirement analysis step

The initial prototype requirements can be generated by the requirement analysis step from the above assumptions. The top-level requirement component *R1.1* is a set of atomic requirement components that are categorized into three groups: *R1.1-1, R1.1-2,* and *R1.1-3*. Requirement components are presented as the following text descriptions stored by individual files:

R1.1-1: *The MD system must provide an output interface for users to monitor the data concerning base position, target position, target situation, missile direction, missile speed, target and missile intersection point, current time, and missile reach time.*

R1.1-1.1: *The output interface must provide the function of monitoring the base position data.*

R1.1-1.2: *The output interface must provide the function of monitoring the target position data.*

R1.1-1.3: *The output interface must provide the function of monitoring the target situation data.*

R1.1-1.4: *The output interface must provide the function of monitoring the missile direction data.*
R1.1-1.5: *The output interface must provide the function of monitoring the missile speed data.*
R1.1-1.6: *The output interface must provide the function of monitoring the target and missile intersection point data.*
R1.1-1.7: *The output interface must provide the function of monitoring the current time data.*
R1.1-1.8: *The output interface must provide the function of monitoring the missile reach time data.*
R1.1-2: *The MD system must provide an input interface for users to enter the data concerning base location, target location, target speed, and delay time of decision making.*
R1.1-2.1: *The input interface must provide the function of entering the base location data.*
R1.1-2.2: *The input interface must provide the function of entering the target location data.*
R1.1-2.3: *The input interface must provide the function of entering the target speed data.*
R1.1-2.4: *The input interface must provide the function of entering the delay time of decision making data.*
R1.1-3: *The MD system must provide a control system capable of efficiently transferring, generating, receiving, and computing information in real time.*
R1.1-3.1: *The control system must provide the function of transferring base locations to base coordinates.*
R1.1-3.2: *The control system must provide the function of transferring target locations to target coordinates and computing coordinates of safety point.*
R1.1-3.3: *The control system must provide the function of receiving data of base coordinates, target coordinates, coordinates of safety point, target speed, and delay time of decision making; and computing data of base position, target position, missile direction, missile speed, target and missile intersection point, and missile reach time.*
R1.1-3.4: *The control system must provide the function of generating system time.*
R1.1-3.5: *The control system must provide the function of receiving data of missile reach time and system time; and computing data of missile reach time, current time, and target situation.*

b. Specification design step

The initial prototype specifications can be generated by the specification design step from the above atomic requirement components. Top-level specification component *S1.1* is a set of atomic specification components that are categorized into two groups: *S1.1-1* and *S1.1-2*. Specification components are presented as following specification files with IMPLEMENTATION and SPECIFICAION PSDL code:

S1.1-1: *c4i.gui_3.imp.psdl & c4i.gui_3.spec.psdl*
S1.1-1.1: *c4i.b_p_68.imp.psdl & c4i.b_p_68.spec.psdl*
S1.1-1.2: *c4i.t_p_71.imp.psdl & c4i.t_p_71.spec.psdl*
S1.1-1.3: *c4i.t_a_47.imp.psdl & c4i.t_a_47.spec.psdl*
S1.1-1.4: *c4i.m_d_38.imp.psdl & c4i.m_d_38.spec.psdl*
S1.1-1.5: *c4i.m_s_41.imp.psdl & c4i.m_s_41.spec.psdl*
S1.1-1.6: *c4i.int_35.imp.psdl & c4i.int_35.spec.psdl*
S1.1-1.7: *c4i.c_t_50.imp.psdl & c4i.c_t_50.spec.psdl*
S1.1-1.8: *c4i.m_r_t_44.imp.psdl & c4i.m_r_t_44.spec.psdl*
S1.1-1.9: *c4i.b_l_56.imp.psdl & c4i.b_l_56.spec.psdl*
S1.1-1.10: *c4i.t_l_59.imp.psdl & c4i.t_l_59.spec.psdl*
S1.1-1.11: *c4i.t_s_62.imp.psdl & c4i.t_s_62.spec.psdl*
S1.1-1.12: *c4i.d_t_65.imp.psdl & c4i.d_t_65.spec.psdl*
S1.1-1.13: *c4i.gui_event_monitor_53.imp.psdl & c4i.gui_event_monitor_53.spec.psdl*

S1.1-2: *c4i.ctrl_6.imp.psdl & c4i.ctrl_6.spec.psdl*
S1.1-2.1: *c4i.trans1_114.imp.psdl & c4i.trans1_114.spec.psdl*
S1.1-2.2: *c4i.trans2_117.imp.psdl & c4i.trans2_117.spec.psdl*
S1.1-2.3: *c4i.ctrller_120.imp.psdl & c4i.ctrller_120.spec.psdl*
S1.1-2.4: *c4i.time_gen_126.imp.psdl & c4i.time_gen_126.spec.psdl*
S1.1-2.5: *c4i.target_123.imp.psdl & c4i.target_123.spec.psdl*

The IMPLEMENTATION and SPECIFICATION PSDL code is automatically generated by CAPS through data flow diagrams in the PSDL editor. In Figure 161, a top-level data flow diagram *c4i* includes *gui* and *ctrl* operators with 4 data streams from operator *gui* to operator *ctrl* and 8 data streams from operator *ctrl* to operator *gui*. Furthermore, Figure 162 shows a decomposed data flow diagram – *gui* including 13 operators and their data streams, and Figure 163 shows a decomposed data flow diagram *ctrl* including 5 operators and their data streams.

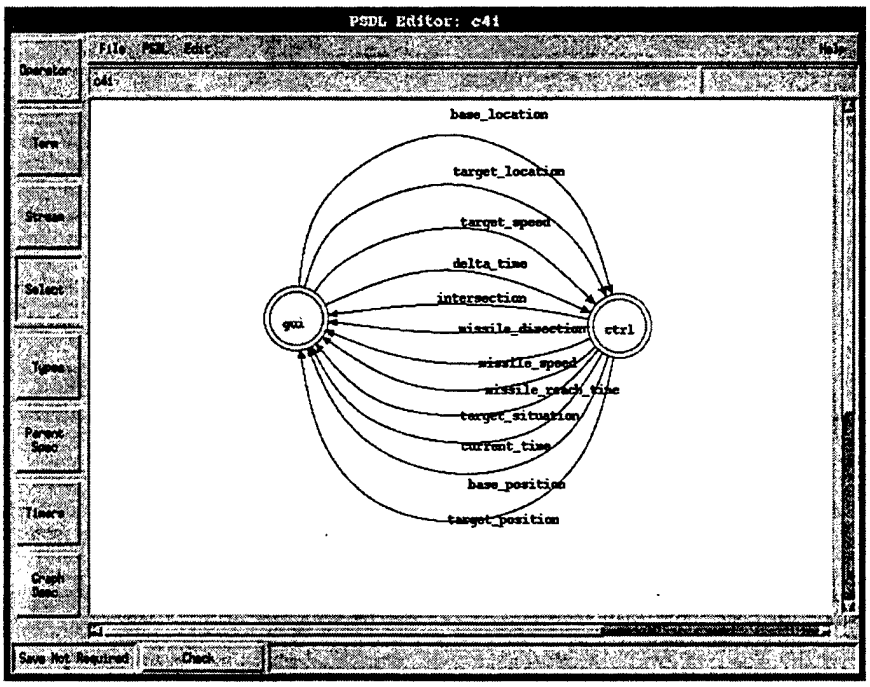


Figure 161: A top-level data flow diagram – *c4i*

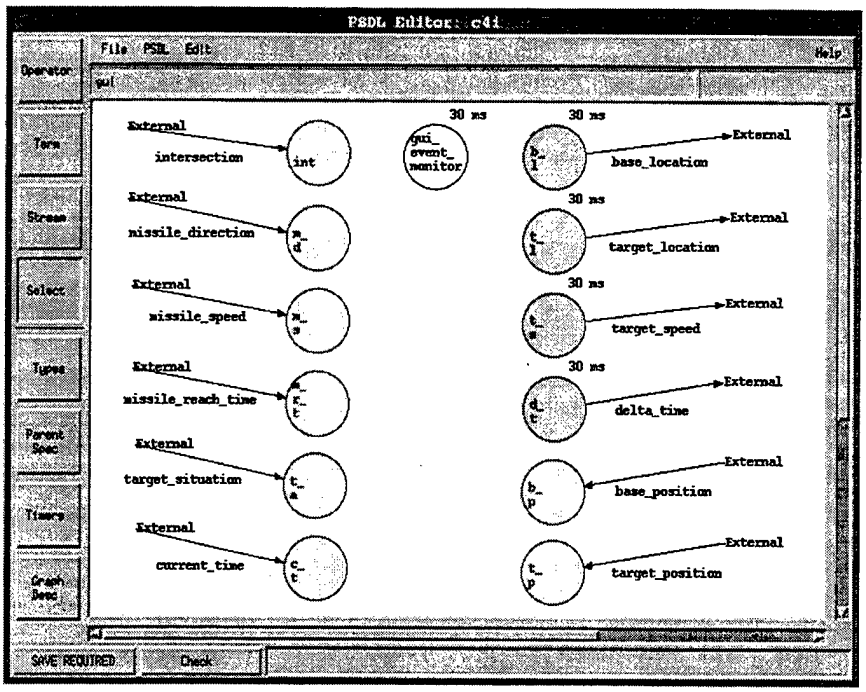


Figure 162: A decomposed data flow diagram – *gui*

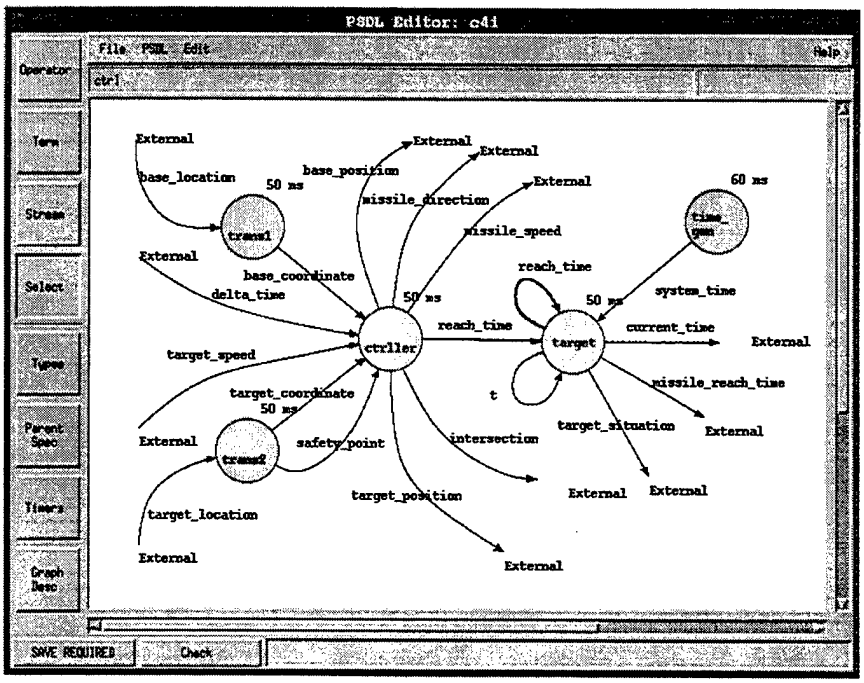


Figure 163: A decomposed data flow diagram – *ctrl*

c. *Module implementation step*

The initial prototype modules can be generated by the module implementation step from the preceding atomic specification components. The top-level module component *M1.1* is a set of atomic module components that are categorized into two groups: *M1.1-1* and *M1.1-2*. Files in *M1.1-1* are automatically generated by CAPS through TAE Plus (Transportable Applications Environment Plus). Files with extension file name "*.at*" are automatically generated by CAPS from related specification components. Files with extension file name "*.a*" are implemented by designers. Therefore, atomic module components are presented as the following files with Ada code:

M1.1-1: *Interface*
- *M1.1-1.1:* *c4i.b_p_68.a*
- *M1.1-1.2:* *c4i.t_p_71.a*
- *M1.1-1.3:* *c4i.t_a_47.a*
- *M1.1-1.4:* *c4i.m_d_38.a*
- *M1.1-1.5:* *c4i.m_s_41.a*
- *M1.1-1.6:* *c4i.int_35.a*
- *M1.1-1.7:* *c4i.c_t_50.a*
- *M1.1-1.8:* *c4i.m_r_t_44.a*
- *M1.1-1.9:* *c4i.b_l_56.a*
- *M1.1-1.10:* *c4i.t_l_59.a*
- *M1.1-1.11:* *c4i.t_s_62.a*
- *M1.1-1.12:* *c4i.d_t_65.a*
- *M1.1-1.13:* *c4i.gui_event_monitor_53.a & c4i.gui_event_monitor_53.at*

M1.1-2: *Controller*
- *M1.1-2.1:* *c4i.trans1_114.a & c4i.trans1_114.at*
- *M1.1-2.2:* *c4i.trans2_117.a & c4i.trans2_117.at*
- *M1.1-2.3:* *c4i.ctrller_120.a & c4i.ctrller_120.at*
- *M1.1-2.4:* *c4i.time_gen_126.a & c4i.time_gen_126.at*
- *M1.1-2.5:* *c4i.target_123.a & c4i.target_123.at*

d. *Program integration step*

The initial prototype programs can be generated by the program integration step from the above atomic module components. The top-level program component *P1.1* is a set of atomic program components that are categorized into four groups: *P1.1-1, P1.1-2, P1.1-3,* and *P1.1-4*. Files in *P1.1* are automatically integrated, scheduled, compiled, and

executed by CAPS. Atomic program components are presented as the following files with ada code:

P1.1-1: *Output*
- *P1.1-1.1:* *c4i.b_p_68.a*
- *P1.1-1.2:* *c4i.t_p_71.a*
- *P1.1-1.3:* *c4i.t_a_47.a*
- *P1.1-1.4:* *c4i.m_d_38.a*
- *P1.1-1.5:* *c4i.m_s_41.a*
- *P1.1-1.6:* *c4i.int_35.a*
- *P1.1-1.7:* *c4i.c_t_50.a*
- *P1.1-1.8:* *c4i.m_r_t_44.a*

P1.1-2: *Input*
- *P1.1-2.1:* *c4i.b_l_56.a*
- *P1.1-2.2:* *c4i.t_l_59.a*
- *P1.1-2.3:* *c4i.t_s_62.a*
- *P1.1-2.4:* *c4i.d_t_65.a*

P1.1-3: *GUI event monitor*
- *P1.1-3.1:* *c4i.gui_event_monitor_53.a*

P1.1-4: *Controller*
- *P1.1-4.1:* *c4i.trans1_114.a*
- *P1.1-4.2:* *c4i.trans2_117.a*
- *P1.1-4.3:* *c4i.ctrller_120.a*
- *P1.1-4.4:* *c4i.time_gen_126.a*
- *P1.1-4.5:* *c4i.target_123.a*

2. Second prototype of MD systems

a. Software prototype demo step

Criticisms to be addressed in the second prototype can be generated by the software prototype demo step from the above atomic program components. The top-level criticism component *C1.2* is a set of atomic criticism components that are categorized into six groups: *C1.2-1, C1.2-2, C1.2-3, C1.2-4, C1.2-5,* and *C1.2-6*. The criticism components are presented as the following text descriptions stored by individual files:

C1.2-1 *MD system must consider the 3d situation.*
- *C1.2-1.1* *The target position must consider the 3d situation.*
- *C1.2-1.2* *The missile intersection point must consider the 3d situation.*

C1.2-2 *The coordinate system of MD system must include the height.*
- *C1.2-2.1* *There is no height at target positions.*
- *C1.2-2.2* *There is no height at missile intersection points.*

C1.2-3 *It is hard to understand units of measurement, target and missile tracks, and degree system via the*

interface.

C1.2-3.1 *The interface must include unit of measurement.*

C1.2-3.2 *The target and missile tracks must be shown graphically.*

C1.2-3.3 *The degree system should provide options for the location of the origin (0 degree).*

C1.2-4 *This version of MD system is too simple.*

C1.2-4.1 *There is no height at missile base positions.*

C1.2-4.2 *The safety point must include 3d.*

C1.2-4.3 *The distance unit system should provide options for kilometers and nautical miles.*

C1.2-4.4 *MD system must consider multiple targets and missiles.*

C1.2-4.5 *MD system must provide the virtual reality image.*

C1.2-5 *MD system should consider the data of missile numbers, missile types, and missile speeds in each base.*

C1.2-5.1 *MD system must consider the number and type of missiles in a base to help the commander make the optimal decision.*

C1.2-5.2 *MD system must show data about available missile speed and type in each base, after detecting the target.*

C1.2-6 *This version of MD system can not protect the theater well.*

C1.2-6.1 *MD system must consider failure to hit the target object and launch another missile.*

C1.2-6.2 *MD system must provide the function to recognize the target object and predict the attack point of target object.*

C1.2-6.3 *MD system must suggest to the commander what kind of missile to launch from which base.*

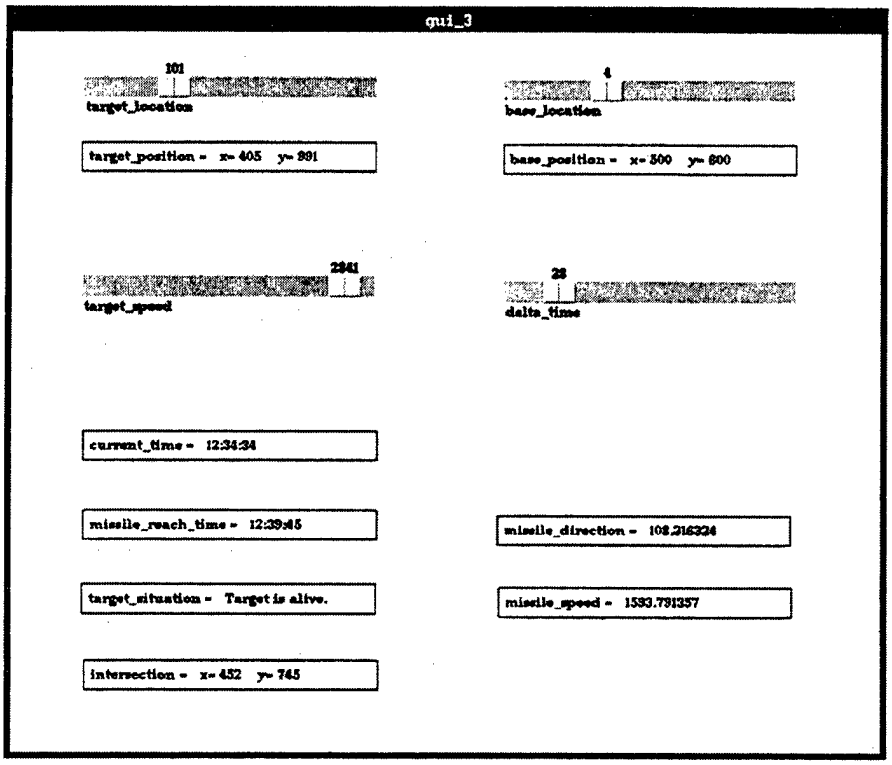


Figure 164: A demo panel of the initial prototype program

Figure 164 shows a demo panel of the initial prototype program. Information from external parts linked to the MD systems, namely communication links, platform sensors, navigation system, and weapons system, is simulated. Therefore, target location, target speed, base location, and command delay delta times are manually manipulated by stakeholders.

b. Issue analysis step

The second prototype issues can be generated by the issue analysis step from above atomic criticism components. The top-level issue component *I1.2* is a set of atomic issue components that are categorized into seven groups: *I1.2-1, I1.2-2, I1.2-3, I1.2-4, I1.2-5, I1.2-6,* and *I1.2-7.* Issue components are presented as the following text descriptions stored by individual files:

I1.2-1	*MD system must consider the 3d situation.*
I1.2-1.1	*The target position must consider the height.*
I1.2-1.2	*The missile intersection point must consider the height.*
I1.2-1.3	*The missile base position must consider the height.*
I1.2-2	*MD system must employ uniform units of measurement.*
I1.2-2.1	*The distance unit system must allow options for kilometers and nautical miles.*
I1.2-2.2	*The distance unit system must be kilometers or nautical miles.*
I1.2-2.3	*The degree system should provide options for the location of the origin (0 degree).*
I1.2-3	*MD system must have a friendly user interface.*
I1.2-3.1	*The user interface must provide degree marks.*
I1.2-3.2	*The target and missile tracks must be shown graphically.*
I1.2-4	*MD system must include detailed data on target objects and missiles.*
I1.2-4.1	*MD system must consider multiple targets and missiles.*
I1.2-4.2	*MD system must consider the number and type of missiles in a base to help the commander make the optimal decision.*
I1.2-4.3	*MD system must show data about the speed and type of available missiles in each base, after detecting the target.*
I1.2-5	*MD system must consider failure to hit target object.*
I1.2-5.1	*MD system must consider failure to hit the target object and launch another missile.*
I1.2-5.2	*MD system must have the capacity to know whether the missile hit the target object or not.*
I1.2-5.3	*MD system must consider how many missiles are left in each base.*
I1.2-6	*MD system must consider missile launch automation and optimization.*
I1.2-6.1	*MD system must provide the function to recognize the target object and predict the attack point of the target object.*
I1.2-6.2	*MD system must suggest to the commander what kind of missile to launch from which base.*
I1.2-6.3	*MD system must provide the function to launch automatically missiles to attack target objects according to intelligence from sensors.*
I1.2-6.4	*MD system must provide the function to simulate the missile defense process and consider the missile launch optimization.*
I1.2-7	*It is hard to design a virtual reality image in CAPS.*
I1.2-7.1	*TAE does not provide a virtual reality design environment.*
I1.2-7.2	*Virtual reality design can be considered in different software development platform.*

c. *Requirement analysis step*

The second prototype requirements can be generated by the requirement analysis step from the above issues. Some issues are able to generalize or specialize new requirements but some issues are not included in new requirements after validating the requirements. Validating requirements is the process through which the customers' real needs are checked against the formalized requirements to make sure that the formalized requirements accurately meet those needs. Top-level requirement component *R1.2* is a set of atomic requirement components that are categorized into two groups: *R1.2-1*, and *R1.2-2*. Requirement components are presented as the following text descriptions stored by individual files:

R1.2-1: *The MD system must employ uniform units of measurement.*
- *R1.2-1.1:* *The distance unit system must be nautical miles.*
- *R1.2-1.2:* *The degree system must locate 0 degrees (the origin) at the north, 90 degrees at the west, 180 degrees at the south, and 270 degrees at the east.*

R1.2-2: *The MD system must provide dynamic output graphic interface for users to monitor the base position, target position, missile direction, missile speed, target and missile intersection point.*
- *R1.2-2.1:* *The dynamic output graphic interface must provide the function of locating the base positions.*
- *R1.2-2.2:* *The dynamic output graphic interface must provide the function of monitoring the target position.*
- *R1.2-2.3:* *The dynamic output graphic interface must provide the function of monitoring the missile direction.*
- *R1.2-2.4:* *The dynamic output graphic interface must provide the function of monitoring the missile speed.*
- *R1.2-2.5:* *The dynamic output graphic interface must provide the function of monitoring the target and missile intersection point.*
- *R1.2-2.6:* *The dynamic output graphic interface must provide two movers: the target object and the missile object, that can be moved in the 2d coordinate system.*
- *R1.2-2.7:* *The dynamic output graphic interface must provide two rings: the target object approaching ring and the safety ring (also called the target and missile intersection ring).*
- *R1.2-2.8:* *The dynamic output graphic interface must provide grid coordinates, distance marks and degree marks.*

d. *Specification design step*

The second prototype specifications can be generated by the specification design step from the above atomic requirement components. The top-level specification component *S1.2* is a set of atomic specification components that are categorized into two groups: *S1.2-1* and *S1.2-2*. The specification components are presented as the following specification files with IMPLEMENTATION and SPECIFICATION PSDL code:

S1.2-1: *c4i.gui_3.imp.psdl & c4i.gui_3.spec.psdl*
- *S1.2-1.1:* *c4i.b_p_68.imp.psdl & c4i.b_p_68.spec.psdl*
- *S1.2-1.2:* *c4i.t_p_71.imp.psdl & c4i.t_p_71.spec.psdl*
- *S1.2-1.3:* *c4i.t_a_47.imp.psdl & c4i.t_a_47.spec.psdl*
- *S1.2-1.4:* *c4i.m_d_38.imp.psdl & c4i.m_d_38.spec.psdl*
- *S1.2-1.5:* *c4i.m_s_41.imp.psdl & c4i.m_s_41.spec.psdl*
- *S1.2-1.6:* *c4i.int_35.imp.psdl & c4i.int_35.spec.psdl*
- *S1.2-1.7:* *c4i.c_t_50.imp.psdl & c4i.c_t_50.spec.psdl*
- *S1.2-1.8:* *c4i.m_r_t_44.imp.psdl & c4i.m_r_t_44.spec.psdl*
- *S1.2-1.9:* *c4i.b_l_56.imp.psdl & c4i.b_l_56.spec.psdl*
- *S1.2-1.10:* *c4i.t_l_59.imp.psdl & c4i.t_l_59.spec.psdl*
- *S1.2-1.11:* *c4i.t_s_62.imp.psdl & c4i.t_s_62.spec.psdl*
- *S1.2-1.12:* *c4i.d_t_65.imp.psdl & c4i.d_t_65.spec.psdl*
- *S1.2-1.13:* *c4i.gui_event_monitor_53.imp.psdl & c4i.gui_event_monitor_53.spec.psdl*
- *S1.2-1.14:* *c4i.merge_233.imp.psdl & c4i.merge_233.spec.psdl*
- *S1.2-1.15:* *c4i.t_m_p_236.imp.psdl & c4i.t_m_p_236.spec.psdl*
- *S1.2-1.16:* *c4i.m_p_239.imp.psdl & c4i.m_p_239.spec.psdl*

S1.2-2: *c4i.ctrl_6.imp.psdl & c4i.ctrl_6.spec.psdl*
- *S1.2-2.1:* *c4i.trans1_114.imp.psdl & c4i.trans1_114.spec.psdl*
- *S1.2-2.2:* *c4i.trans2_117.imp.psdl & c4i.trans2_117.spec.psdl*
- *S1.2-2.3:* *c4i.ctrller_120.imp.psdl & c4i.ctrller_120.spec.psdl*
- *S1.2-2.4:* *c4i.time_gen_126.imp.psdl & c4i.time_gen_126.spec.psdl*
- *S1.2-2.5:* *c4i.target_123.imp.psdl & c4i.target_123.spec.psdl*
- *S1.2-2.6:* *c4i.movers_250.imp.psdl & c4i.movers_250.spec.psdl*

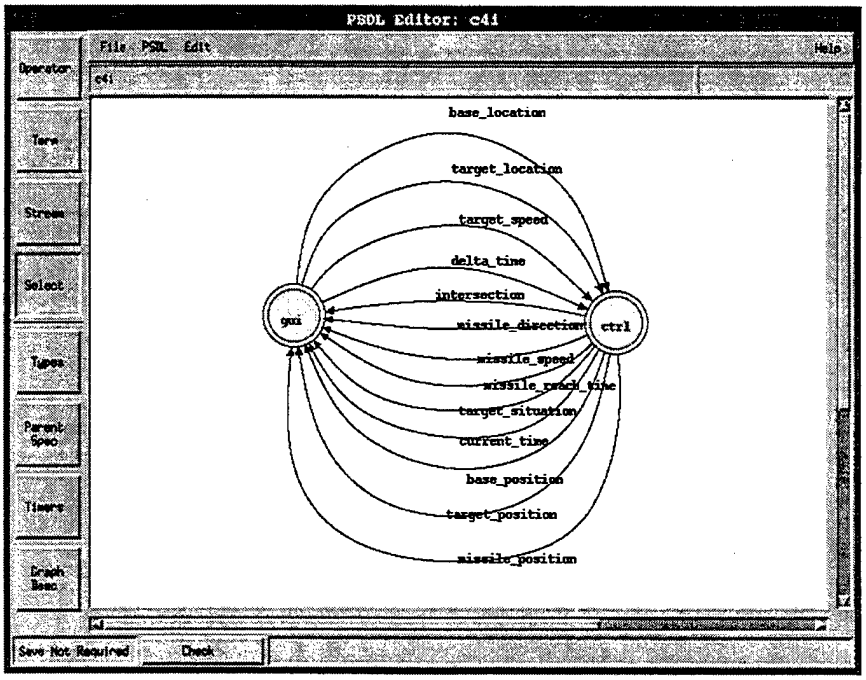


Figure 165: A top-level data flow diagram – *c4i (modified)*

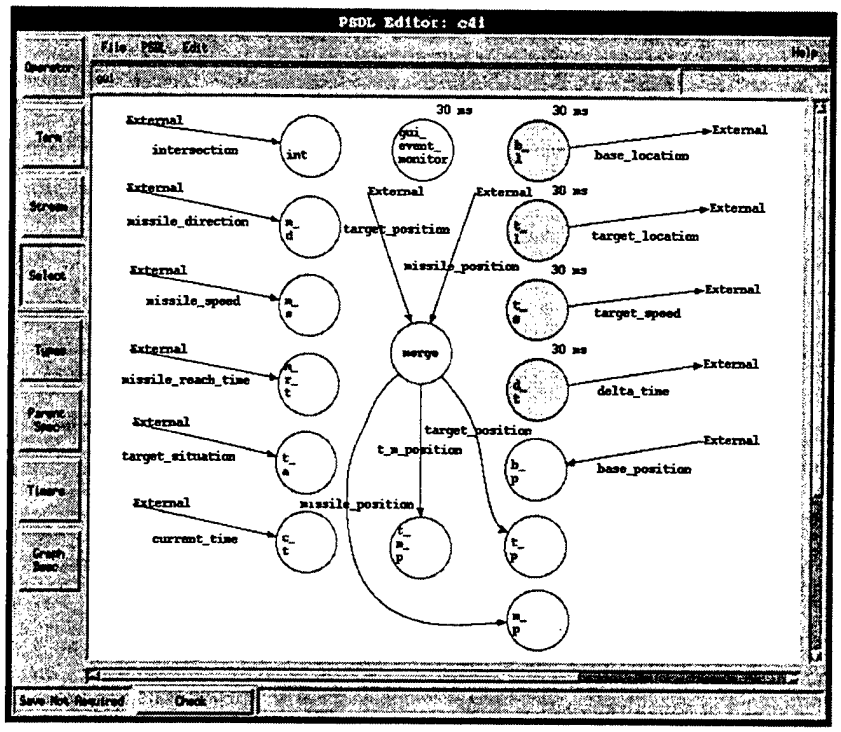


Figure 166: A decomposed data flow diagram – *gui (modified)*

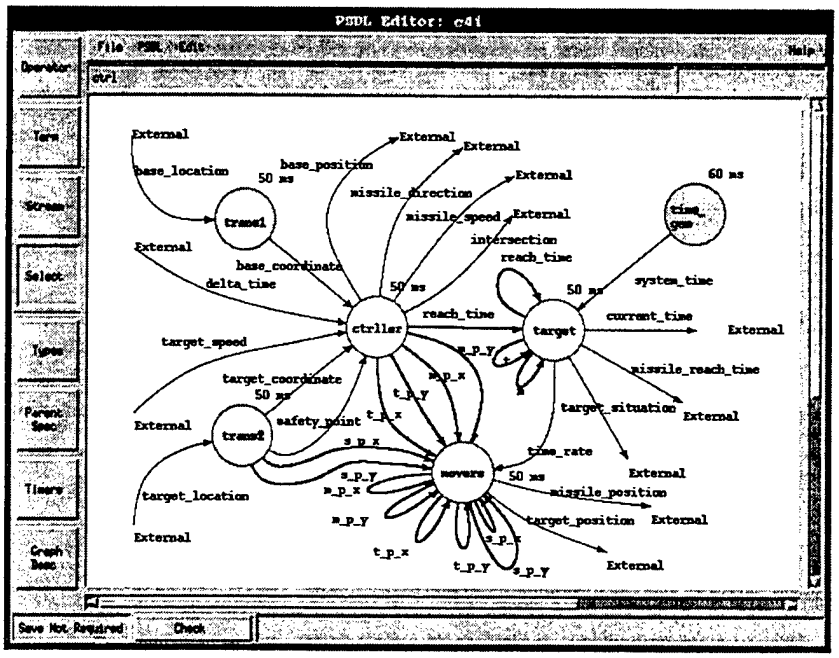


Figure 167: A decomposed data flow diagram – *ctrl (modified)*

In Figure 165, one new data stream missile-position modifies the top-level data flow diagram *c4i*. Furthermore, Figure 166 shows a decomposed data flow diagram *gui,* modified by 3 new operators (*merge, t_n_p*, and *m_p*) and their related data streams, and Figure 167 shows a decomposed data flow diagram *ctrl* modified by one new operator (movers) and its related data streams.

e. Module implementation step

The second prototype modules can be generated by the module implementation step from the above atomic specification components. The top-level module component *M1.2* is a set of atomic module components that are categorized into two groups: *M1.2-1* and *M1.2-2*. The atomic module components are presented as the following files with ada code:

M1.2-1: *Interface*
- *M1.2-1.1:* *c4i.b_p_68.a*
- *M1.2-1.2:* *c4i.t_p_71.a*
- *M1.2-1.3:* *c4i.t_a_47.a*
- *M1.2-1.4:* *c4i.m_d_38.a*
- *M1.2-1.5:* *c4i.m_s_41.a*
- *M1.2-1.6:* *c4i.int_35.a*
- *M1.2-1.7:* *c4i.c_t_50.a*
- *M1.2-1.8:* *c4i.m_r_t_44.a*
- *M1.2-1.9:* *c4i.b_l_56.a*
- *M1.2-1.10:* *c4i.t_l_59.a*
- *M1.2-1.11:* *c4i.t_s_62.a*
- *M1.2-1.12:* *c4i.d_t_65.a*
- *M1.2-1.13:* *c4i.gui_event_monitor_53.a & c4i.gui_event_monitor_53.at*
- *M1.2-1.14:* *c4i.merge_233.a & c4i.merge_233.at*
- *M1.2-1.15:* *c4i.t_m_p_236.a*
- *M1.2-1.16:* *c4i.m_p_239.a*

M1.2-2: *Controller*
- *M1.2-2.1:* *c4i.trans1_114.a & c4i.trans1_114.at*
- *M1.2-2.2:* *c4i.trans2_117.a & c4i.trans2_117.at*
- *M1.2-2.3:* *c4i.ctrller_120.a & c4i.ctrller_120.at*
- *M1.2-2.4:* *c4i.time_gen_126.a & c4i.time_gen_126.at*
- *M1.2-2.5:* *c4i.target_123.a & c4i.target_123.at*
- *M1.2-2.6:* *c4i.merge_233.a & c4i.merge_233.at*

f. Program integration step

The second prototype programs can be generated by the program integration step from the above atomic module components. The top-level program component *P1.2* is a set of atomic program components that are categorized into four groups: *P1.2-1, P1.2-2, P1.2-3*, and *P1.2-4*. The atomic program components are presented as the following files with ada code:

P1.2-1: *Output*
- *P1.2-1.1:* *c4i.b_p_68.a*
- *P1.2-1.2:* *c4i.t_p_71.a*
- *P1.2-1.3:* *c4i.t_a_47.a*
- *P1.2-1.4:* *c4i.m_d_38.a*
- *P1.2-1.5:* *c4i.m_s_41.a*
- *P1.2-1.6:* *c4i.int_35.a*
- *P1.2-1.7:* *c4i.c_t_50.a*
- *P1.2-1.8:* *c4i.m_r_t_44.a*
- *P1.2-1.9:* *c4i.merge_233.a*
- *P1.2-1.10:* *c4i.t_m_p_236.a*
- *P1.2-1.11:* *c4i.m_p_239.a*

P1.2-2: *Input*
- *P1.2-2.1:* *c4i.b_l_56.a*
- *P1.2-2.2:* *c4i.t_l_59.a*
- *P1.2-2.3:* *c4i.t_s_62.a*
- *P1.2-2.4:* *c4i.d_t_65.a*

P1.2-3: *GUI event monitor*
- *P1.2-3.1:* *c4i.gui_event_monitor_53.a*

P1.2-4: *Controller*
- *P1.2-4.1:* *c4i.trans1_114.a*
- *P1.2-4.2:* *c4i.trans2_117.a*
- *P1.2-4.3:* *c4i.ctrller_120.a*
- *P1.2-4.4:* *c4i.time_gen_126.a*
- *P1.2-4.5:* *c4i.target_123.a*
- *P1.2-4.6:* *c4i.merge_233.a*

Figure 168 shows a demo panel of the second prototype program. In addition to new output items and unit marks in the panel, there are two movers, a target object approaching ring, a safety ring, and ten missile base positions in the two dimension coordinate system. It has been improved from the initial prototype program. Further iteration may be required until the prototype program fully meets the users' requirements (illustrated in Figure 16).

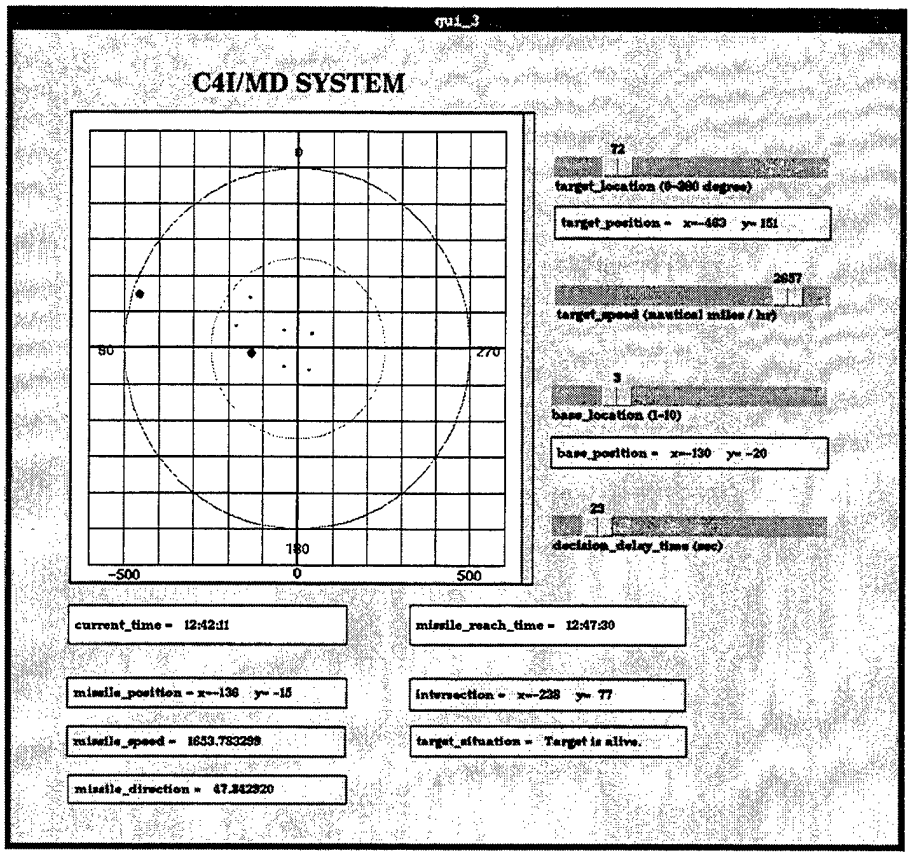


Figure 168: A demo panel of the second prototype program

Different evolution processes of C4I systems decompose evolution activities in different ways [LUQI97]. Our formal model emphasizes a domain-specific software development architecture. There are some difficulties in formalizing domains that are subject to frequent requirements changes. The RH model with prototyping has resolved some of the problems in evolution processes of C4I systems, such as evolution path traceability, object management, process description, and requirements analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. EVALUATION AND VALIDATION

A. EVALUATION

1. By a requirements management tool: QSS DOORS

We compare CASES with a similar tool called QSS DOORS to evaluate the performance of CASES. QSS DOORS is a software system development tool currently selected by the U.S. Treasury Inspector General for Tax Administration Corporate Management Information System Project [DATT99] [ROTH99].

a. QSS DOORS

In the software system development environment, a life cycle approach to software systems development for manageability and repeatability is needed for system engineering. In [DATT99], Datta pointed out that a Requirements Management Tool (RMT), such as QSS DOORS, can manage all the phases of the life cycle. The RMT not only provides configuration management and traceability that maintains links between documents in each phase, but also offers interfaces to an object-oriented development tool and a software testing tool. The life cycle phases include requirements gathering, analysis, modeling or design, coding and testing as shown in Figure 169.

To navigate through these phases, many integrated tools have been selected to assist in the life cycle. Like most software system development tools, none of the RMTs evaluated were optimally suited to all phases of the development process. Customization, by creating special RMT programs, was needed to meet the end users' work styles and capture legacy data [ROTH99]. The following tools were evaluated for appropriate phases of the life cycle:

- QSS DOORS, Rational Requisite Pro, RMT for requirement management and traceability.
- Cayenne Object Team, Rational Rose and Paradigm Plus for object modeling and

design.

- ERWIN and Cayenne Data Team for data modeling and database design.
- Mercury WinRunner, Segue and Rational SQA suite for testing.
- PVCS and Rational ClearCase for configuration management.

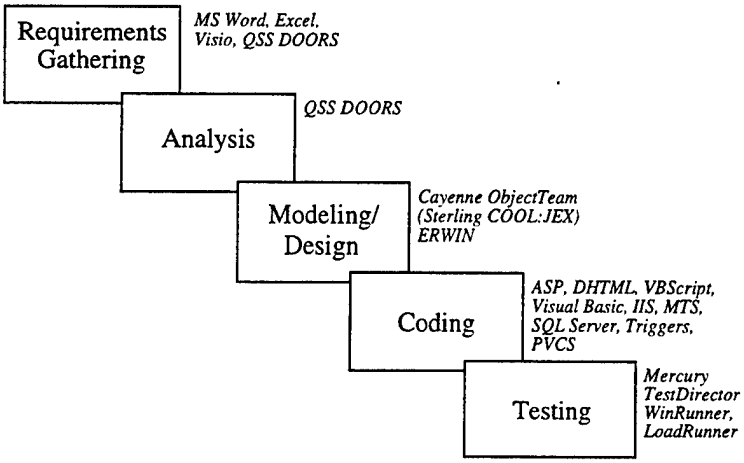


Figure 169: Life cycle phases and tools

In [ROTH99], Rothstein noted that the U. S. Treasury software development project selected QSS DOORS to capture, manage, and maintain user requirements critical to a software development project because of the following criteria it offered:

- a way of importing existing structured MS Word, outline-text, spreadsheet, and graphics documents,
- a method of maintaining and managing changes to the documents with inputs from the requirements group, the users, and the developers, and
- a method of bidirectionally linking to both an object-oriented computer-aided software engineering (CASE) modeling tool, and a testing system.

Although QSS DOORS met the selection criteria, it had some limitations in user availability, user familiarity, and ease-of-use.

b. CASES

At least fifteen features of CASES are listed. These features of CASES and comparisons to QSS DOORS are as follows:

(1) Design purpose

- CASES is designed for general purposes, not only for software evolution but also for any evolution of system engineering. QSS DOORS is designed only for system engineering life cycle management with traceability.
- CASES is designed for assisting large and complex software system evolution. QSS DOORS is designed to provide a blueprint for system engineering manageability and success for increasingly complex systems development.
- CASES can be used not only in a real-time embedded system but also in a management information system (MIS). QSS DOORS is developed for a Corporate Management Information System (CMIS) project that employs RMT.

(2) Software evolution process

- CASES can be used in different phases of software evolution processes, including software prototyping evolution processes and software product generation processes. QSS DOORS is limited to the traditional waterfall Software Development Life Cycle (SDLC) model.
- CASES provides functions to adjust the software evolution processes by stakeholders to support process improvement. QSS DOORS is limited to the rigorous SDLC model, which includes requirement gathering, analysis, modeling or design, coding and testing.
- CASES provides functions to propose, approve, schedule, assign, decompose, complete, and abandon jobs. QSS DOORS is only a requirements management tool which is suitable for managing and configuring all requirements documents.
- CASES allows different stakeholders to manage and control software evolution processes. QSS DOORS is designed for the people who are involved in the SDLC.

(3) Automation

- CASES provides functions to automate step and component versions by dependency rules. Each process in QSS DOORS was numbered using a scheme of 1.0, 1.1, 1.1.1, etc., for parent and children processes.
- CASES' automation part is created by dependency rules. There are no dependency rules in QSS DOORS.
- CASES provides functions to change default component versions based on a set of dependency rules for stakeholder's needs. QSS DOORS only provides a

numbering scheme for parent and children processes.

- CASES provides functions to specify dependencies among software evolution objects and to build relationships by these dependencies. Data elements in QSS DOORS are linked to the parent requirements using the requirements number and storing them with each data element object.

(4) Connection

- CASES can completely capture, manage, and maintain user requirements critical to a software development project by text file, data file, URL, and CAPS file links. QSS DOORS uses the easy way of importing legacy-requirements data already captured in MS Word, MS Excel, and Visio.

(5) Traceability

- CASES provides functions to trace software evolution history horizontally and to refine the software evolution components vertically. QSS DOORS provides a unique and traceable methodology to link requirements with design and keep it synchronized. The traceability functions of QSS DOORS are less than those of CASES.

(6) Job scheduling and assignment

- CASES provides functions to manage security levels, required skills and levels. There is no function to manage data associated with stakeholders in QSS DOORS.
- CASES provides scheduling heuristic rules to schedule and assign jobs. There is no function to schedule and assign jobs to stakeholders in QSS DOORS.

2. By job scheduling and assignment algorithms

a. Scheduling algorithms of John Evans

John Evans designed Project Scheduling Tool (PST) based on scheduling algorithms of ECS which was developed by Salah Badr [BADR93]. Salah's thesis delved into a broad array of issues related to managing large projects and their concomitant complexity. John Evans improved Salah's ECS and some essential issues of scheduling algorithms have been reviewed and outlined [EVAN97] as follows:

- The problem of optimal scheduling tasks for both the preemptive and nonpreemptive cases is NP-complete [ULLM75]. Scheduling nonpreemptive tasks with arbitrary ready times is also NP-complete in both multiprocessor and uniprocessor systems [RAMA89b]. For dynamic systems with more than one task,

and mutual exclusion constrains between tasks, Mok and Dertouzos [MOK78] showed that an optimal scheduling algorithm does not exist.

- Shiah, et al. [RAMA89a] came up with an heuristic scheduling algorithm that ran in order kN time. Salah Badr extended the algorithm to consider arbitrary precedence constraints between pairs of tasks. His scheduler forms the basis of the current Evolution Control System scheduling algorithm.
- The scheduling algorithm, as implemented by Salah Badr [BADR93], was recursive. It consumed order N^2 memory for a set of N tasks. It attempted to improve performance by limiting backtracking, but was still at least order N^2 in time. It was based on an algorithm described in the paper by Ramamritham [RAMA89b]. The requirement for order N^2 space limited the size of the problem domain. This thesis describes the algorithm and steps taken to make the algorithm run using only order N space by Salah Badr. It is based on the "myopic" algorithm [RAMA89a] and a radical restructuring of the data structures in the Ada code.

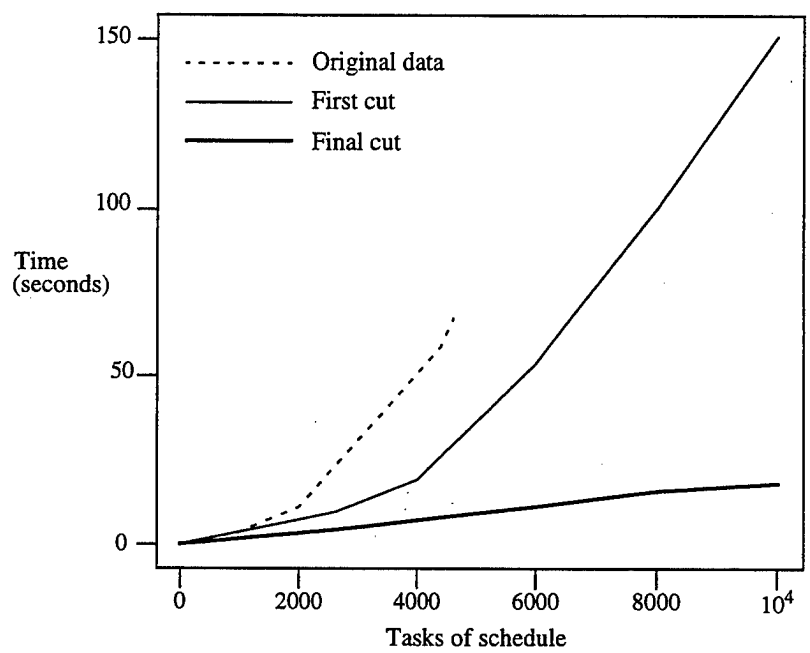


Figure 170: Plot of scheduler run-time vs. number of tasks to schedule

Once the space problem was corrected, it became evident that the routine was also $O(N^2)$ in time. But this was easily rectified by using the "myopic" algorithm. John Evans applied this algorithm to analyze the time complexity in Figure 170, which shows

the speed-up in processing speed vs. number of tasks to be scheduled for different versions of the code to a degree. The original data came with the original code. After the N^2 space problem was resolved and before the myopic version of the code was added (first cut), we see that the code still runs in order N^2 time.

The final cut shows the run-time for the final version of the code. The original data collected goes up to only 4,600 tasks because the storage required was $O(N^2)$ in the number of tasks to be scheduled. A number larger than 4,600 tasks would cause the program to raise a storage-error exception.

The "myopic" algorithm can be described as follows:

- The goal of the scheduling algorithm is to determine if a schedule for executing the tasks that satisfies the timing, precedence, and resource constraints exists, and to calculate such a schedule if it exists. A schedule that meets these constraints is termed *feasible*. However, it is not guaranteed to be *optimal*.
- Scheduling a set of tasks to find a full feasible schedule is actually a search problem. The search space is a tree. The scheduling algorithm starts at the root of a tree using a predetermined heuristic and selects a candidate task to schedule. If the remaining tasks can be added to the schedule in the order given by the heuristic without violating the constraints, then the partial schedule is termed *strongly-feasible*, and the task is added to the search tree as a vertex node, and the process is repeated, recursively, until a full, feasible schedule is found. If instead, after the candidate task is selected and any one of the remaining tasks added to the schedule violates the constraints, the candidate task is rejected and the next eligible, candidate task (ordered by the ranking function $H(T)$) is selected. The search process continues until all the tasks are scheduled, or no feasible schedule is found.
- Instead of using all of the remaining tasks to determine if a partial schedule is strong-feasible, Ramamritham [RAMA89a] limited the candidate tasks to check to some number k. So, instead of checking $N, N-1, \ldots, 1$ remaining tasks, or $N(N-1)/2$ total tasks, they limited the search to k or at most kN tasks to check. (This is where the term "myopic" comes in. Instead of looking at all the remaining tasks, we "myopically" examine only the next k tasks.)

b. Scheduling and assignment algorithm of CASES

We apply the heuristic scheduling algorithms of John Evans to build dependency policy rules for an appropriate job scheduling and assignment environment.

The features of job scheduling and assignment algorithm in CASES and comparisons with Salah's ECS and John's PSD are addressed as follows:

- CASES integrates the heuristic scheduling algorithms and design job scheduling functions based on six heuristics described in [EVANS]. Salah's ECS only used the Min_D + Min_S heuristic which is superior in all cases. John's PSD improved the space and time complexity of job scheduling but didn't integrate the heuristic scheduling algorithms well.
- In order to resolve dynamic rescheduling problems, the two job slots of stakeholders have been designed. CASES assigns no more than two jobs at once to a stakeholder: a current on-hand job and a queuing job. If a stakeholder who has a current on-hand job and a queuing job completes the current on-hand job, CASES will take the queuing job as the current on-hand job and schedule the stakeholder a new job to substitute the queuing job by one of the dependency policy rules chosen by project organizers. The special design of two job slots takes into account the dynamic rescheduling cases and provides a chance for stakeholders to change the heuristic scheduling algorithm specified. There is no choice to change heuristics for rescheduling jobs during the scheduling period in Salah's ECS and John's PST when stakeholders hit dynamic rescheduling problems.
- Project organizers are involved in the scheduling process with CASES and arbitrarily choose one of heuristic scheduling algorithms in the job scheduling state to schedule a job for stakeholders. Salah's ECS schedules all jobs by one heuristic algorithm: Min_D + Min_S. John's PST schedules all jobs by a specified heuristic scheduling algorithm per time.
- The job scheduling and assignment algorithms of CASES provides a security level filter and a skill and level sorter to decide appropriate stakeholders to carry a job. The security filter can filter the stakeholders whose security level is lower than the required job security level. The skill and level sorter can list the stakeholders by the order based on the matching number of skills and levels with a scheduling job. There is no security level consideration in Salah's ECS and John's PST. Salah's ECS considered capacities of stakeholders in three broad categories: low, medium, and high. John's PST considered the capacities by special abilities with three levels: low, medium, and high.
- The job scheduling and assignment algorithm of CASES may assign a job to a stakeholder who is not the first priority in a candidate list if the two job slots of the first priority candidate have already been fulfilled. CASES assigns a feasible stakeholder for a major job and the first priority stakeholder for a minor job. The major job stakeholders should be carried out by themselves, but the minor job may not be carried out alone. If the stakeholder has a minor job, he needs to provide related expertise to the minor job and help stakeholders who own this job. Sahla's ECS and John's PST only assign a job to one stakeholder.

3. By inferred dependencies

We compare CASES with the ECS created by Salah Badr based on inferred dependencies to evaluate the performance of CASES. Even though the lightweight inference engines with dependency rules of CASES and ECS are designed inside algorithms and programs, CASES has some distinct advantages over ECS:

- CASES is built by RH model and a dependency-computing model. ECS is built by graph model [LUQI90]. We extend a graph model and a hypergraph model [LUQI97] into an RH model to enhance the object control, management, formation, refinement, traceability and assignment functions. Salah modified and extended 16 rules from the graph model of which we modified and extended 57 dependency rules from this model.
- CASES uses dependency rules to automate object version control on the whole software evolution processes including the software prototype evolution process and the software product generation process. ECS uses dependency rules to automate version control only on the program specification and implementation processes.
- CASES uses scheduling policy rules to help project managers change job scheduling policies. ECS has no scheduling policy rules.
- CASES can automatically form an atomic SPIDER by dependency rules, but ECS forms a step by users.
- In order to classify many different dependencies in the same type, we construct the dependencies of CASES by class structure. Due to few dependencies in ECS, there is no class structure in ECS.

B. VALIDATION

We have successfully validated CASES by CASES users and C4I/MD systems.

1. By CASES users

CASES is manipulated by the CASES user who is also one of the stakeholders. Stakeholders including project organizers, project evaluators, system analysts, and system designers facilitate CASES for different purposes. We have validated CASES through different stakeholders who manipulate CASES under manual directions in different software evolution activities.

a. Project organizers

The project organizers take the responsibility of organizing a software project. Project organizers can achieve the following activities based on CASES manual directions:

(1) Create a project and define software evolution object types under the specific software project.

Step 1. Create a project by using the menu item *Create Project* under the menu bar *Project.*
Step 2. Define software evolution step types when the frame *Project Schema: Step Type* is selected.
Step 3. Define software evolution component types when the frame *Project Schema: Component Type* is selected.

(2) Modify definitions of software evolution object types under a specific software project.

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Modify definitions of software evolution step types when the frame *Project Schema: Step Type* is selected.
Step 3. Modify definitions of software evolution component types when the frame *Project Schema: Component Type* is selected.

(3) Create software evolution processes under a specific software project.

Step 1. Finish defining software evolution object types by the steps of activity (1) or activity (2).
Step 2. Create software evolution processes when the frame *Project Schema: Evolution Process* is selected.

(4) Modify software evolution processes under a specific software project.

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Modify software evolution processes when the frame *Project Schema: Evolution Process* is selected.

(5) Define or modify dependencies among software evolution objects.

Step 1. Finish creating software evolution processes by the steps of activity (1) or activity (2).
Step 2. Define dependencies among software evolution objects when the frame *Project Schema: Dependency* is selected.

(6) Create a new step version under a specific software project.

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Select the menu item *Create Step Version, Evolution History Splitting,* or *Evolution History Merging* under the menu bar *Automated Version Control.*
Step 3. Create a new step version by giving data in the *Automated Version Control-New Step Version, Automated Version Control-Evolution History Splitting,* or *Automated Version Control-Evolution History Merging* panel.

(7) Manage the required skills and levels, and security level of a stakeholder.

Step 1. Select the menu item *Personnel Data* under the menu bar *Tools.*
Step 2. Specify or modify the required skills and levels, and security level of a stakeholder in the *Personnel Data* panel.

(8) Organize an atomic SPIDER as a job and propose the job with scheduling, skill, and security constraints to a project evaluation team.

Step 1. Finish defining software evolution objects by the steps of activity (1) or activity (2).
Step 2. Finish creating software evolution processes by the steps of activity (3) or activity (4).
Step 3. Open current software evolution step by using the menu item *Open Step* under the menu bar *Automated Version Control.*
Step 4. Select the menu item *Edit* or *Decompose* under the menu bar *SPIDER.*
Step 5. Organize an atomic SPIDER as a job.
Step 6. Select the menu item *Step Content* under the menu bar *SPIDER.*
Step 7. Propose the job (the atomic SPIDER) and add scheduling, skill, and security constraints in the *SPIDER-Step Content* panel.
Step 8. Select the data item *Proposed* in the *Status* combo box of the *SPIDER-Step Content* panel.
Step 9. Select the menu item *Component Content* under the menu bar *SPIDER.*

Step 10. Add the related component content links in the *Connection Link Editor* frame.

(9) Schedule and assign a job to a project analysis team or a project design team.

Step 1. Select the menu item *Scheduling* under the menu bar *Job Schedule.*
Step 2. Select a job scheduling policy.
Step 3. Select the menu item *Assignment* under the menu bar *Job Schedule.*
Step 4. Assign a job to a project analyst or a project designer.

b. Project evaluators

The project evaluators take the responsibility of evaluating a software project. Project evaluators can achieve the following activities based on CASES manual directions:

(1) Evaluate and modify software evolution processes under a specific software project.

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Modify or add software evolution processes when the frame *Project Schema: Evolution Process* is selected.

(2) Evaluate and upgrade security levels, required skills and levels for stakeholders.

Step 1. Select the menu item *Personnel Data* under the menu bar *Tools.*
Step 2. Upgrade security levels, required skills and levels of a stakeholder in the frame *Personnel Data Panel.*

(3) Evaluate the formation of an atomic SPIDER with the scheduling, skill, and security constraints proposed by project organizers or system designers.

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Open current software evolution step by using the menu item *Open Step* under the menu bar *Automated Version Control.*

Step 3. Select the menu item *Edit* or *Decompose* under the menu bar *SPIDER.*
Step 4. Select the menu item *SPIDER-Step Content* under the menu bar *SPIDER.*
Step 5. Browse and evaluate atomic SPIDERs whose status is *Proposed* or *Decomposed*
Step 6. Evaluate the job (the atomic SPIDER) whose status is *Proposed* or *Decomposed* with scheduling, skill, and security constraints in the *SPIDER-Step Content.*
Step 7. Select a status *Approved, Assigned, Completed* or *Abandoned* in the *Status* combo box of the *SPIDER-Step Content* panel.

(4) Make the risk assessment and the failure impact evaluation for a job.

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Open current software evolution step by using the menu item *Open Step* under the menu bar *Automated Version Control.*
Step 3. Select the menu item *Edit* under the menu bar *SPIDER.*
Step 4. Browse and evaluate atomic SPIDERs.
Step 5. Select the menu item *Step Content* under the menu bar *SPIDER.*
Step 6. Evaluate the job (the atomic SPIDER) with scheduling, skill, and security constraints in the *SPIDER-Step Content* panel.
Step 7. Make the risk assessment and the failure impact evaluation for the job and enter the evaluation number into the data item *Evaluation.*

c. System analysts and system designers

The system analysts and system designers are responsible for completing the job (the atomic SPIDER) assigned by CASES. The following CASES manual directions can help system analysts and system designers carry out a job:

Step 1. Open a project by using the menu item *Open Project* under the menu bar *Project.*
Step 2. Open the current software evolution step by using the menu item *Open Step* under the menu bar *Automated Version Control.*
Step 3. Select the menu item *Trace* under the menu bar *SPIDER* to understand the step (job) content and component content of the assigned atomic SPIDER, and to trace the software evolution history in the *SPIDER-Trace* panel.
Step 4. Enter or modify component content links of the assigned SPIDERs in the *SPIDER-Component Content* and *Component Content Editor* panels by selecting menu item *Component Content* in the menu bar *SPIDER.*
Step 5. Select the menu item *Text Editor, MS Word, MS Excel, Netscape,* or *CAPS*

under the menu bar *Tools* and carry out the assigned job with the specified tool.

Step 6. Select a data item *Decomposed, Approved, Completed* or *Abandoned* in the *Status* combo box of the *SPIDER-Step Content* panel to change the assigned job status.

2. By C4I/MD systems

CASES has been validated by two software evolution generations of C4I/MD systems in the Chapter VII. The project *c4i* involves at least four different CASES users: project organizers, project evaluators, system analysts, and system designers to organize the project and to propose, approve, schedule, assign, decompose, complete, and abandon the jobs of the project.

a. Organize a project

At the beginning, the project organizers have to organize the project *c4i* and manipulate CASES as follows:

Step 1. Create the project *c4i* by selecting the menu item *Create Project* under the menu bar *Project.*

Step 2. Open the *Project Schema* frame to define the step types, component types, software evolution processes, and dependencies.

Step 3. Define the following step types in the *Project Schema: Step Type* frame: *Software Prototype Demo Step (s-C), Issue Analysis Step (s-I), Requirement Analysis Step (s-R), Specification Design Step (s-S), Module Implementation Step (s-M), Program Integration Step (s-P), Software Product Demo Step (s-O),* and *Software Product Implementation Step (s-Pd).*

Step 4. Define the following component types in the *Project Schema: Component Type* frame: *Criticisms (C), Issues (I), Requirements (R), Specifications (S), Modules (M), Software Prototype Programs (P), Optimizations (O), Software Product Programs (Pd), Test Scenarios (T),* and *Virtual Teams (VT).*

Step 5. Define the following software evolution processes in the *Project Schema: Evolution Process* frame: *Software Prototype Evolution Process (s-C, s-I, s-R, s-S, s-M, s-P, s-Pd)* and *Software Product Generation Process (s-O, s-Pd).*

Step 6. Define the following dependencies in the *Project Schema: Dependency* frame: $C \leftarrow s\text{-}C(C, P, T, VT)$, $I \leftarrow s\text{-}I(I, C, VT)$, $R \leftarrow s\text{-}R(R, I, VT)$, $S \leftarrow s\text{-}S(S, R, VT)$, $M \leftarrow s\text{-}M(M, S, VT)$, $P \leftarrow s\text{-}P(P, M, VT)$, $O \leftarrow s\text{-}O(O, Pd, T, VT)$, and $Pd \leftarrow s\text{-}Pd(Pd, VT)$.

b. Propose a job

The project organizers propose (or abandon) SPIDERs by using CASES as the following manipulations:

Step 1. Create a new step version *s-R1.1* in the *Automated Version Control-Create Step Version* panel by selecting the menu item *Create Step Version* in the menu bar *Automated Version Control.*

Step 2. Open the step version *s-R1.1* in the *Automated Version Control-Open Step Version* panel by selecting the menu item *Open Step Version* in the menu bar *Automated Version Control.*

Step 3. Edit SPIDERs in the *SPIDER-Edit* panel by selecting the menu item *Edit* in the menu bar *SPIDER.*

Step 4. Decompose or edit SPIDERs in the *SPIDER-Decompose* panel by selecting the menu item *Decompose* in the menu bar *SPIDER.*

Step 5. Enter component content links of the SPIDERs in the *SPIDER-Component Content* and *Component Content Editor* panels by selecting the menu item *Component Content* in the menu bar *SPIDER.*

Step 6. Enter step content data of the SPIDERs in the *SPIDER-Step Content* panel by selecting the menu item *Step Content* in the menu bar *SPIDER.*

Step 7. When the SPIDERs are proposed, select their status *Proposed* in the status combo box of the *SPIDER-Step Content* panel.

Step 8. When the SPIDERs are abandoned, select their status *Abandoned* in the status combo box of the *SPIDER-Step Content* panel.

c. Approve a job

The project evaluators evaluate and approve (or abandon) the SPIDERs by using CASES as the following manipulations:

Step 1. Open the project *c4i* by selecting menu item *Open Project* in the menu bar *Project.*

Step 2. Open the step version *s-R1.1* in the *Automated Version Control-Open Step Version* panel by selecting the menu item *Open Step Version* in the menu bar *Automated Version Control.*

Step 3. Review the proposed SPIDERs in the *SPIDER-Edit* panel by selecting the menu item *Edit* in the menu bar *SPIDER.*

Step 4. Review the proposed SPIDERs in the *SPIDER-Decompose* panel by selecting the menu item *Decompose* in the menu bar *SPIDER.*

Step 5. Review component content links of the proposed SPIDERs in the *SPIDER-Component Content* and *Component Content Editor* panels by selecting the menu item *Component Content* in the menu bar *SPIDER.*

Step 6. Review step content data of the proposed SPIDERs in the *SPIDER-Step Content* panel by selecting the menu item *Step Content* in the menu bar

SPIDER.

Step 7. When the SPIDERs are evaluated to determine if they should be approved, select their status *Approved* in the status combo box of the *SPIDER-Step Content* panel.

Step 8. When the SPIDERs are evaluated to determine if they must be abandoned, select their status *Abandoned* in the status combo box of the *SPIDER-Step Content* panel.

d. Schedule and assign a job

The project organizers schedule and assign (or abandon) the SPIDERs of the requirement analysis step by using CASES as the following manipulations:

Step 1. Select the menu item *Scheduling* under the menu bar *Job Schedule.*

Step 2. Select a job scheduling policy and CASES will automatically schedule a job and change the job status into *Scheduled.*

Step 3. Select the menu item *Assignment* under the menu bar *Job Schedule.*

Step 4. Assign a job to a project analyst or a project designer and CASES will automatically change the job status into *Assigned.*

Step 5. When the SPIDERs are abandoned, select their status *Abandoned* in the status combo box of the *SPIDER-Step Content* panel.

e. Complete or decompose a job

The system analysts and designers complete or decompose (or abandon) the SPIDERs of the requirement analysis step by using CASES as the following manipulations:

Step 1. Open the project *c4i* by using the menu item *Open Project* under the menu bar *Project.*

Step 2. Open the step version *s-R1.1* in the *Automated Version Control-Open Step Version* panel by selecting the menu item *Open Step Version* in the menu bar *Automated Version Control.*

Step 3. Select the menu item *Trace* under the menu bar *SPIDER* to understand the step (job) content and component content of the assigned atomic SPIDER, and to trace the software evolution history in the *SPIDER-Trace* panel.

Step 4. Enter or modify component content links of the assigned SPIDERs in the *SPIDER-Component Content* and *Component Content Editor* panels by selecting the menu item *Component Content* in the menu bar *SPIDER.*

Step 5. Select the menu item *Text Editor, MS Word, MS Excel, Netscape,* or *CAPS* under the menu bar *Tools* and carry out the assigned job with the specified tool.

Step 6. When the SPIDERs are completed, select their status *Completed* in the status

combo box of the *SPIDER-Step Content* panel.

Step 7. When the SPIDERs are proposed to decomposition, select their status *Decomposed* in the status combo box of the *SPIDER-Step Content* panel.

Step 8. When the SPIDERs are evaluated to determine if they must be abandoned, select their status *Abandoned* in the status combo box of the *SPIDER-Step Content* panel.

According to the software evolution process with CASES (shown as Figure 16) and the state transition diagram for software evolution steps (shown as Figure 18) in Chapter IV, after the system analysts or designers complete all the jobs in a specified step in the above section ***e***, the project organizers must create a new step and propose new jobs according to the above section ***b***. The SPIDERs of each software evolution step have been created in Appendix C. The software evolution processes of the C4I/MD systems have been developed by CASES, shown as Figure 28 to Figure 160 in Appendix B.

IX. CONCLUSIONS

A. CONCLUSIONS

Applying the RH model and dependency-computing model in the CASES and C4I systems is the result of inferred dependencies. We use formal representation to extend hypergraph and related dependency rules that are the fundamental architecture of CASES to improve the issues of requirements changes and software evolution component reuse.

In order to resolve the software evolution process issues, we have built a new automated software evolution architecture using CASES; in order to resolve the software evolution traceability issues, we enhanced the graph model and the hypergraph model into the RH model. In order to resolve the software evolution description issues, we formalized software evolution objects and their dependencies. In order to resolve software evolution management issues, we determined and computed many types of dependency rules. Finally, in order to resolve the software evolution control issues, we improved the scheduling algorithm and modified a new ECS.

The RH model can be used to describe software evolution history, software evolution object refinement, software evolution process, and SPIDER formation. The dependency-computing model provides a means of automating software evolution. The design of CASES carries out the computer-aided software evolution based on the inferred dependencies. The study of C4I systems provides a basis for understanding how a large and complex software system is developed, and validates the primitive CASES functions: control, management, formation, refinement, traceability, and assignment. In contrast to existing approaches to software evolution, we compare CASES with QSS DOORS, PTS, and ECS from different aspects.

B. LIMITATIONS

Our limitation to identifying the software evolution process is that only the dependency rules inside algorithms and programs are automated. The lightweight inference engine and dependency rules are embedded in the CASES program. This point may be somewhat confusing when one considers traditional program design, because the traditional program design emphasizes logical reasoning by algorithms. We, on the other hand, focus on the dependency rules to express the dependencies among software evolution objects and the reasoning processes after related rules are triggered.

The embedding notwithstanding, the lightweight inference engine and dependency rules can be designed separately. This method allows users to define dependency rules rather than to define only dependencies of objects.

C. FUTURE DIRECTIONS

The remainder of this chapter discusses some possible extensions to the approaches taken in this dissertation and considers computer-aided software evolution.

1. Network manipulation

The current version of CASES, 1.1, is created by JDK 1.17 with Java applications and can only be manipulated in a single local machine. Even though transferring Java applications into Java applets for the purposes of general network manipulation is available, functions of CASES 1.1 should be upgraded to match the following network manipulations:

- relationships among clients and severs,
- data communications,
- distributed database management,
- network security,
- stakeholder management, and
- job assignment channel.

2. Job scheduling improvement

CASES 1.1 provides heuristic job scheduling algorithms to improve the job scheduling of NP-complete problems. There are six scheduling policy heuristics in CASES 1.1 to schedule two jobs to the stakeholders each time. The two job slots for a stakeholder are not enough to analyze dynamic scheduling. We have to improve the scheduling mechanism to assign more than two jobs to the stakeholders to allow users to handle dynamic scheduling.

3. Special skill management

We provide basic skills for stakeholders and jobs. In the future, we have to improve the management of personnel skills. How to train stakeholders for the needs of scheduling and assigning jobs, and how to upgrade and qualify their skill levels are essential problems. CASES has to provide extra functions to match these requirements.

4. Inter-project component reuse

CASES 1.1 can only reuse the software evolution components that are in one project with different versions. In the future, CASES 1.1 should be improved to reuse the components of different projects. Moreover, CASES should reuse components not only in one or more projects but also in reusable software database.

5. Database management

There is no database management system in the current version of CASES 1.1. CASES 1.1 creates several file structures that include a CASES directory, text directory, data file directory, skill directory, and program directory. We need to upgrade the CASES file management system.

The following questions should be understood before creating a database:

- What kind of files should be created?
- What are the attributes in each file?
- What are the user requirements for using the database?
- What are the relationships between CASES and the database?

6. Lightweight inference engine

We design several dependency rules embedded inside the Java programs of CASES 1.1. Even though the separate inference rules and lightweight inference engine using Java is hard to design, in the future at least the inference rules have to be separated from the CASES main program so that stakeholders can modify these rules independently and arbitrarily.

7. Risk assessment

Evaluating the risks that may occur in the software evolution step is the other essential topic in the software evolution study. We list the following future directions of extension:

- Enhance the class structure of CASES for the needs of the risk assessment;
- Collect the timing, skill, and security constraint data from the step content of a SPIDER to assess the risk;
- Calculate the program complexity by the numbers of operators and data streams of PSDL in a real-time embedded system developed by CAPS to evaluate the cost;
- Design a risk assessment step appending in each step of software evolution processes to evaluate the software evolution step's performance; and
- Build a risk assessment model to calculate risk-related data and provide new data for the decision making of impact prevention.

LIST OF REFERENCES

[ABRA95] D. Abramson and R. Sosic, "A Debugging Tool for Software Evolution," *Proceedings of the second Working Conference on Reverse Engineering*, July 14-16, 1995, pp. 282-290.

[AVEL92] G. Avellis, "CASE Support for Software Evolution: A Dependency Approach to Control the Change Process," *Proceedings of the Fifth International Conference on Computer-Aided Software Engineering*, Tecnopolis Csata, Valenzano, Italy, July 6-10, 1992, pp. 62-73.

[BADG88] L. Badger and M. Weiser, "Minimizing Communication for Synchronizing Parallel Dataflow Programs," *Proceedings of the 1988 International Conference on Parallel Processing*, St. Charles, IL, August 15-19, 1988.

[BADR93] S. Badr, "A Model and Algorithms for a Software Evolution Control System," *Ph.D. Dissertation*, Computer Science Department, Naval Postgraduate School, Monterey, CA, December, 1993.

[BADR94] S. Badr and Luqi, "Automation Support for Concurrent Software Engineering," *Proceeding of the 6th International Conference on Software Engineering and Knowledge Engineering*, Jurmala, Latvia, June 20-23, 1994, pp. 46-53.

[BERG89] C. Berge, *Hypergraphs*, North-Holland, 1989.

[BERZ91] V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison-Wesley, 1991.

[BERZ93] V. Berzins, Luqi, and A. Yehudai, "Using Transformations in Specification-Based Prototyping," *IEEE Transactions on Software Engineering,* Vol. 19, No. 5, May 1993, pp. 436-452.

[BERZ97] V. Berzins, O. Ibrahim, and Luqi (1997), "A Requirements Evolution Model for Computer Aided Prototyping," *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering*, Madrid, Spain, June 17-20, 1997, pp. 38-47.

[BERZ98] V. Berzins, "Lightweight Inference for Automation Efficiency," *Proceedings of the 1998 ARO/ONR/NSF/DARPA Monterey Workshop on Engineering Automation for computer Based Systems*, Carmel, California, October 23-26, 1998, pp. 19-29.

[BIMS90] K.D. Bimson and L. B. Burris, "Evolutionary Prototyping: Techniques for Structuring the Iterative Development of Knowledge-Based Systems," *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, Vol. 2, Austin, TX, January 2-5, 1990, pp. 211-219.

[BOWE93] J. Bowen and M. Hinchley, "Seven More Myths of Formal Methods," *IEEE Software*, July 1995, pp. 34-41.

[BROD98] A. Brodeen, G. Hartwig, and M. Lopez, "Multistage Command, Control, and Communications System Evaluation," *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 244-250.

[BROS98] B. M. Brosgol, "A Comparison of the Concurrency and Real-Time Features of Ada 95 and Java," *SIGAda'98 Proceedings*, 1998.

[CALL88] D. Callahan, "The Program Summary Graph and Flow-Sensitive Interprocedural Data Flow Analysis," *Proceeding of the ACM SIGPLAN 88 Conference on Programming Language Design and Implementation*, Atlanta, Ga., Not. 23, 7, June 22-24, 1988, pp. 47-56.

[CAPR94] M. A. M. Capretz and L. F. Capretz, "The Object-Oriented Paradigm for Software Evolution," *Proceedings of the Eighteenth International Conference on Computer Software and Application Conference (COMPSAC)*, November 9-11, 1994, pp. 23-28.

[CENT93] Century Computing, Inc., *TAE Plus User Interface Developer's Guide*, Version 5.3, Century Computing, Inc., September, 1993.

[CHAN73] C. Chang and C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, ACADEMIC PRESS, Inc., 1973.

[CONK88] J. Conklin and M. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions on Office Information System*, Vol. 6, October 1988, pp. 303-331.

[CORM94] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1994.

[CRAI93] D. Craigen, S. Gerhart, and T. Ralston, "An International Survey of Industrial Applications of Formal Methods," *Tech. Report TR GCR 93/626, U. S. Nat'l Inst. of Strandards and Technology,* Washington, D. C., 1993.

[DAMP94] D. A. Dampier, Luqi, and V. Berzins, "Automated Merging of Software Prototypes," *Journal of Systems Integration*, Vol. 4, No. 1, February 1994, pp. 33-49.

[DATT99] S. Datta, C. Weaver, D. Parfitt, and M. Rothstein, "Systems Engineering Life Cycle Management with Traceability," *Proceedings of the Thirteenth International Conference on Systems Engineering*, Las Vegas, Nevada, August 9-12, 1999, pp. SE: 61-66.

[EVAN97] J. Evans, "Project Scheduling Tool," *Master Thesis*, Computer Science Department, Naval Postgraduate School, Monterey, CA, 1997.

[FLAN97] D. Flanagan, *Java in a Nutshell*, Second Edition, O'Reilly & Associates, Inc., 1997.

[GALL93] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, "Directed Hypergraphs and Applications," *Discrete Applied Mathematics,* Vol. 42, 1993, pp. 177-201.

[GOGU96] J. Goguen, D. Nguyen, J. Meseguer, Luqi, D. Zhang, and V. Berzins, "Software Component Search," *Journal of Systems Integration*, Vol. 6, 1996, pp. 93-134.

[HARN99a] M. Harn, V. Berzins, and Luqi, "Software Evolution via Reusable Architecture," *Proceedings of 1999 IEEE Conference and Workshop on Engineering of Computer-Based Systems,* Nashville, Tennessee, March 7-12, 1999, pp. 11-17.

[HARN99b] M. Harn, "Computer-Aided Software Evolution Based on Inferred Dependencies," *Proceedings of Conference on Advanced Information Systems Engineering: 6th Doctoral Consortium,* Heidelberg, Germany, June 14-15, 1999, pp. 116-127.

[HARN99c] M. Harn, V. Berzins, and Luqi "A Dependency Computing Model for Software Evolution," *Proceedings of the Eleventh International Conference on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany, June 17-19, 1999, pp. 278-282.

[HARN99d] M. Harn, V. Berzins, Luqi, and W. Kemple, "Evolution of C4I Systems," *Proceedings of 1999 Command and Control Research and Technology Symposium*, United States Naval War College, Newport, Rhode Island, June 29 - July 1, 1999, pp.1361-1380.

[HARN99e] M. Harn, V. Berzins, and Luqi, "Computer-Aided Software Evolution Based on a Formal Model," *Proceedings of the Thirteenth International Conference on Systems Engineering*, Las Vegas, Nevada, August 9-12, 1999, pp. CS:55-60.

[HARN99f] M. Harn, V. Berzins, and Luqi, "Software Evolution Process via a Relational Hypergraph Model," *Proceedings of IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, Tokyo, Japan, October 5-8, 1999, pp. 599-604.

[HARN99g] M. Harn, V. Berzins, and Luqi, "A Formal Model for Software Evolution," *Proceedings of the 3rd International Conference on Computational Intelligence and Multimedia Applications*, New Delhi, India, September 23-26, 1999, pp. 143-147.

[HIX93] D. Hix and H. R. Hartson (1993), *Developing User Interfaces: Ensuring Usability through Product & Process*, John Wiley & Sons.

[HOPC79] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Language, and Computation*, Addison-Wesley, 1979.

[HORW90] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural Slicling Using Dependence Graphs," *ACM Trans. Programming Languages and Systems,* Vol. 12, No. 1, January 1990, pp.26-60.

[HORW92] S. Horwitz and T. Reps, "The Use of Program Dependence Graphs in Software Engineering," *Proceedings of the 14th International Conference on Software Engineering*, Melbourne, Australia, May 11-15, 1992, pp. 392-411.

[IBRA96] O. M. Ibrahim, "A Model and Decision Support Mechanism for Software Requirements Engineering," *Ph.D. Dissertation*, Computer Science Department, Naval Postgraduate School, Monterey, CA, 1996.

[KEMP98] W. G. Kemple, G. R. Porter, R. Benson, S. G. Hutchins, and S. Hocevar, "Research in the Classroom and Simulation Laboratory: Combining C2 Research and Education," *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 360-366.

[LE99] H. C. T. Le, "Design of a Persistence Server for the Relational Hypergraph Model," *Master Thesis*, Software Engineering Curriculum, Naval Postgraduate School, Monterey, CA, 1999.

[LEE98] J. Lee and K. M. Carley, "Adaptive Strategies for Improving C3 Performance," *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 66-77.

[LEHM91] M. M. Lehman, "Software Engineering, the Software Process and Their Support," *Software Engineering Journal*, Vol. 6, No. 5, September 1991, pp. 243-258.

[LEON97] T. Leonard, V. Berzins, Luqi, and M. J. Holden, "Gathering Requirements from Remote Users," *Proceeding of the ninth International Conference on Tools with Artificial Intelligence*, Newport Beach, California, November 3-8, 1997, pp. 462-471.

[LIEB93] K. J. Lieberherr and C. Xiao, "Object-Oriented Software Evolution," *IEEE Trans. on Software Engineering*, Vol. 19, No. 4, April 1993, pp. 313-343.

[LOSS98] K. Lossau, "Technical Architecture for Distributed C4I-Based Application," *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 221-230.

[LUQI88a] Luqi and M. Ketabchi, "A Computer-Aided Prototyping System," *IEEE Software*, March 1988. pp. 66-72.

[LUQI88b] Luqi and V. Berzins, "Rapidly Prototyping Real-Time Systems," *IEEE Software*, September 1988. pp. 25-36.

[LUQI89] Luqi, "Software Evolution Through Rapid Prototyping," *IEEE Computer*, May 1989, pp. 13-25.

[LUQI90] Luqi, "A Graph Model for Software Evolution," *IEEE Trans. on Software Engineering*, Vol. 16, No. 8, August 1990, pp. 917-927.

[LUQI92] Luqi, "Computer-Aided Prototyping for a Command-And-Control System Using CAPS," *IEEE Software*, January 1992, pp. 56-67.

[LUQI93] Luqi, "How to Use Prototyping for Requirements Engineering," *Proceedings of IEEE/ACM Symposium on Requirements Engineering*, San Diego, CA, January 1993, p. 229.

[LUQI97] Luqi and J. A. Goguen, "Formal Methods Promises and Problems," *IEEE Software*, January 1997, pp. 73-85.

[LYLE86] J. Lyle and M. Weiser, "Experiments on Slicing-Based Debugging Tools," *Proceedings of the First Conference on Empirical Studies of Programming*, June, 1986.

[MADH92] N. Madhavji, "Environment Evolution: The Prism Model of Changes," *IEEE Trans. on Software Engineering*, Vol. 18, No. 5, May 1992, pp. 380-392.

[MALE98] S. Malerud, E. H. Feet, and U. Thorsen, "A Method for Analyzing Command and Control Systems," *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 231-240.

[MANB89] U. Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, 1989.

[MEYE94] B. Meyer, *Reusable software: The base object-oriented component libraries*, Prentice Hall, 1994.

[MOK78] A. K. Mok and M. L. Dertouzos, "Multiprocessor Scheduling in a Hard Real-Time Environment," *Proceedings of the IEEE Real-Time Systems Symposium*, November 1978.

[OTTE84] K. J. Ottenstein and L. M. Ottenstein, "The Program Dependence Graph in a Software Development Environment," *Proceedings of the ACM SIGSOFT/ SIGPLAN Software Engineering Symposium on Practical Software Development Environments,* Pittsburgh, PA, April 23-25, 1984, pp. 177-184.

[PERR94] D. E. Perry, "Dimensions of Software Evolution," *Proceedings of the International Conference on Software Maintenance*, September 19-23, 1994, pp. 296-303.

[PRET93] D. Pretolani, "A Linear Time Alogorithm for Unique Horn Satisfiability," *Information Processing Letters*, Vol. 48, No. 2, November, 1993, pp. 61-66.

[RAMA89a] K. Ramamritham, J. A. Stankovic, and P. Shiah, "Efficient Scheduling Algorithm for Real-Time Multiprocessor Systems," *COINS Technical Report 89-37,* Department of Computer and Information Science, University of Massachusetts, 1989.

[RAMA89b] K. Ramamritham, J. A. Stankovic, P. Shiah, and W. Zhao "Real-Time Scheduling Algorithms for Multiprocessors," *COINS Technical Report 89-47,* Department of Computer and Information Science, University of Massachusetts, 1989.

[RAME95] B. Ramesh and Luqi, "An Intelligent Assistant for Requirements Validation for Embedded Systems," *Journal of Systems Integration*, Vol. 5, No. 2, 1995, pp. 157-177.

[ROET98] W. H. Roetzheim, "Formal Process Improvement - Ignore at Your Own Risk," *Trends in Software Engineering Process Management*, November 1998, http://www.marotz.com/journal/nov98/fpi.htm.

[ROTH99] M. Rothstein, S. Datta, S. Chew, K. Johnson, and L. Gray, "Customizing a Requirements Management Tool for Use in the Corporate MIS Software Project," *Proceedings of Thirteenth International Conference on Systems Engineering*, Las Vegas, Nevada, August 9-12, 1999, pp. SE: 301-307.

[SACC85] D. Saccà, "Closures of Database Hypergraphs," *Journal of the ACM*, Vol. 32, No. 4, October 1985, pp. 774-803.

[SEIT98] L. Seiter, J. Palsberg, K. Leiberherr (1998), "Evolution of Object Behavior Using Context Relations", *IEEE Trans. on Software Engineering*, Vol. 24, No. 1, January, pp. 79-92.

[SOMM96] I. Sommerville, *Software Engineering*, Addison-Wesley, 1996.

[STEI91] R. Steigerwald, Luqi, and J. McDowell, "CASE Tool for Reusable Software Component Storage and Retrieval in Rapid Prototyping," *Information and Software Technology*, England, Vol. 38, No. 9, November 1991, pp. 698-706.

[SUN96] Sun, "The Java™ Phenomenon," document available on-line at http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

[SYMN98] Symantec Corporation, *Symantec VisualCafé*, Database Edition, Version 3.0., Symantec Corporation, 1997-1998.

[ULLM75] J. D. Ullman, "NP-complete Scheduling Problems," *J. Comput. System Sci.*, 10:384-393, 1975.

[WEIS83] M. Weiser, "Reconstructing Sequential Behavior from Parallel Behavior Projections," *Inf. Process, Lett. 17*, October 1983, pp. 129-135.

[WEIS84] M. Weiser, "Program Slicing," *IEEE Trans. Software Engineering,* SE-10, 4, July, 1984, pp. 352-357.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ATTRIBUTES OF FILES

Table 1: Attributes of file: *step.cfg*

Name	Description
stepID	a string to record a step identifier, e.g. s-C
stepName	a string to record a step name of *stepID*, eg. Software prototype demo step
stepDescription	a string to record more detail description of *stepID*

Table 2: Attributes of file: *component.cfg*

Name	Description
componentID	a string to record a component identifier, e.g. C
componentName	a string to record a component name of *componentID*, e.g. Criticism
componentDescription	a string to record more detail description of *componentID*

Table 3: Attributes of file: *loop.cfg*

Name	Description
EHLName	a string to record an evolution history loop name, e.g. Software prototype loop

Table 3: Attributes of file: *loop.cfg*

EHLPath	a string to record a software histotry loop path of *EHLName* with seperator ",", e.g. s-C, s-I, s-R, s-S, s-M, s-P

Table 4: Attributes of file: *dependency.cfg*

Name	Description
step	a string to record a step, e.g. s-C
loopName	a string to record an evolution history loop name of *step*, e.g. Software prototype loop
outputComponent	a string to record output components of *step*, e.g. C
primaryInput	a string to record a primary input component of *step*, e.g. C
secondaryInput	a string to record secondary input components of *step* with seperator "," , e.g. P, T, VT

Table 5: Attributes of file: *current.vsn*

Name	Description
currentStep	a string to record a current step, e.g. s-C
currentLoop	a string to record a current evolution history loop of *currentStep* with seperator ",", e.g. s-C, s-I, s-R, s-S, s-M, s-P
currentVariant	a string to record a current variant number of *currentStep*, e.g. 1
currentVersion	a string to record a current version number of *currentStep*, e.g. 2

Table 6: Attributes of file: *step.cnt*

Name	Description
stepVersion	a string to record a step version, e.g. s-C 1.1
status	a string to record a staus of *stepVersion* specified by the following statuses: proposed, approved, scheduled, assigned, decomposed, completed, and abandoned, e.g. scheduled
skill	a string to record a required skill number of *stepVersion*, e.g. 1
skillLevel	a string to record a skill level number of *skill* specified by 0, 1, 2, and 3, e.g. 3
securityLevel	a string to record a security level number of *stepVersion* specified by 0, 1, 2, 3, 4, and 5, e.g. 1
evaluation	a string to record an evaluation number, e.g. 5
evaluator	a string to record a evaluator identifier of *stepVersion*, e.g. 1500
organizer	a string to record an organizer identifier of *stepVersion*, e.g. 1500
predecessor	a string to record predecessors of *stepVersion* with seperator "," e.g. s-M1.1-3.1, s-M1.1-6.2
priority	a string to record a priority number of *stepVersion* specified by 1, 2, 3, 4, and 5, e.g. 1
estimatedDuration	a string to record an estimated duration day of *stepVersion* specified by a number, e.g. 10

Table 6: Attributes of file: *step.cnt*

deadline	a string to record a deadline of *stepVersion* specified by year, month, and date, e.g. 19990823
earliestStartTime	a string to record an earliest start time of *stepVersion,* specified by year, month, and date, e.g. 19990823
finishTime	a string to record a finish time of *stepVersion,* specified by year, month, and date, e.g. 19990823
manager	a string to record a manager identifier of *stepVersion*, e.g. 1500

Table 7: Attributes of file: *input.p*

Name	Description
(no attribute name)	a string to record primary input components with seperator ",", e.g. C1.1-2, C2.1-1

Table 8: Attributes of file: *input.s*

Name	Description
(no attribute name)	a string to record secondary input components with seperator ",", e.g. P1.1-2, T-P1.1-2, VT-P1.1-2

Table 9: Attributes of file: *txt.link*

Name	Description
(no attribute name)	a string to record text file names with seperator ",", e.g. D:\text\c1001.txt, D:\text\c1002.txt

Table 10: Attributes of file: *word.link*

Name	Description
(no attribute name)	a string to record text file names with seperator ",", e.g. D:\text\c1001.doc, D:\text\c1002.doc

Table 11: Attributes of file: *excel.link*

Name	Description
(no attribute name)	a string to record text file names with seperator ",", e.g. D:\text\c1001.xls, D:\text\c1002.xls

Table 12: Attributes of file: *data.link*

Name	Description
(no attribute name)	a string to record data file names seperated by carriage return, e.g. D:\stakeholder\1500 D:\stakeholder\1510

Table 13: Attributes of file: *url.link*

Name	Description
(no attribute name)	a string to record URLs seperated by carriage return, e.g. http://cs.nps.navy.mil/ file:/n/suns5/capsbuild/c4i/abc.html

Table 14: Attributes of file: *caps.link*

Name	Description
(no attribute name)	a string to record CAPS file names seperated by carriage return, e.g. /.caps/patriot/1.1/patriot.Track.imp.psdl, /.caps/patriot/1.1/patriot.Track.spec.psdl

Table 15: Attributes of stakeholder data files

Name	Description
ID	a string to record a stakeholder identifier, e.g. 1500
name	a string to record a stakeholder name of *ID*, e.g. Hanh Le
skill	a string to record a skill number of *ID*, e.g. 15
skillLevel	a string to record a skill level number of *skill* specified by 0, 1, 2, and 3, e.g. 3
securityLevel	a string to record a security level number of *ID* specified by 0, 1, 2, 3, 4, and 5, e.g. 1

Table 15: Attributes of stakeholder data files

email	a string to record an e-mail of *ID* , e.g. harn@cs.nps.navy.mil
telephone	a string to record a telephone number of *ID*, e.g. 831-6562615
fac	a string to record a facimile number of *ID*, e.g. 831-6563225
address	a string to record an address of *ID*, e.g. SGC 1640, NPS, Monterey, CA, 93940
majorJobs	a string to record on-hand jobs of *ID* with seperator ",", e.g. s-C1.2-1.1, s-C1.2-1.2
minorJobs	a string to record on-hand jobs of *ID* with seperator ",", e.g. s-C1.2-1.1, s-C1.2-1.2

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. CASES USER INTERFACE

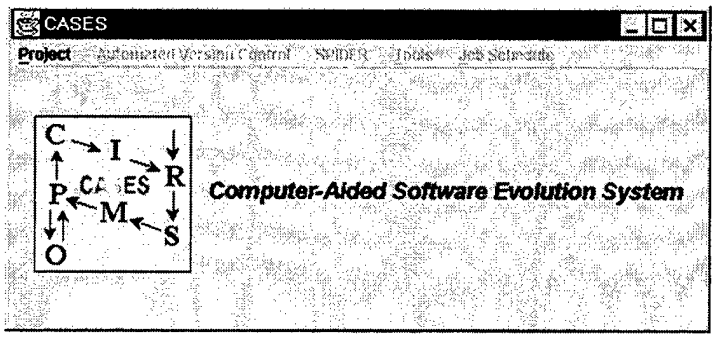


Figure 28: CASES (1)

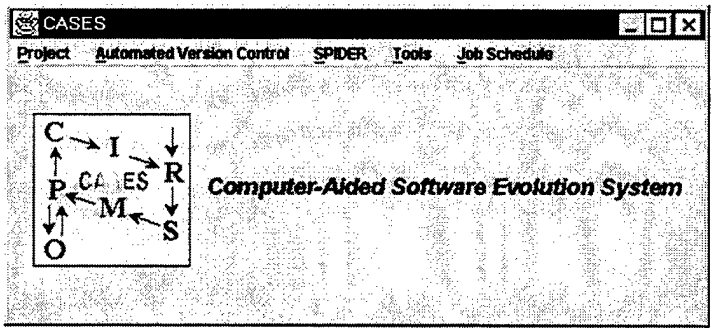


Figure 29: CASES (2)

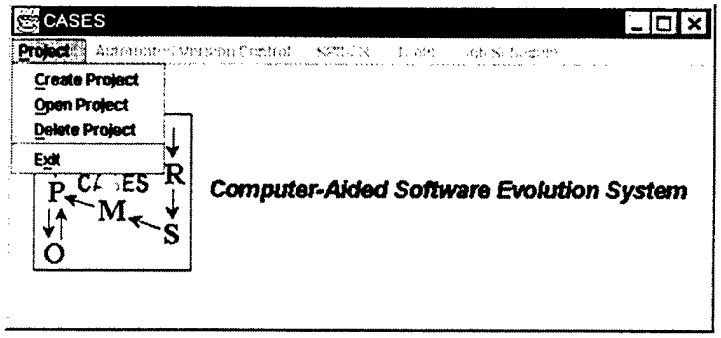


Figure 30: Project menu bar

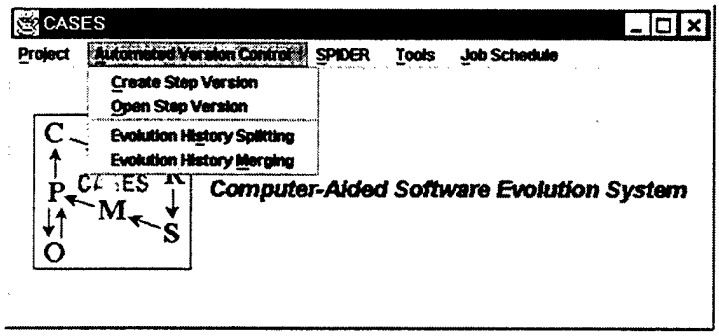


Figure 31: Automated version control menu bar

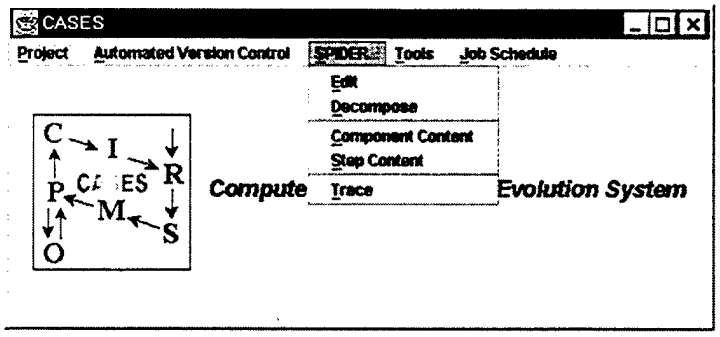


Figure 32: SPIDER menu bar

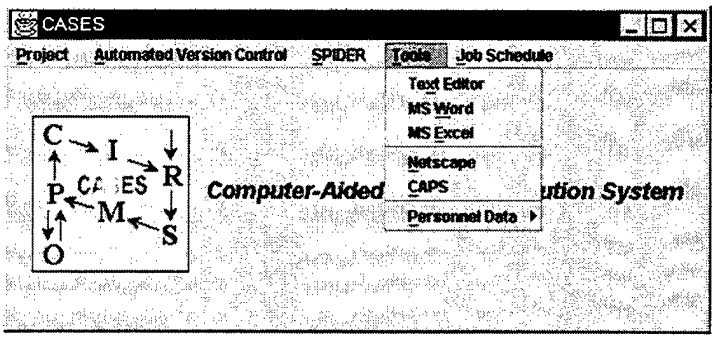


Figure 33: Tools menu bar

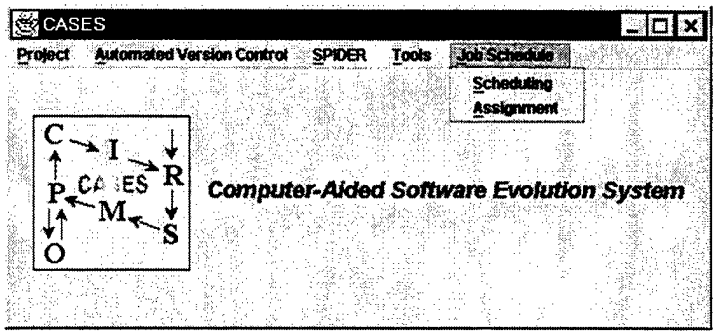


Figure 34: Job schedule menu bar

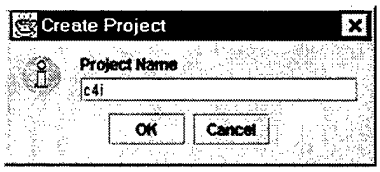


Figure 35: Create project frame

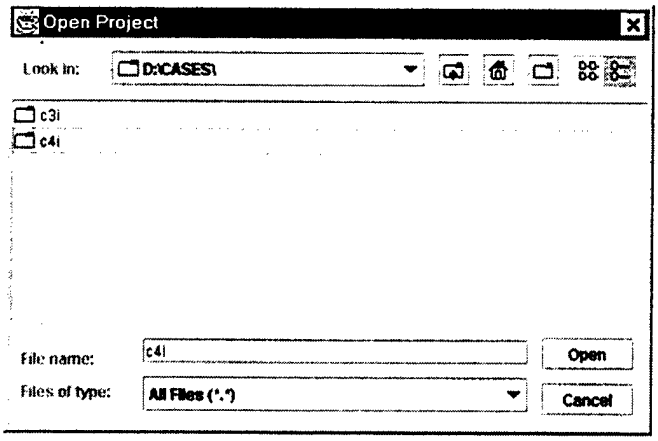


Figure 36: Open project – file chooser (1)

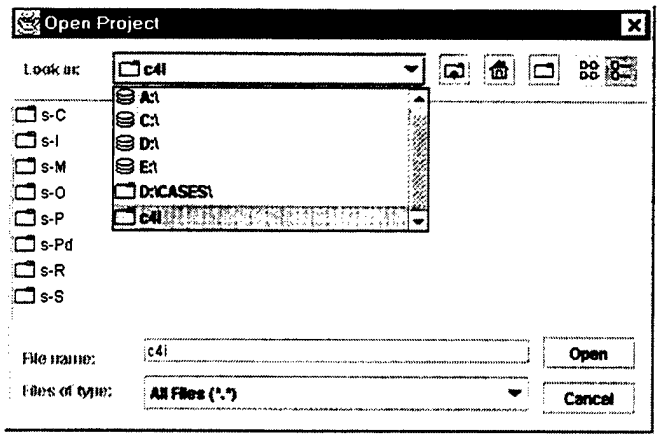


Figure 37: Open project – file chooser (2)

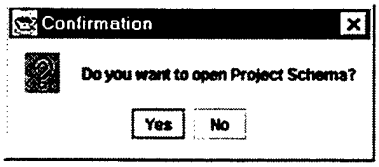


Figure 38: Open project – confirmation

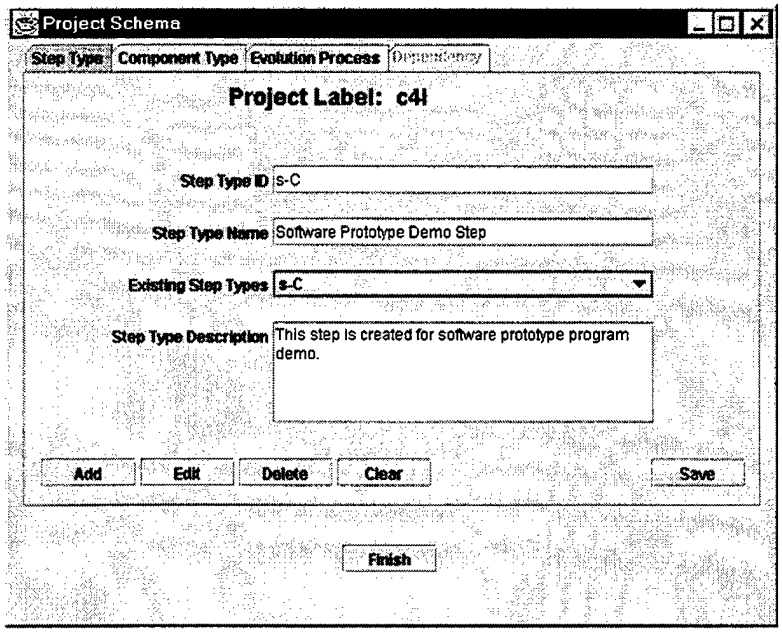


Figure 39: Project schema – step type (1)

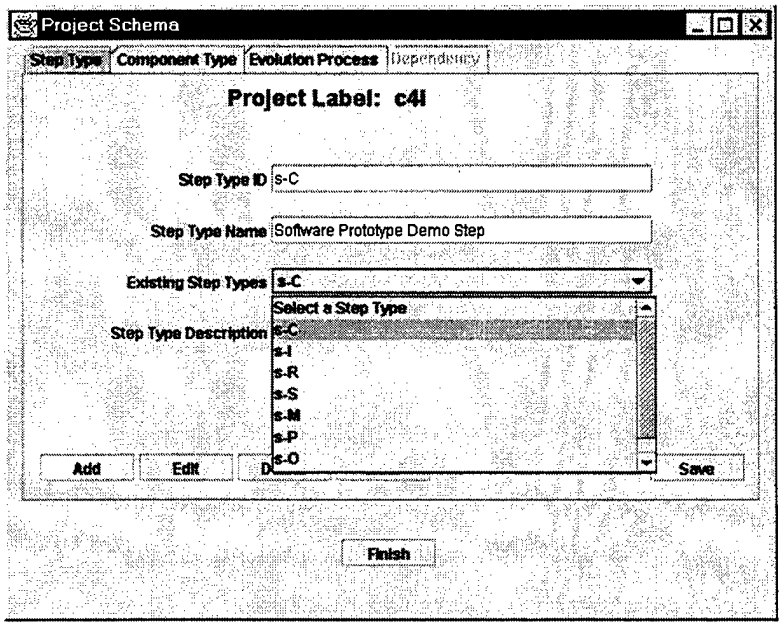


Figure 40: Project schema – step type (2)

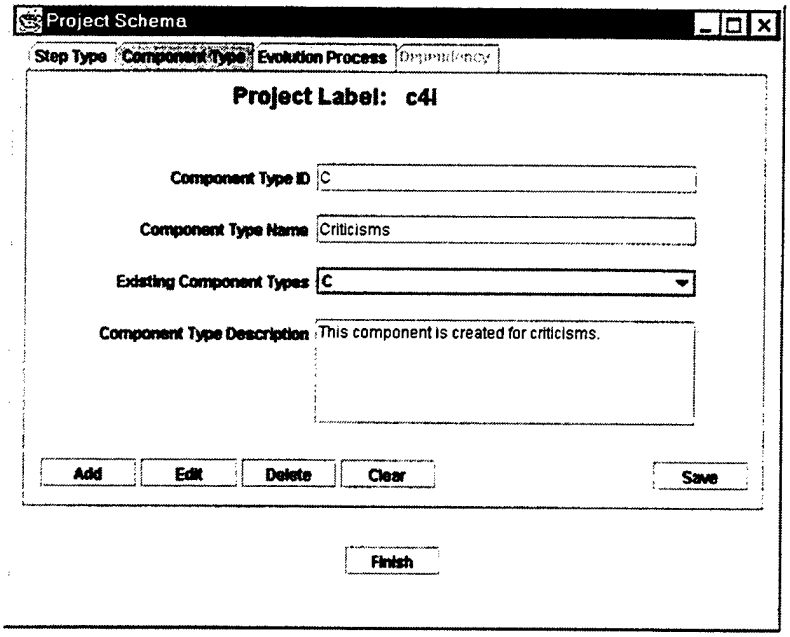


Figure 41: Project schema – component type (1)

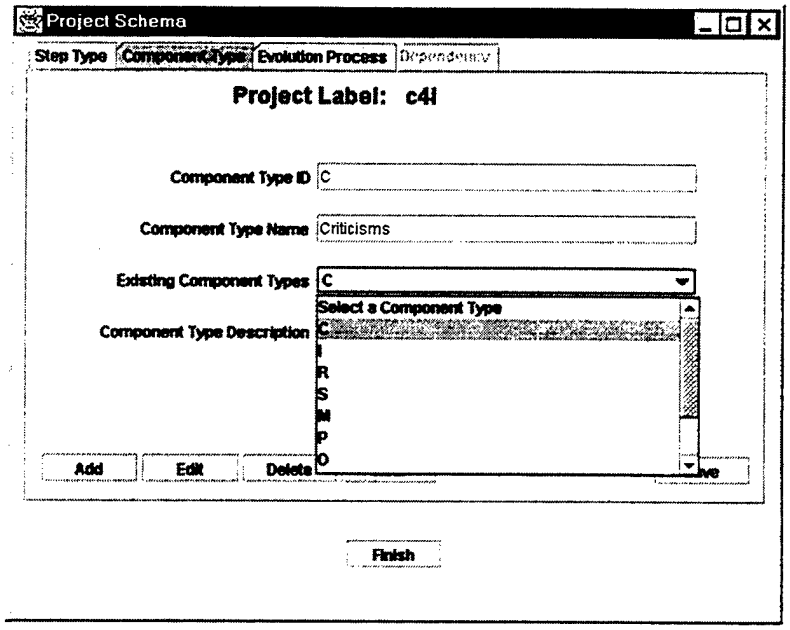


Figure 42: Project schema – component type (2)

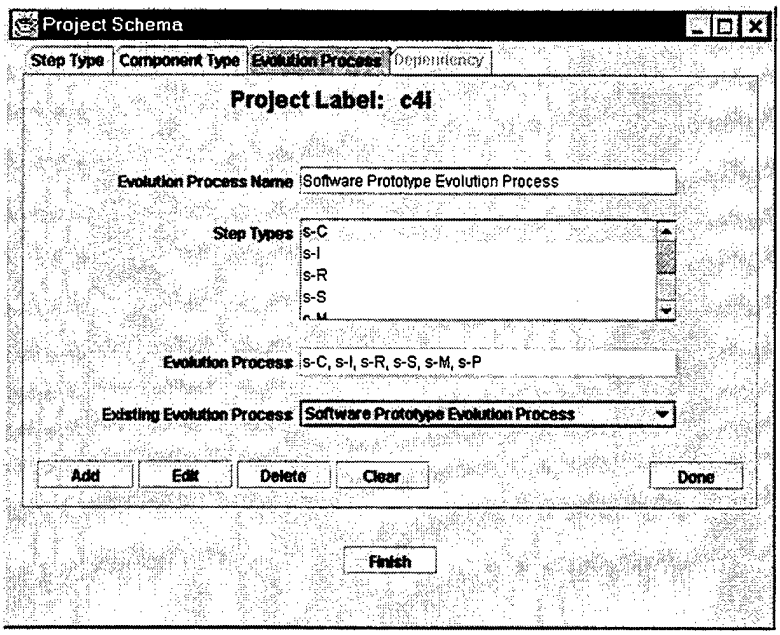


Figure 43: Project schema – evolution process (1)

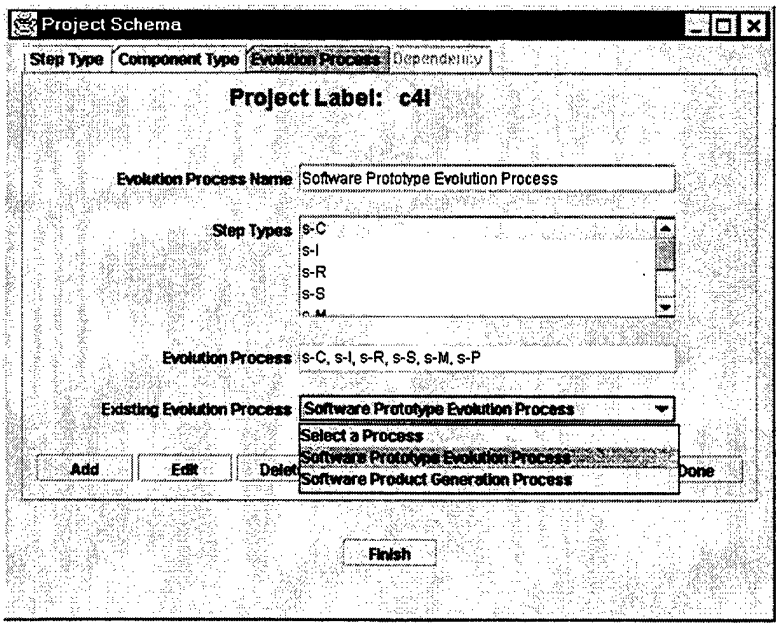


Figure 44: Project schema – evolution process (2)

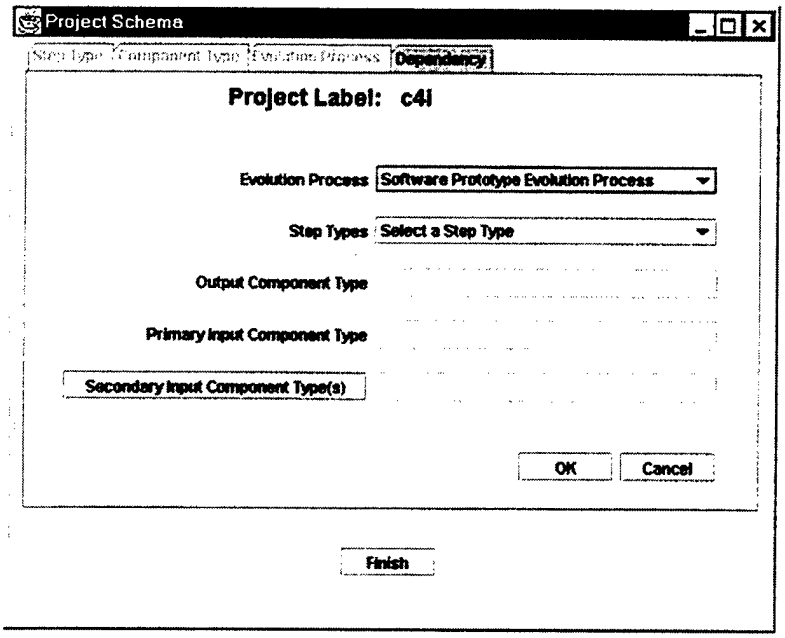


Figure 45: Project schema – dependency (1)

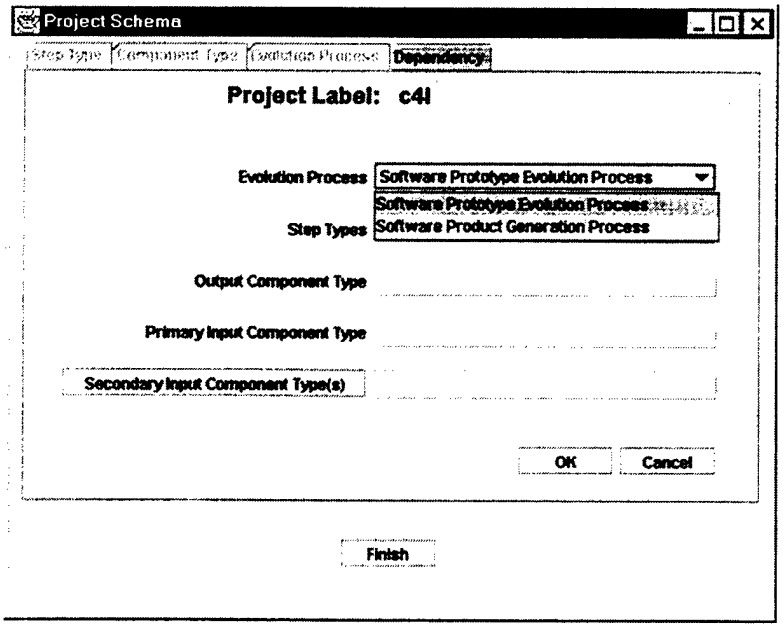


Figure 46: Project schema – dependency (2)

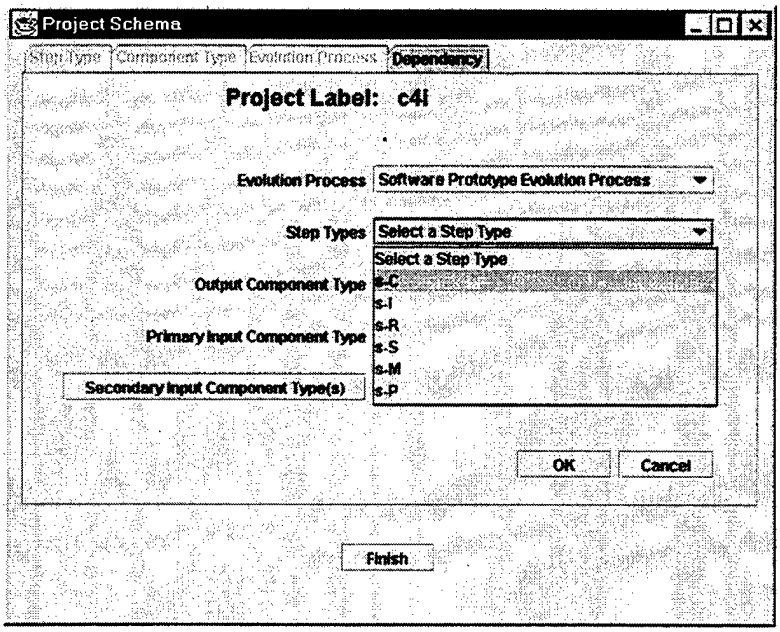


Figure 47: Project schema – dependency (3)

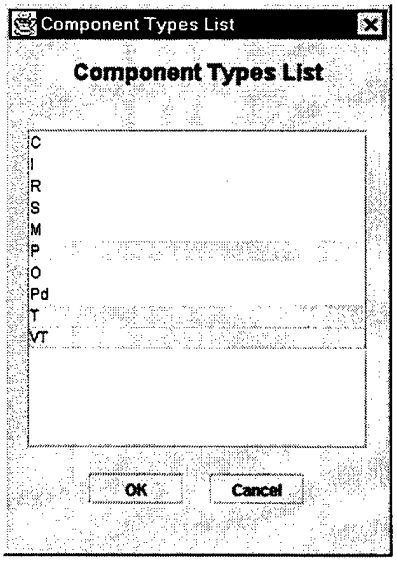


Figure 48: Project schema – dependency (4)

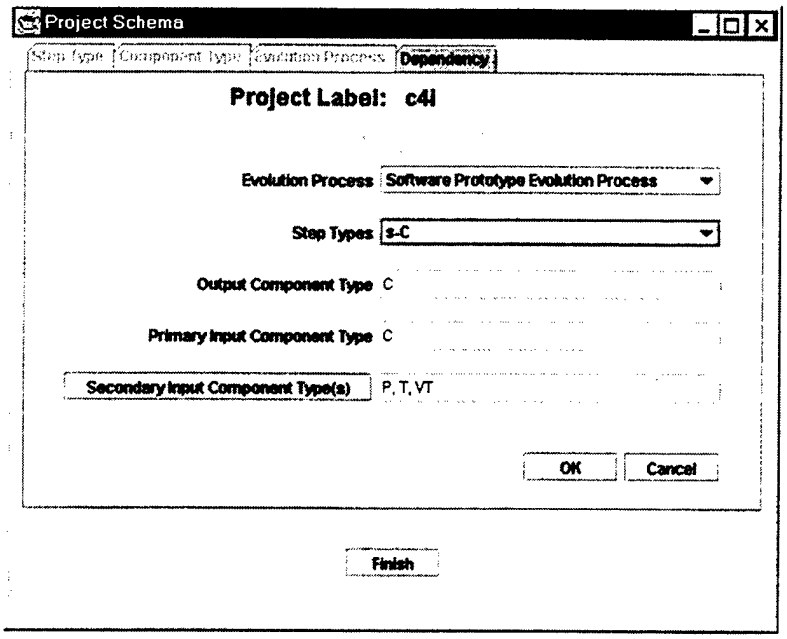


Figure 49: Project schema – dependency (5)

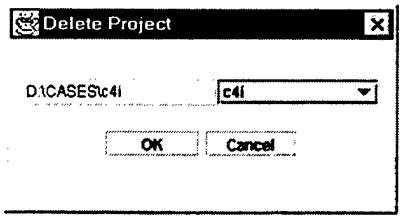


Figure 50: Delete project (1)

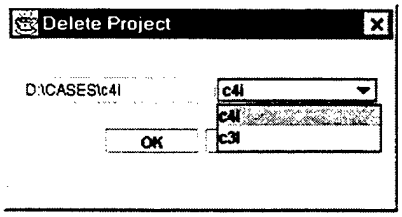


Figure 51: Delete project (2)

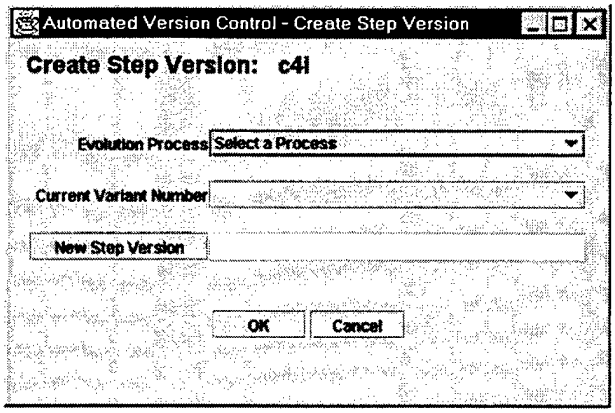


Figure 52: Automated version control – create step version (1)

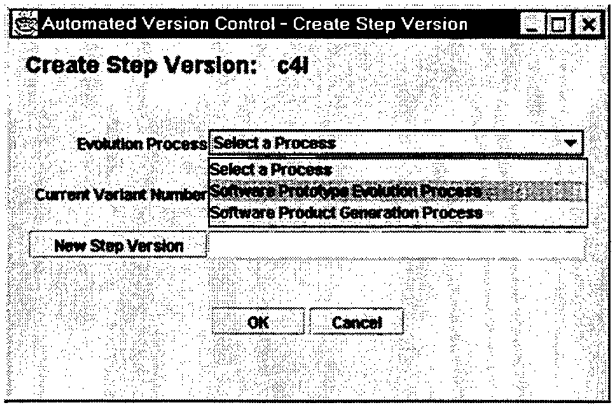


Figure 53: Automated version control – create step version (2)

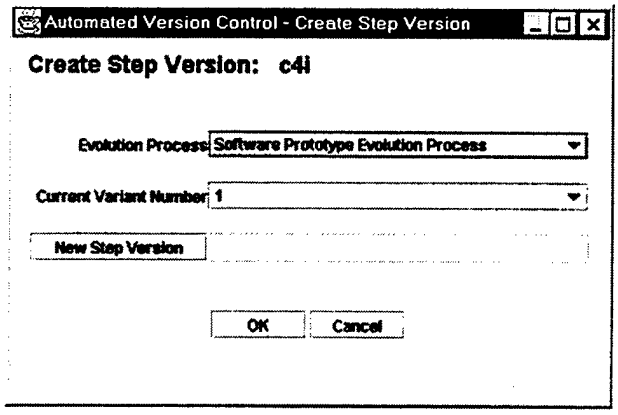


Figure 54: Automated version control – create step version (3)

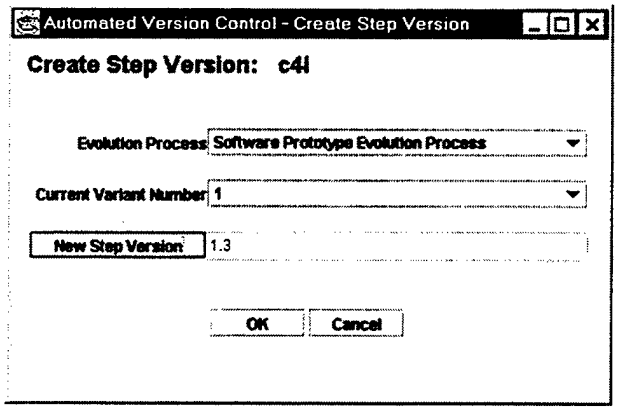


Figure 55: Automated version control – create step version (4)

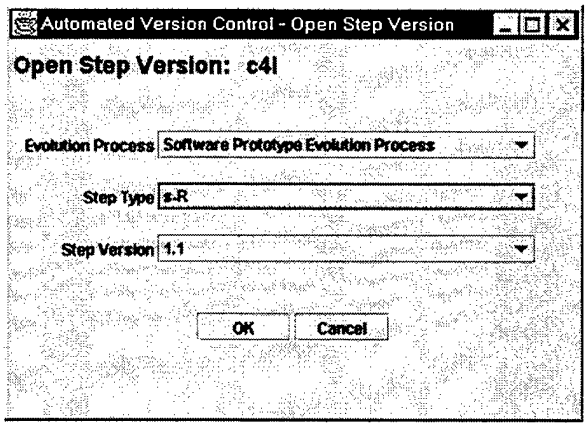


Figure 56: Automated version control – open step version (1)

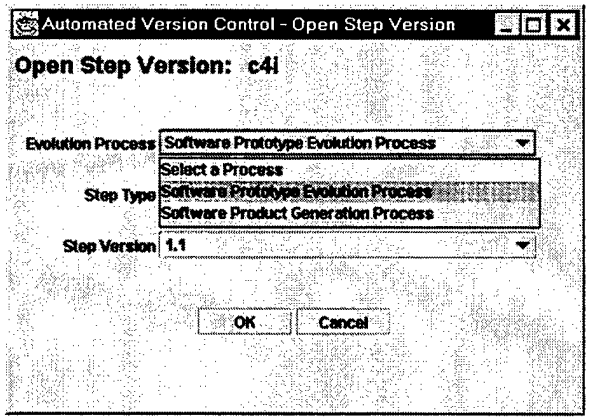


Figure 57: Automated version control – open step version (2)

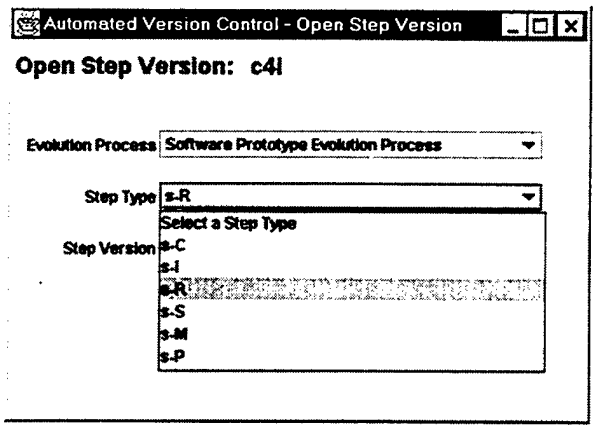


Figure 58: Automated version control – open step version (3)

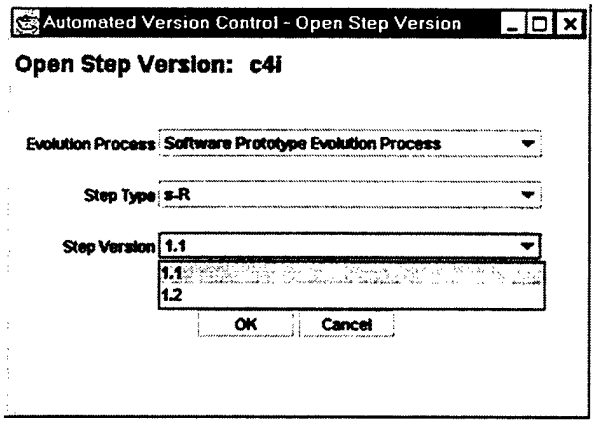


Figure 59: Automated version control – open step version (4)

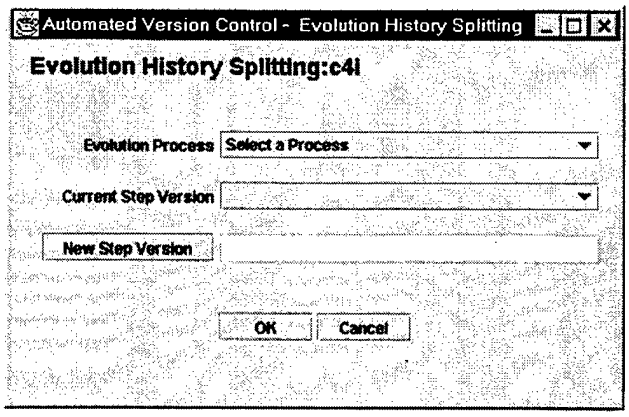


Figure 60: Automated version control – evolution history splitting (1)

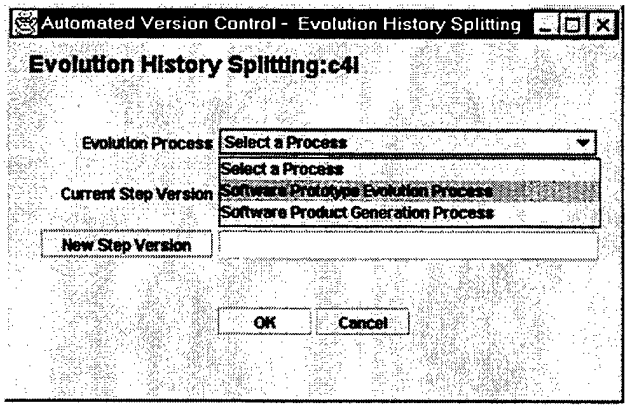


Figure 61: Automated version control – evolution history splitting (2)

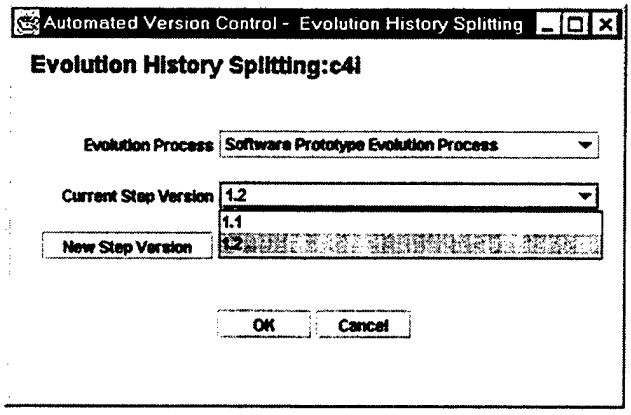


Figure 62: Automated version control – evolution history splitting (3)

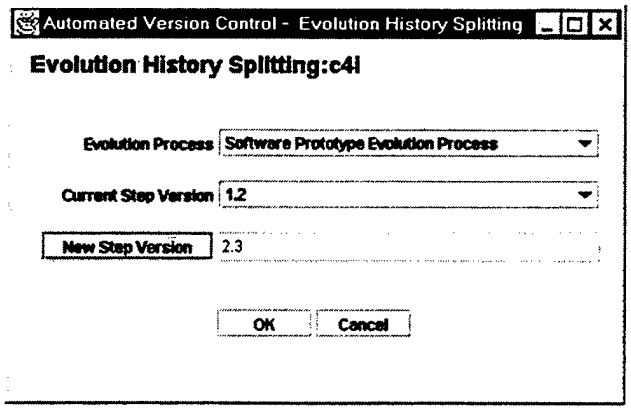


Figure 63: Automated version control – evolution history splitting (4)

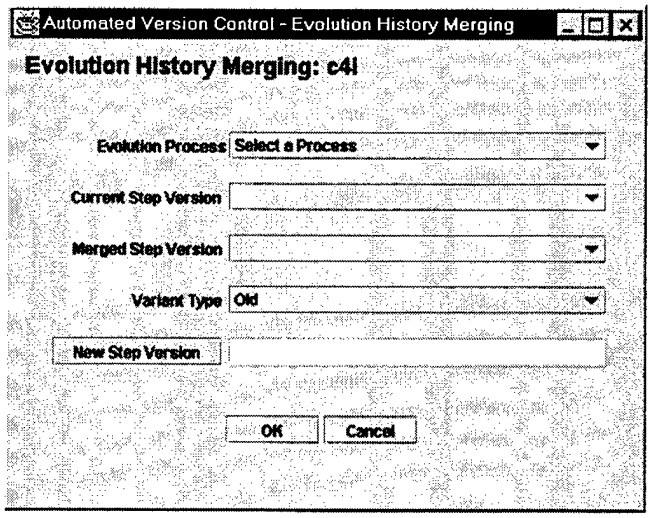


Figure 64: Automated version control – evolution history merging (1)

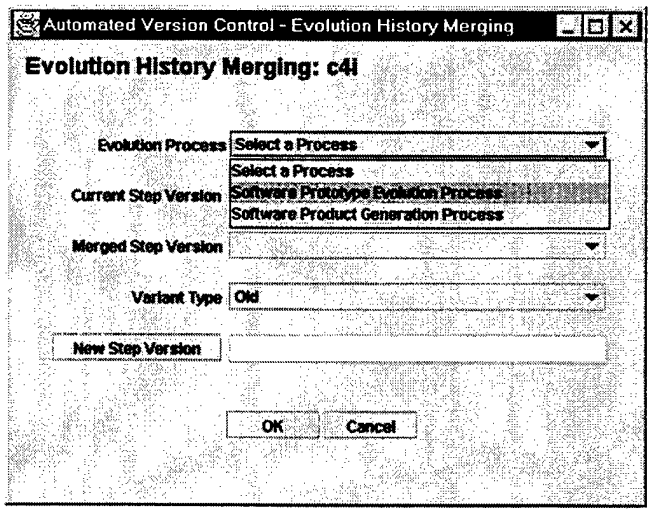


Figure 65: Automated version control – evolution history merging (2)

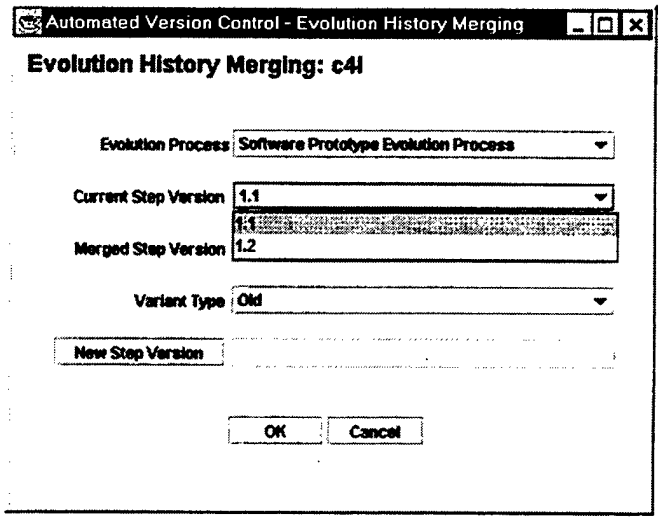


Figure 66: Automated version control – evolution history merging (3)

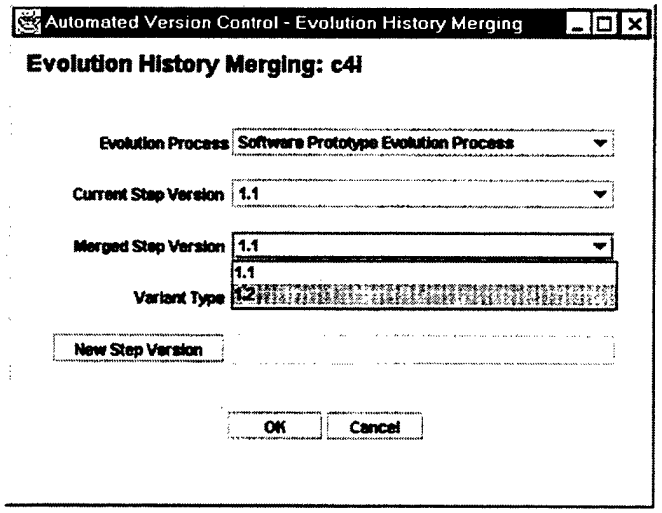


Figure 67: Automated version control – evolution history merging (4)

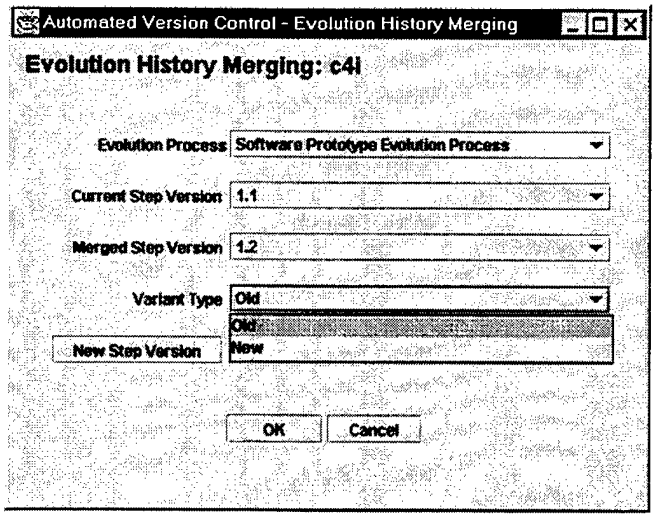


Figure 68: Automated version control – evolution history merging (5)

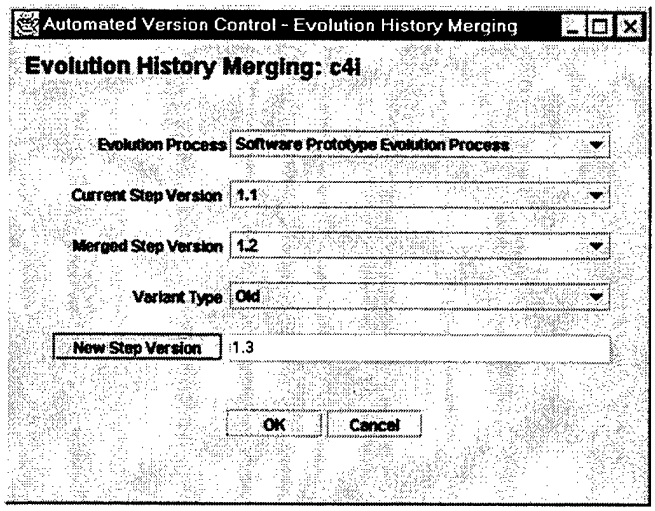


Figure 69: Automated version control – evolution history merging (6)

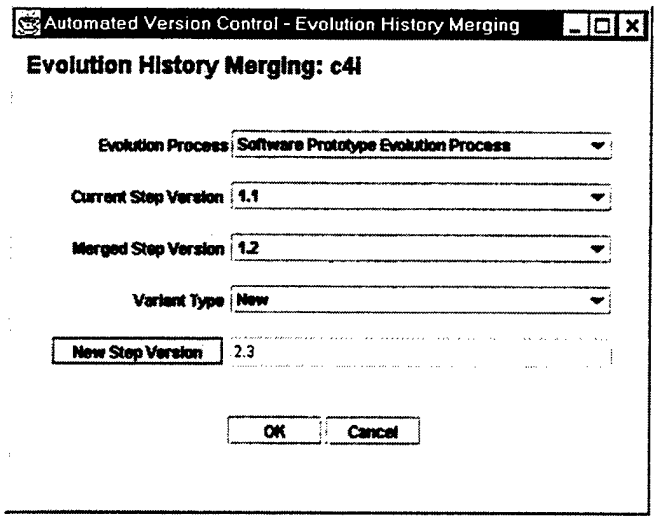


Figure 70: Automated version control – evolution history merging (7)

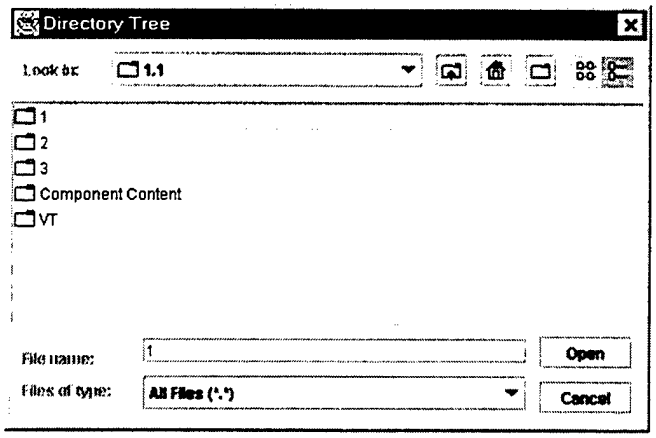


Figure 71: File chooser (1) of SPIDER – edit

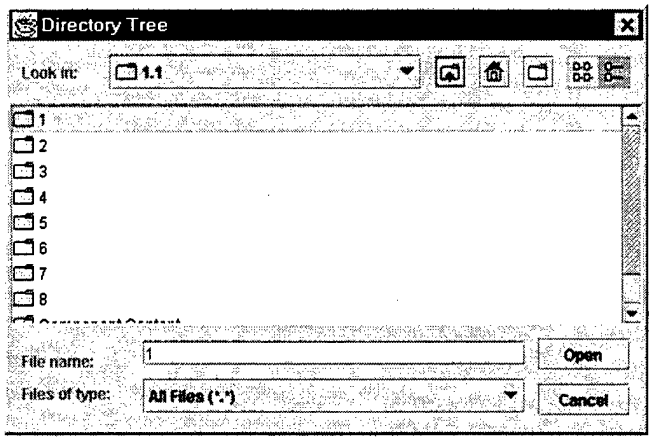


Figure 72: File chooser (2) of SPIDER – edit

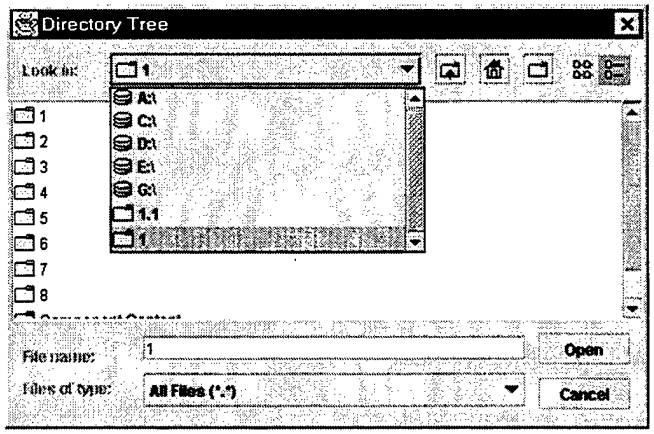


Figure 73: File chooser (3) of SPIDER – edit

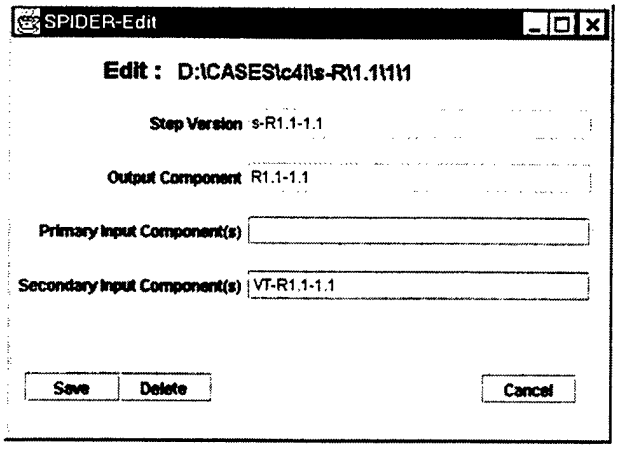


Figure 74: SPIDER – edit

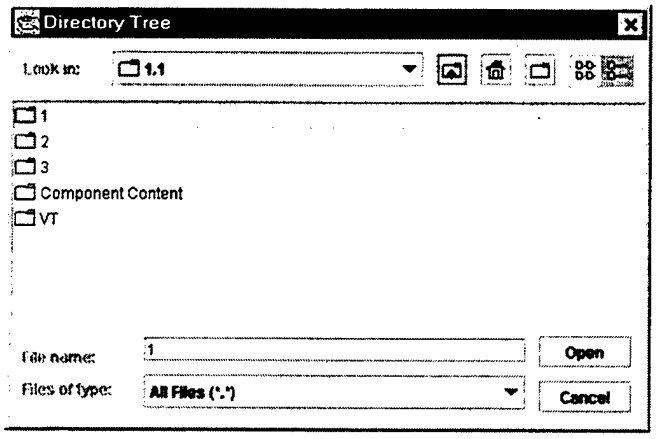


Figure 75: File chooser (1) of SPIDER – decompose

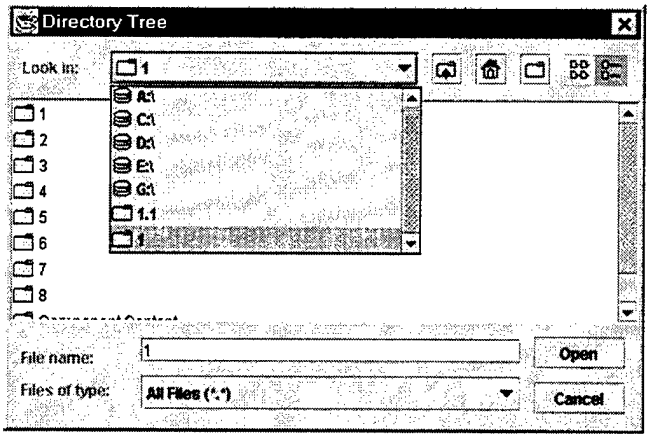


Figure 76: File chooser (2) of SPIDER – decompose

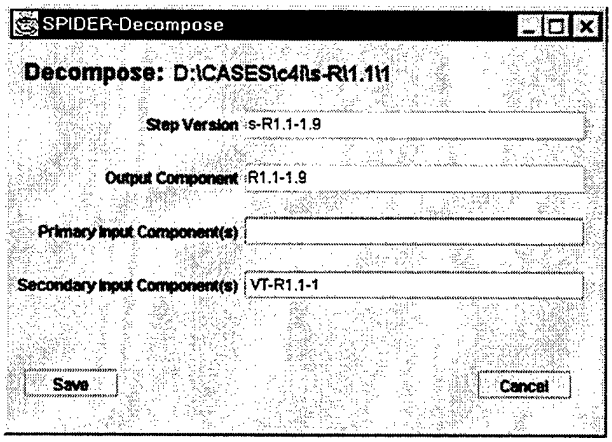


Figure 77: SPIDER – decompose

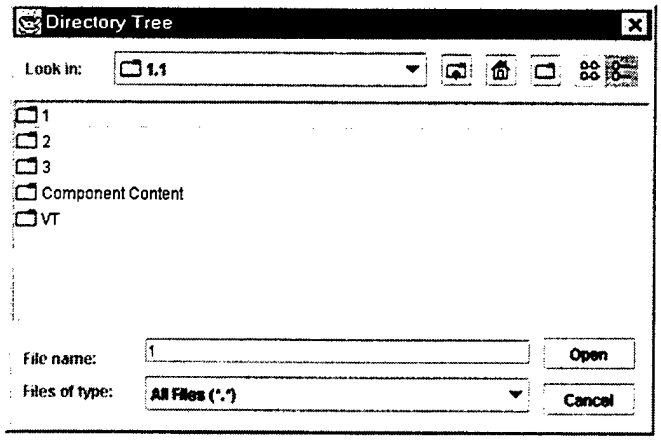


Figure 78: File chooser (1) of SPIDER – component content

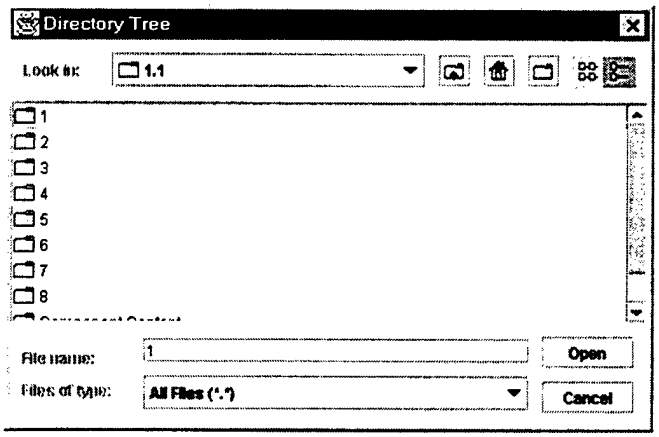


Figure 79: File chooser (2) of SPIDER – component content

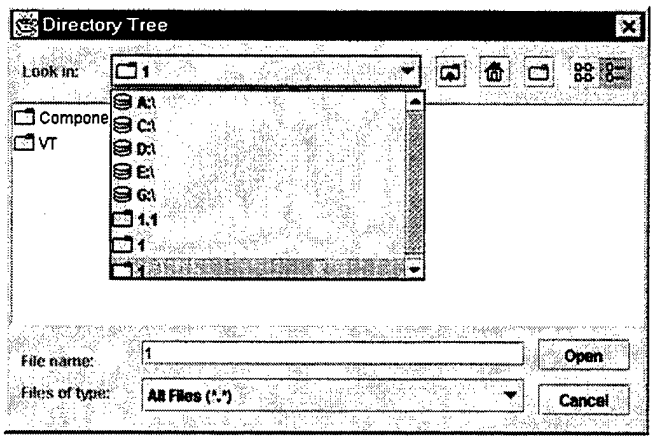


Figure 80: File chooser (3) of SPIDER – component content

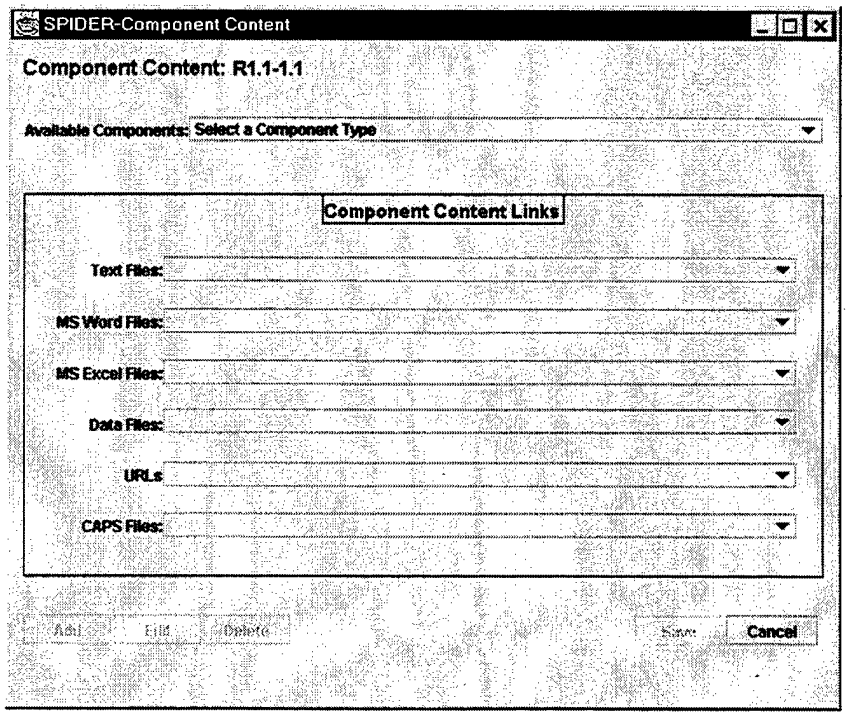


Figure 81: SPIDER – component content (1)

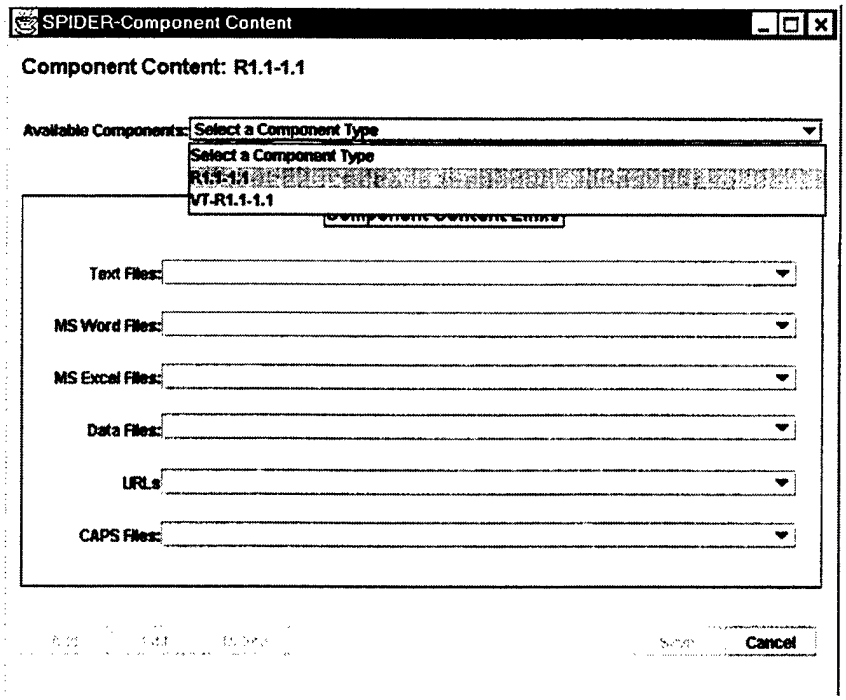


Figure 82: SPIDER – component content (2)

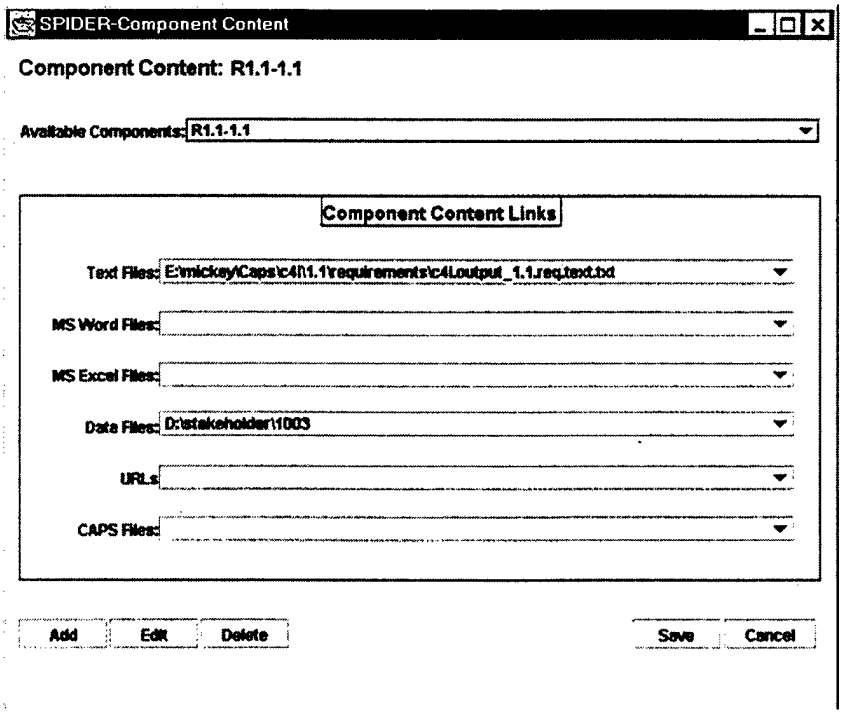


Figure 83: SPIDER – component content (3)

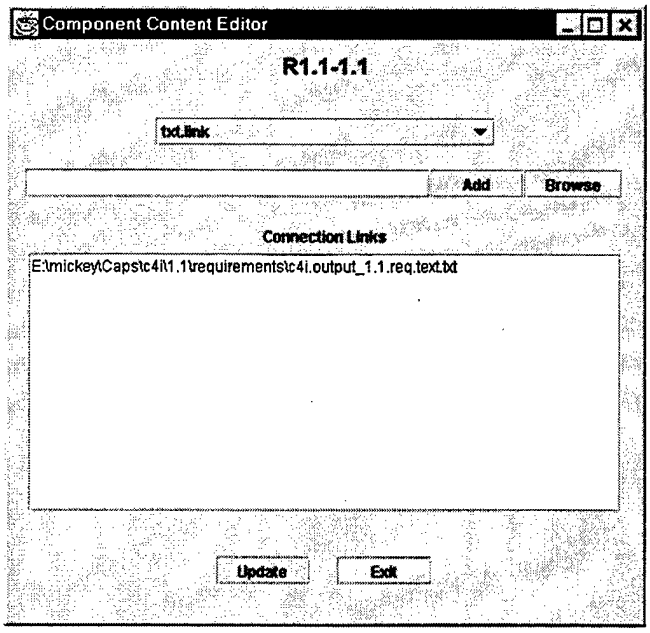


Figure 84: Add a component content in the component content editor (1)

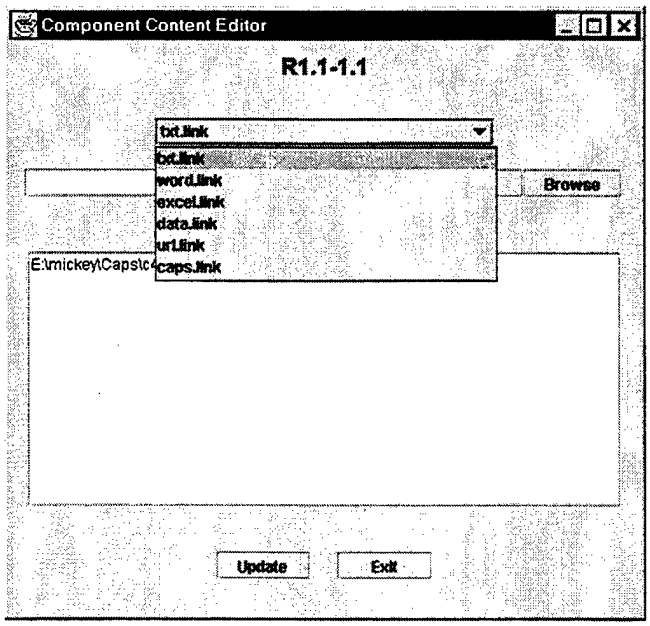


Figure 85: Add a component content in the component content editor (2)

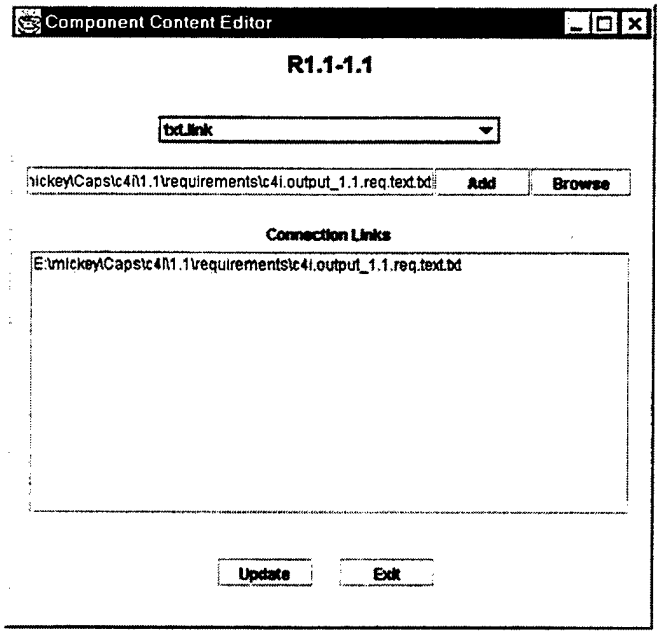


Figure 86: Add a component content in the component content editor (3)

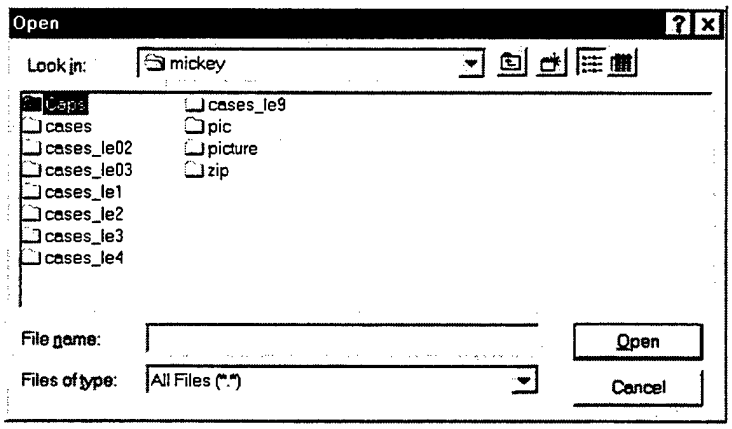


Figure 87: Add a component content via browser (1)

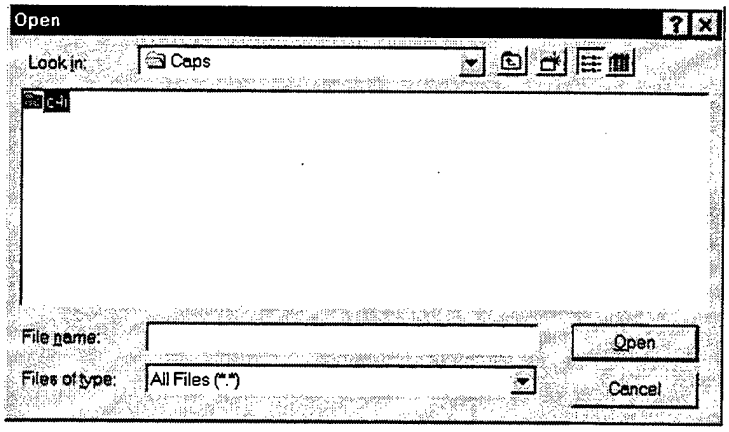


Figure 88: Add a component content via browser (2)

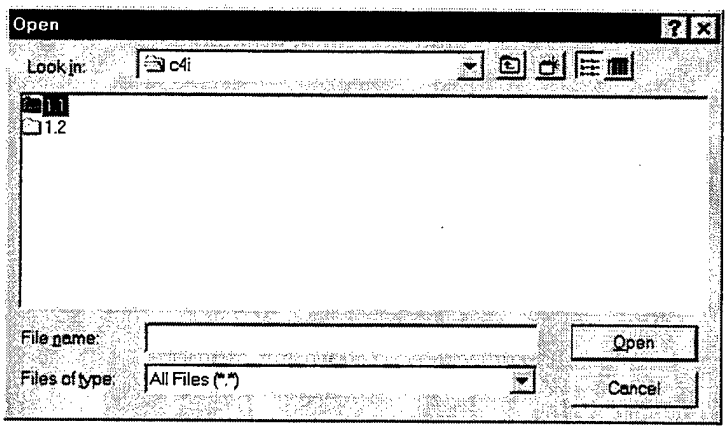


Figure 89: Add a component content via browser (3)

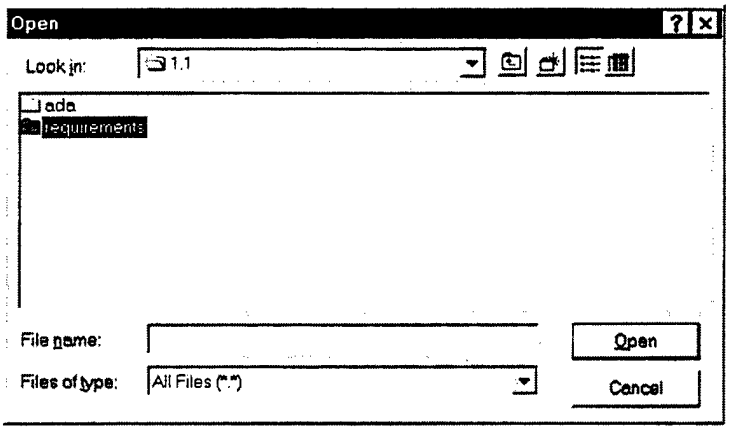


Figure 90: Add a component content via browser (4)

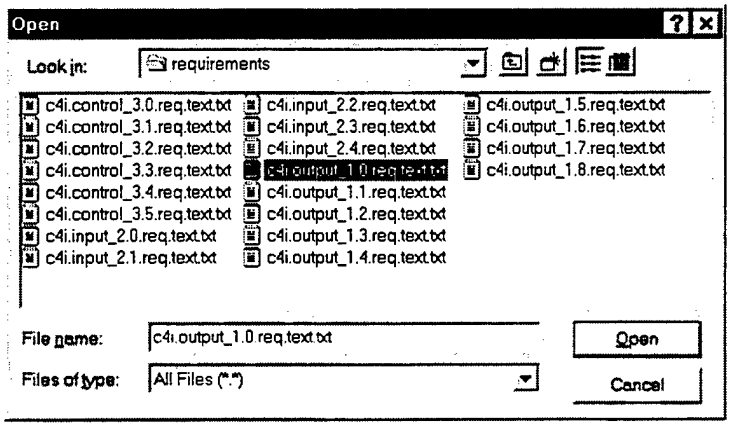


Figure 91: Add a component content via browser (5)

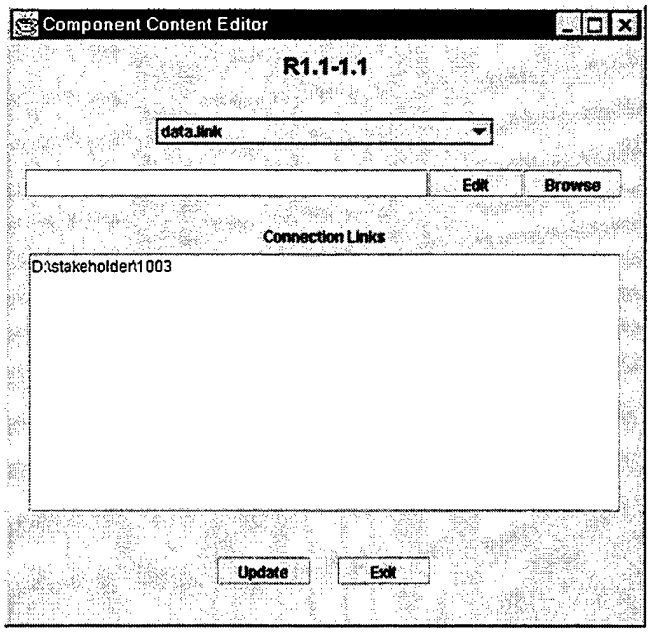


Figure 92: Edit a component content in the component content editor (1)

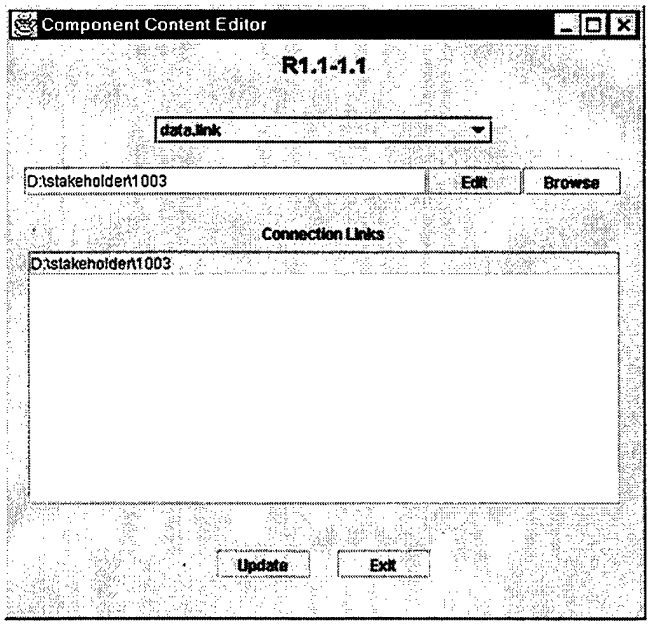


Figure 93: Edit a component content in the component content editor (2)

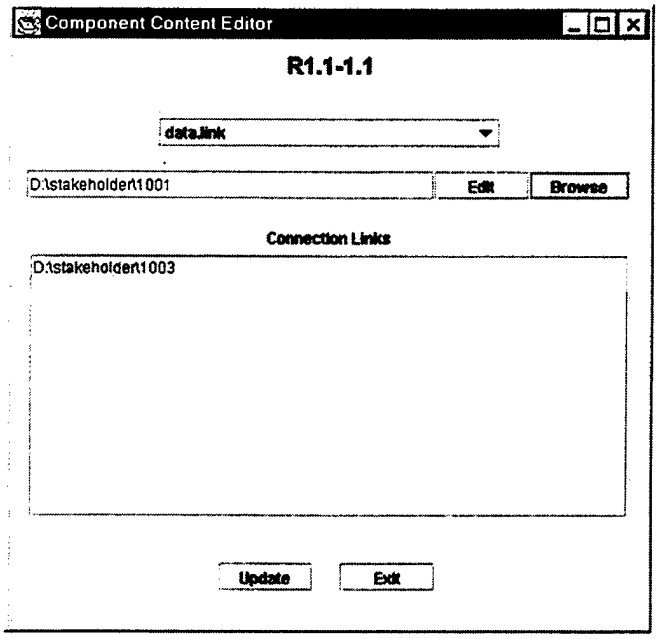


Figure 94: Edit a component content in the component content editor (3)

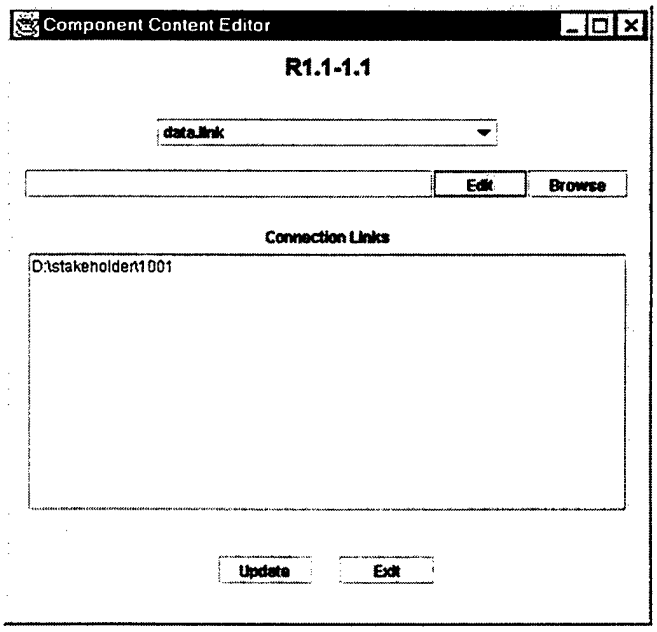


Figure 95: Edit a component content in the component content editor (4)

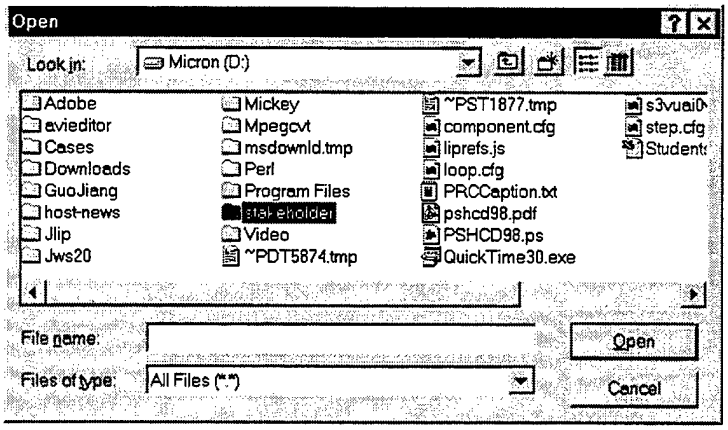


Figure 96: Edit a component content via browser (1)

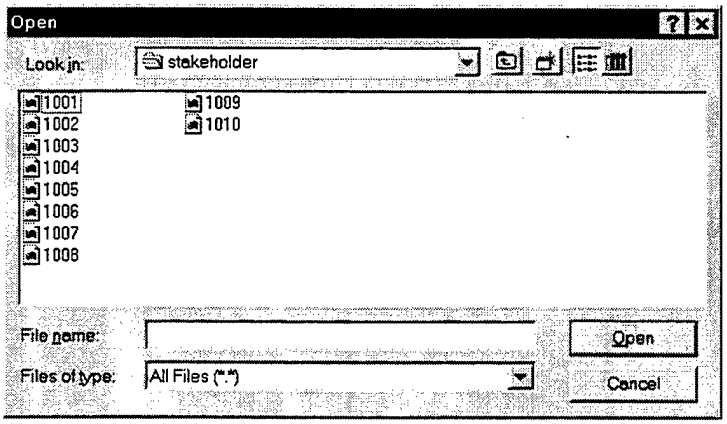


Figure 97: Edit a component content via browser (2)

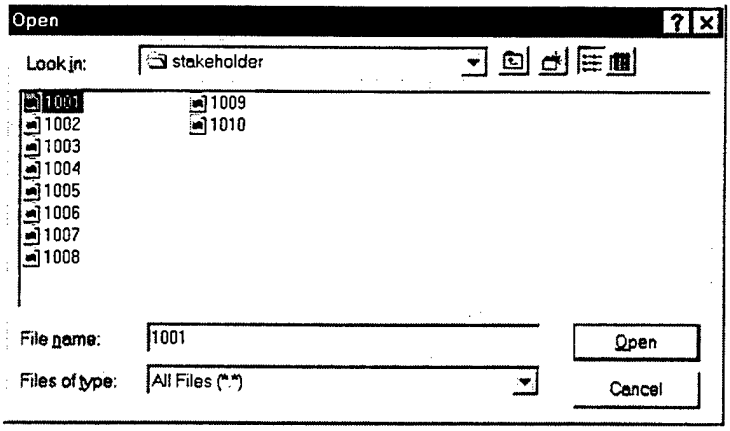


Figure 98: Edit a component content via browser (3)

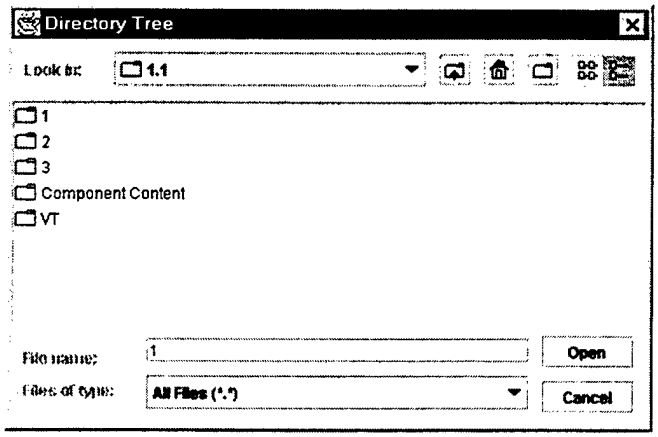


Figure 99: File chooser (1) of SPIDER – step content

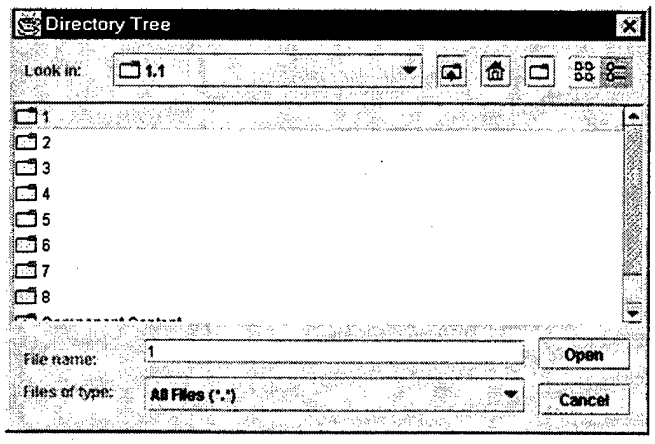


Figure 100: File chooser (2) of SPIDER – step content

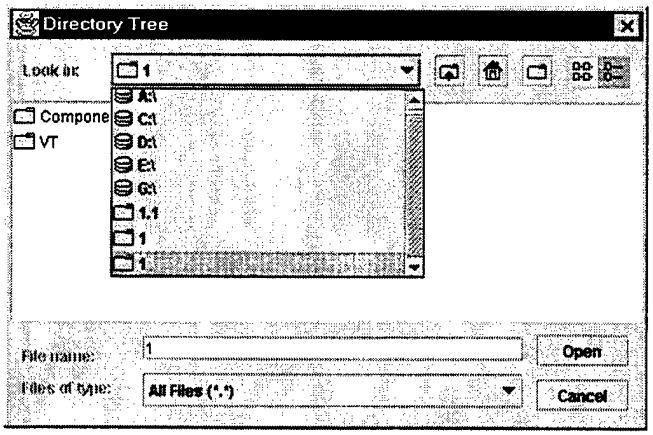


Figure 101: File chooser (3) of SPIDER – step content

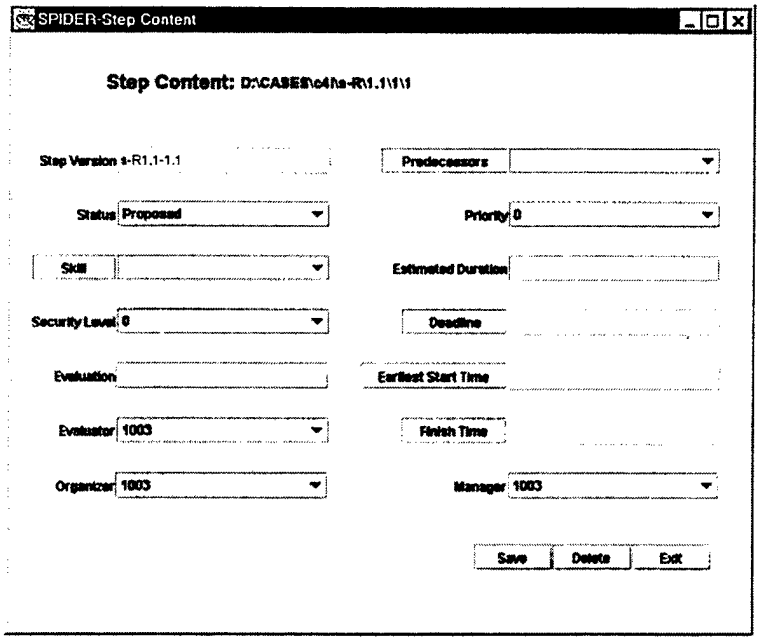


Figure 102: SPIDER – step content (1)

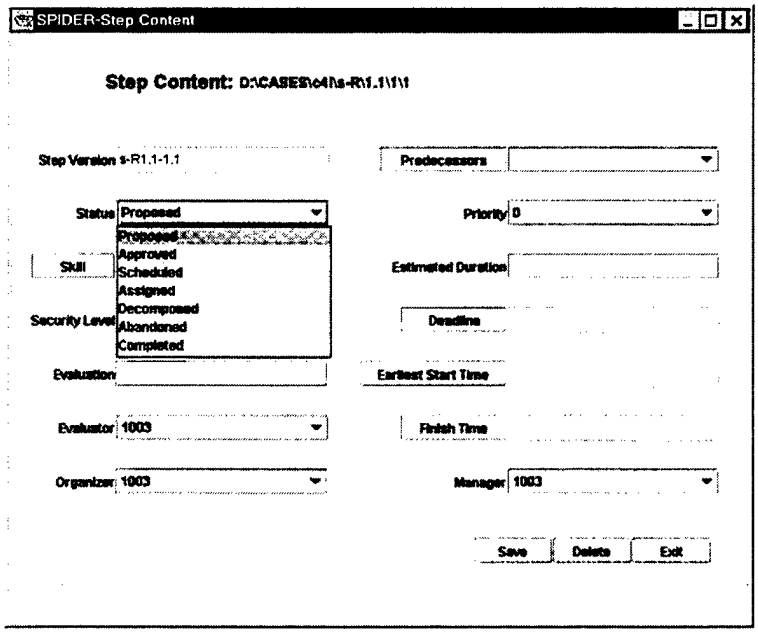


Figure 103: SPIDER – step content (2)

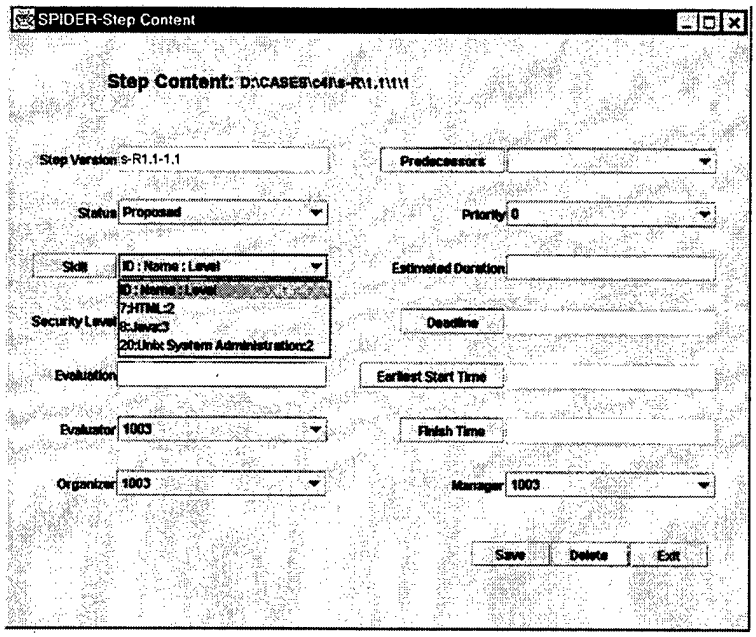


Figure 104: SPIDER – step content (3)

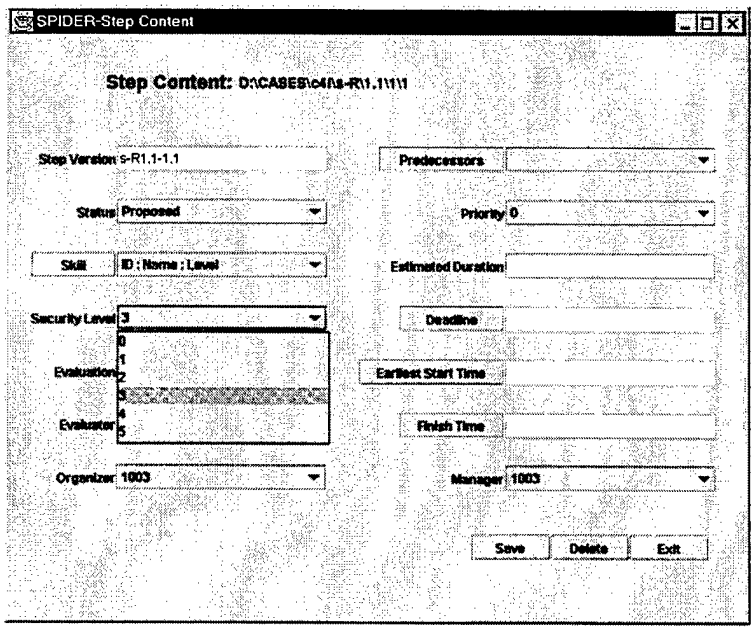


Figure 105: SPIDER – step content (4)

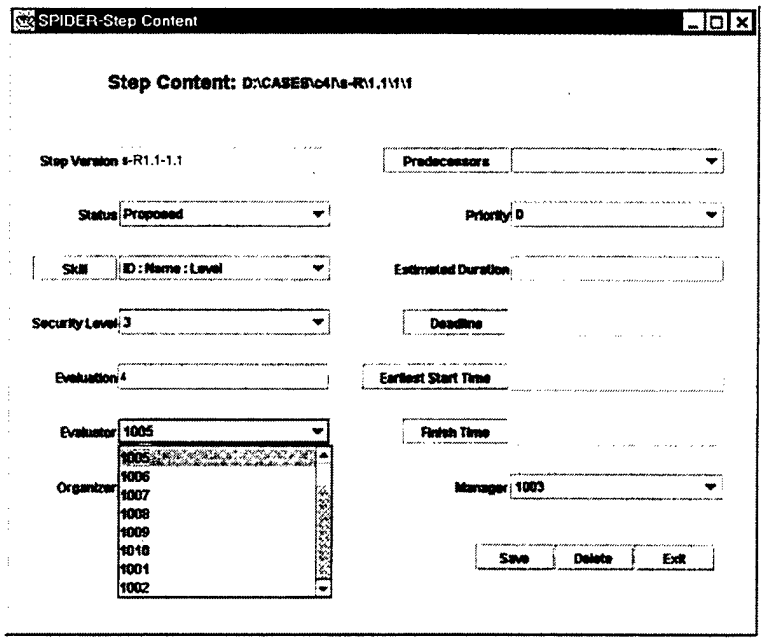


Figure 106: SPIDER – step content (5)

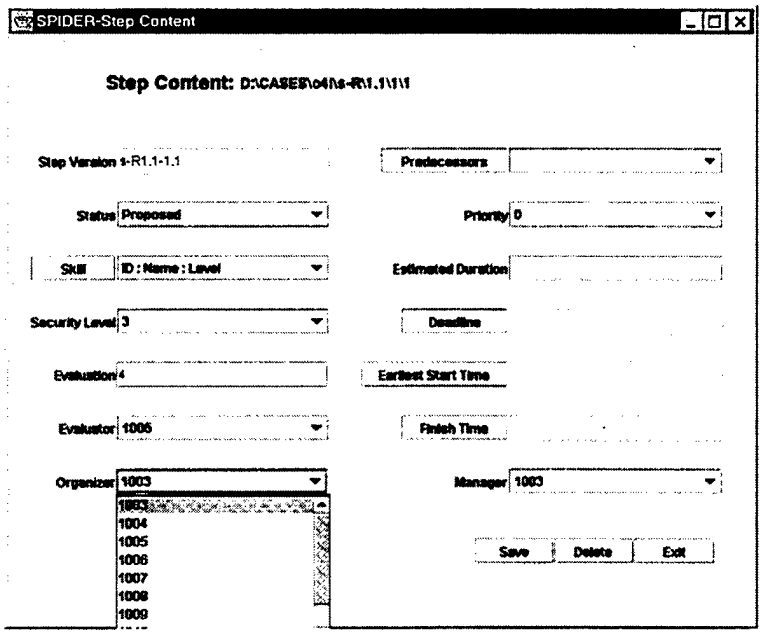


Figure 107: SPIDER – step content (6)

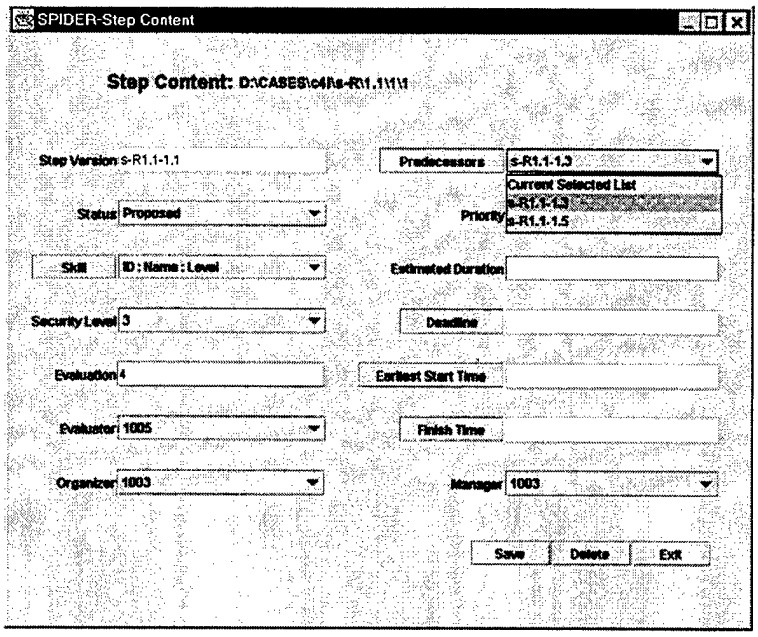


Figure 108: SPIDER – step content (7)

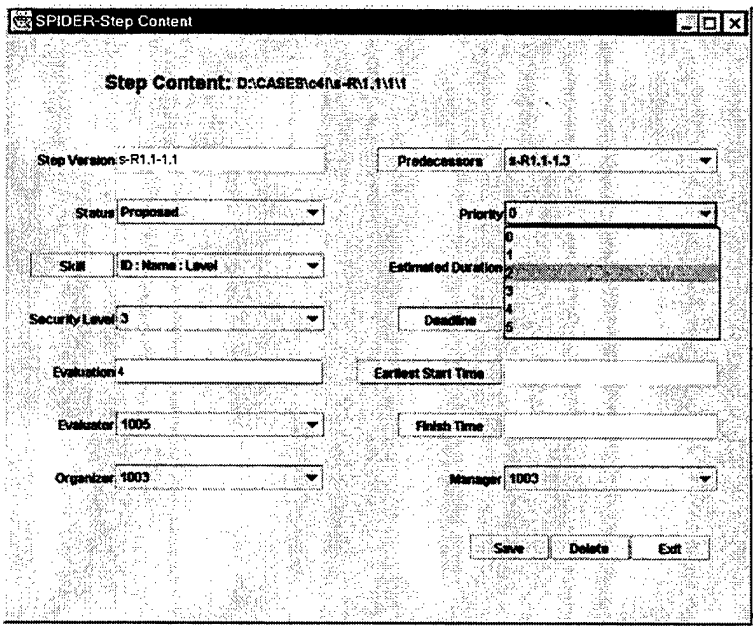


Figure 109: SPIDER – step content (8)

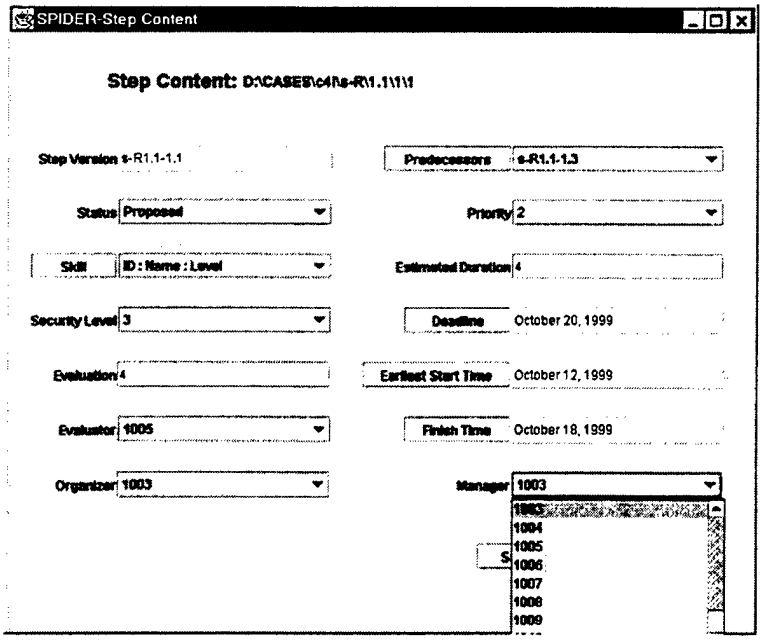


Figure 110: SPIDER – step content (9)

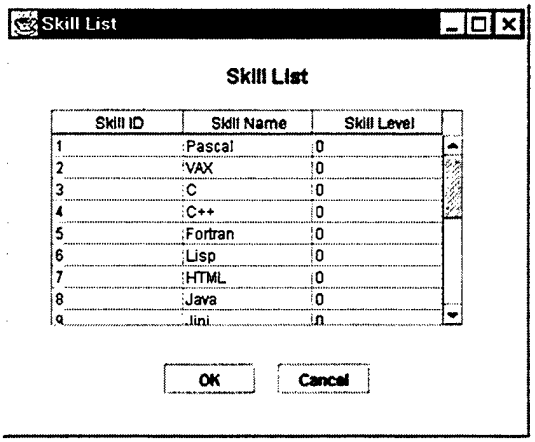


Figure 111: Skill list (1) of SPIDER – step content

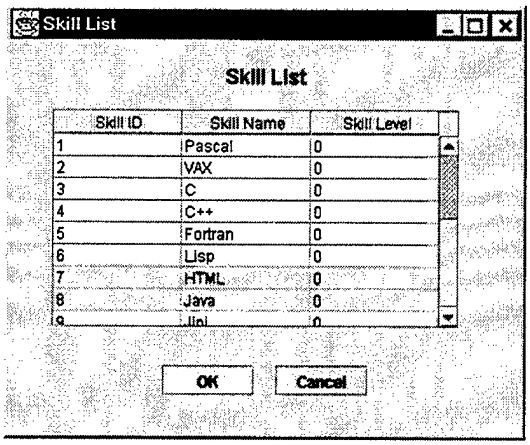


Figure 112: Skill list (2) of SPIDER – step content

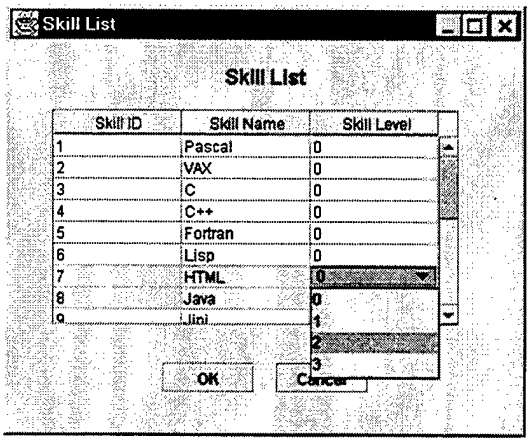


Figure 113: Skill list (3) of SPIDER – step content

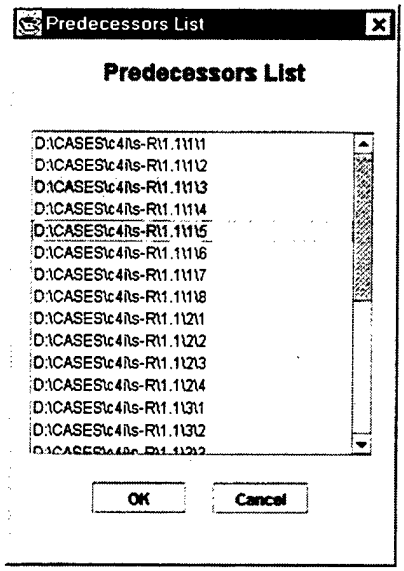


Figure 114: Predecessor list of SPIDER – step content

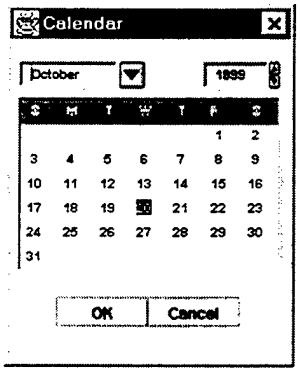


Figure 115: Date chooser of SPIDER – step content

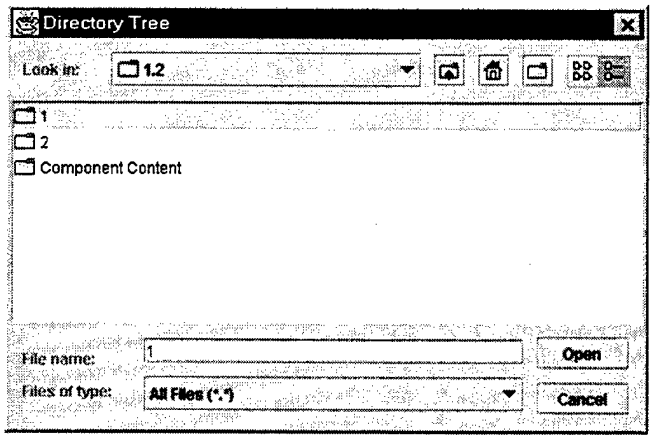


Figure 116: File chooser (1) of SPIDER – trace

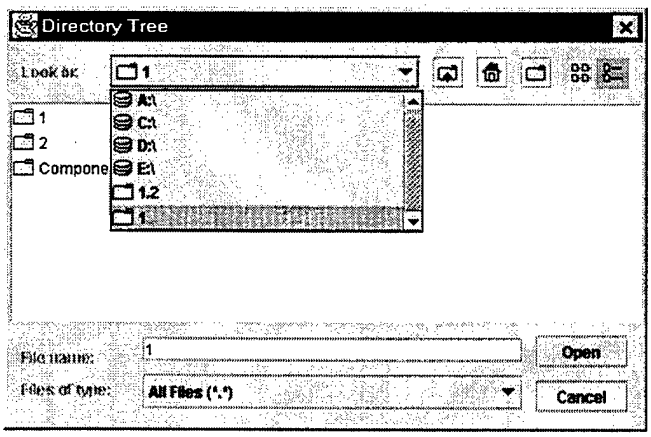


Figure 117: File chooser (2) of SPIDER – trace

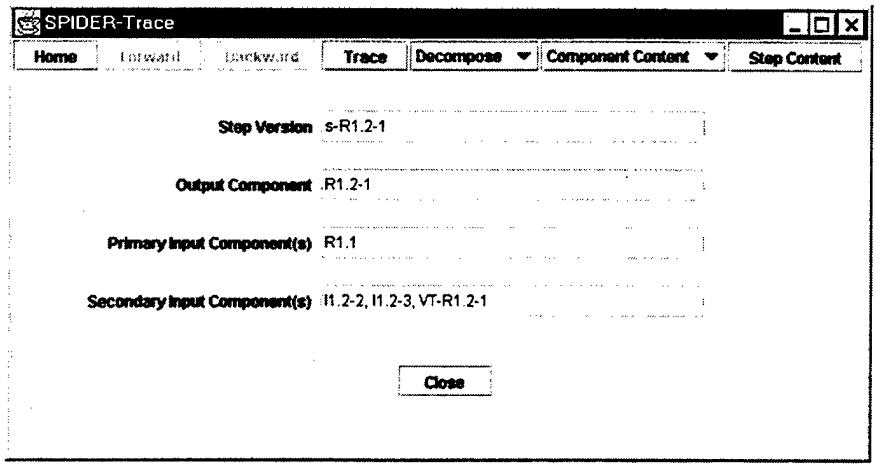


Figure 118: SPIDER – trace

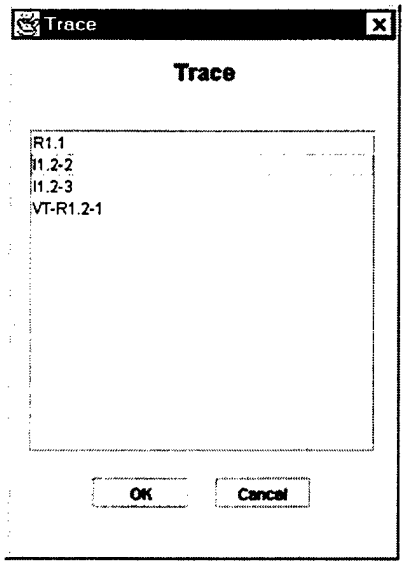


Figure 119: Trace a component in the SPIDER – trace (1)

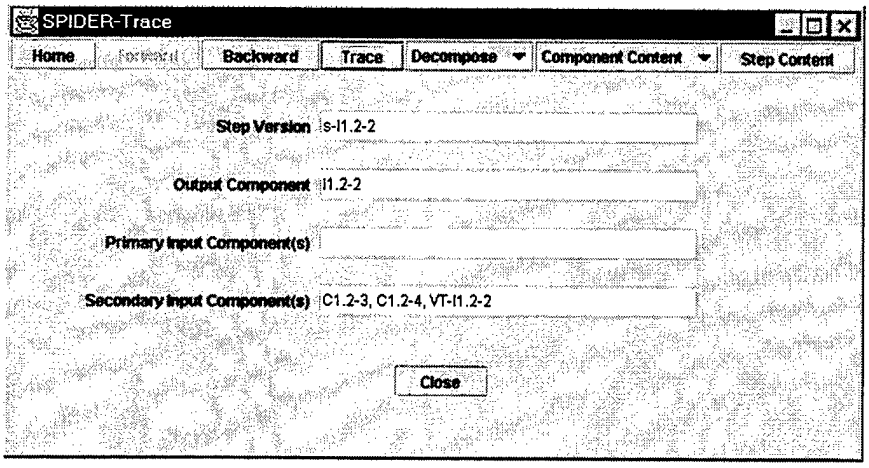


Figure 120: Trace a component in the SPIDER – trace (2)

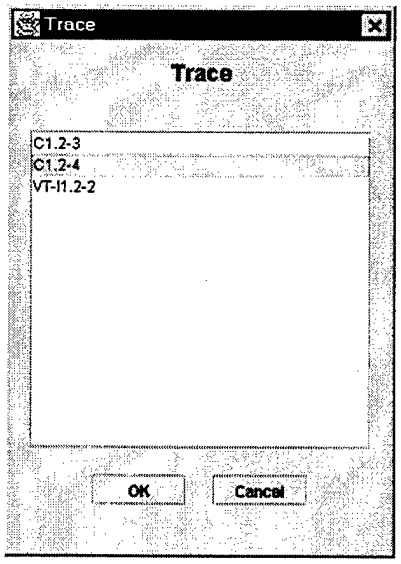


Figure 121: Trace a component in the SPIDER – trace (3)

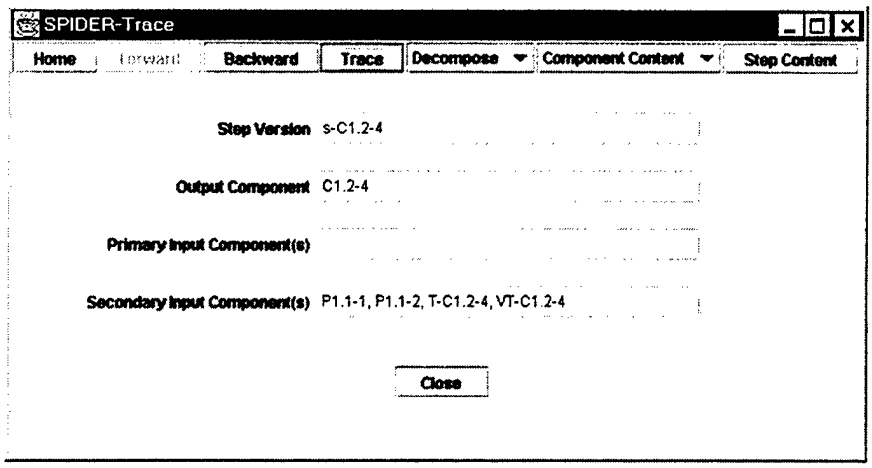


Figure 122: Trace a component in the SPIDER – trace (4)

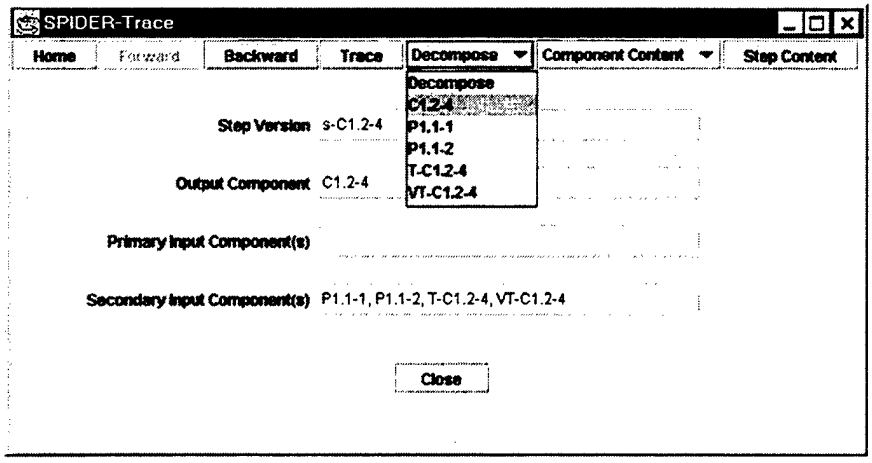


Figure 123: Decompose a component in the SPIDER – trace (1)

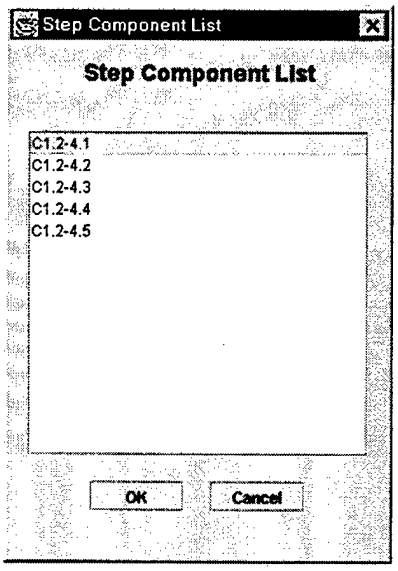


Figure 124: Decompose a component in the SPIDER – trace (2)

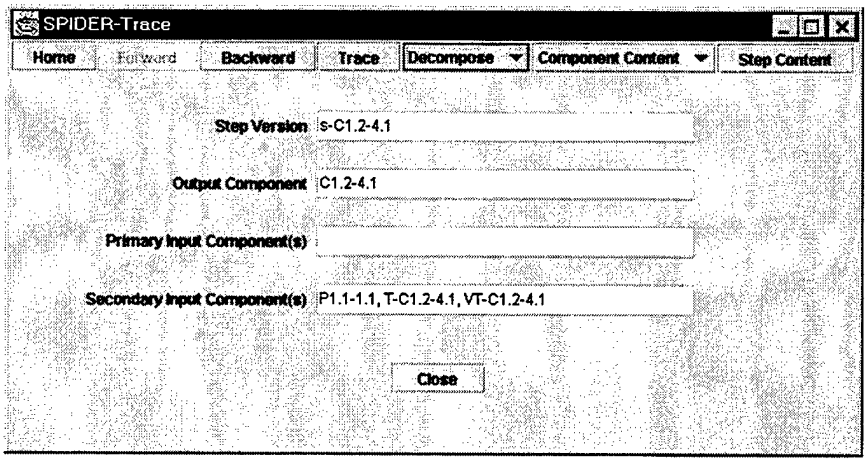


Figure 125: Decompose a component in the SPIDER – trace (3)

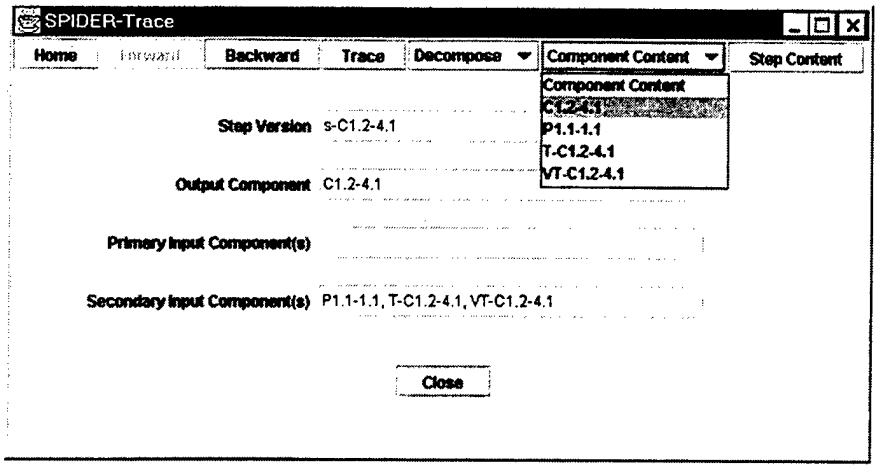


Figure 126: Review component content in the SPIDER – trace (1)

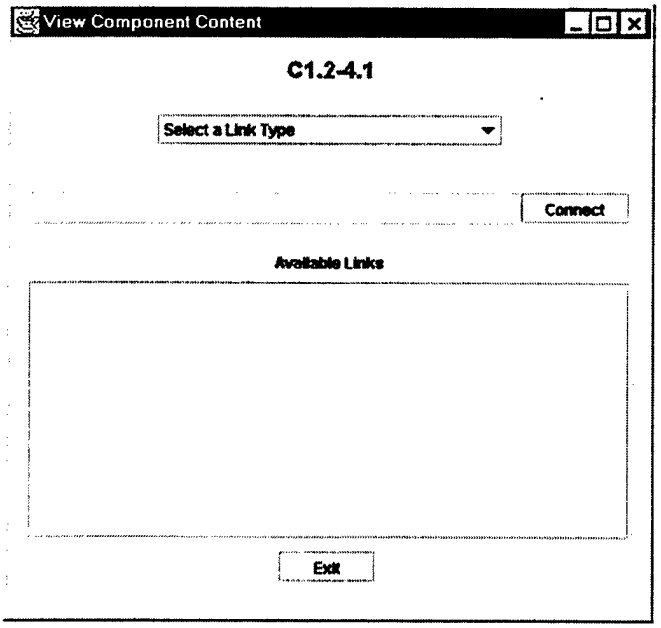


Figure 127: Review component content in the SPIDER – trace (2)

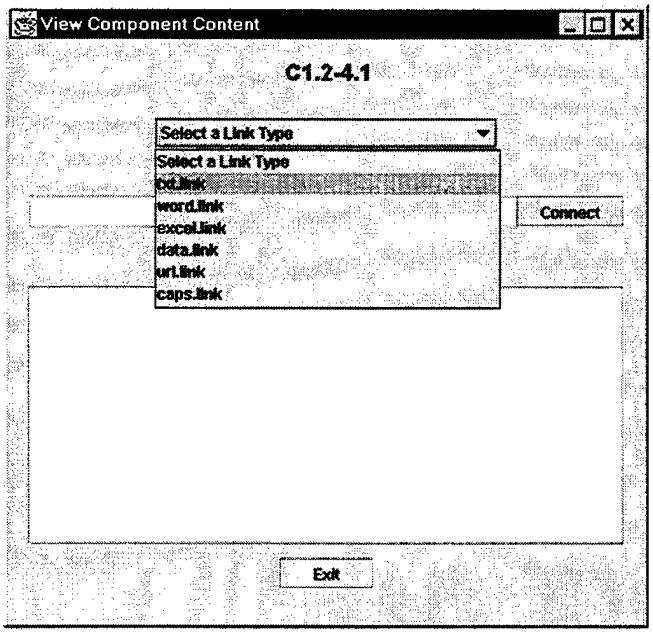


Figure 128: Review component content in the SPIDER – trace (3)

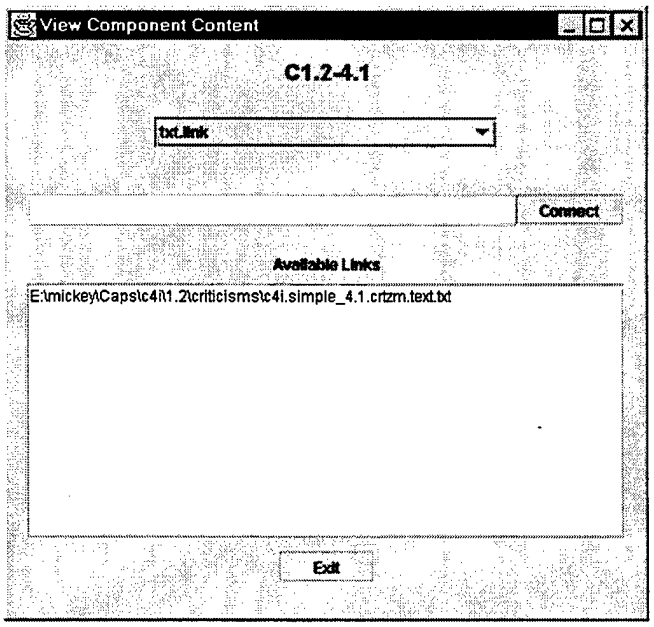


Figure 129: Review component content in the SPIDER – trace (4)

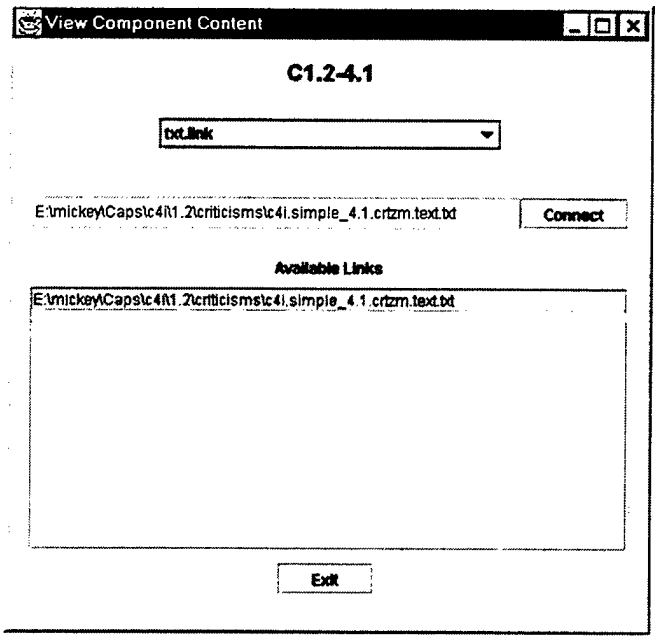


Figure 130: Review component content in the SPIDER – trace (5)

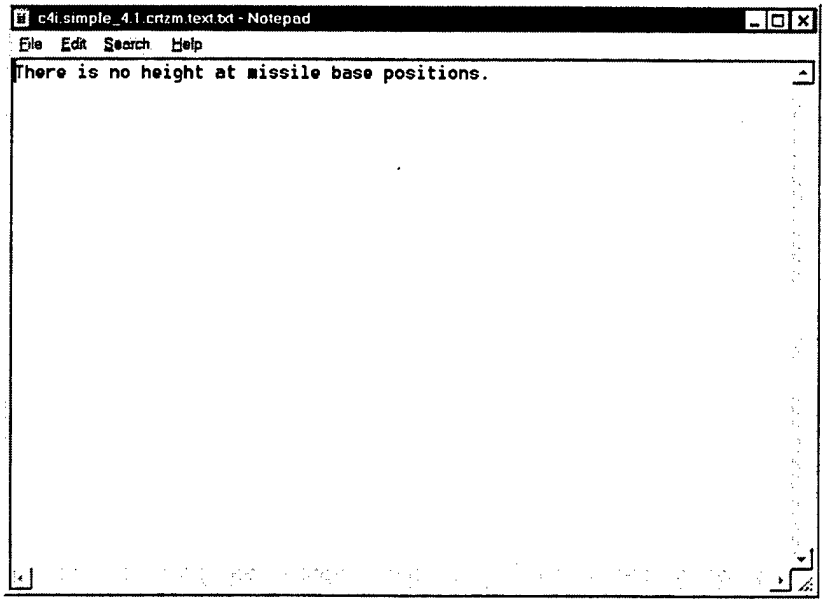


Figure 131: Review component content in the SPIDER – trace via notepad

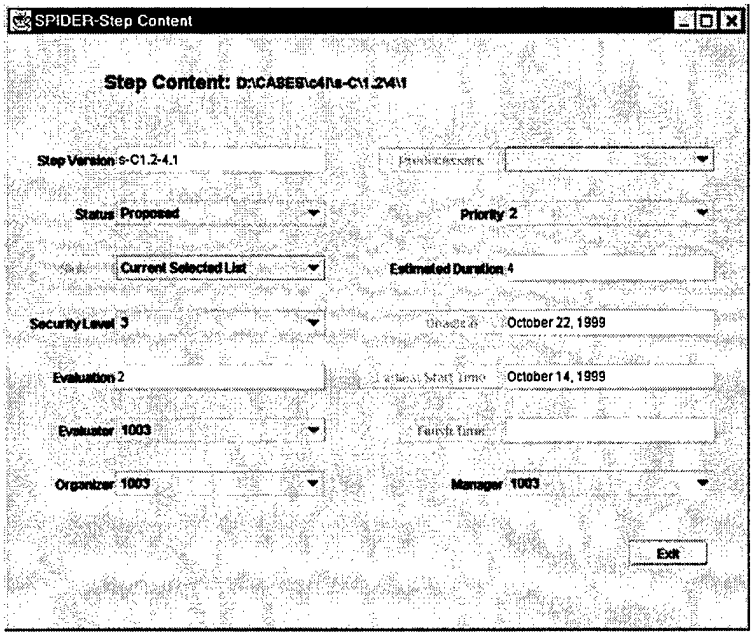


Figure 132: Review step content in the SPIDER – trace

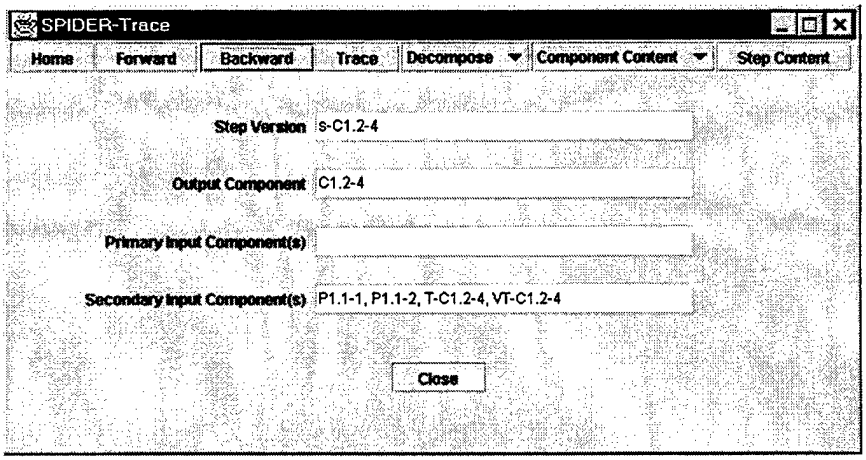


Figure 133: Trace a component in the SPIDER – trace by backward button

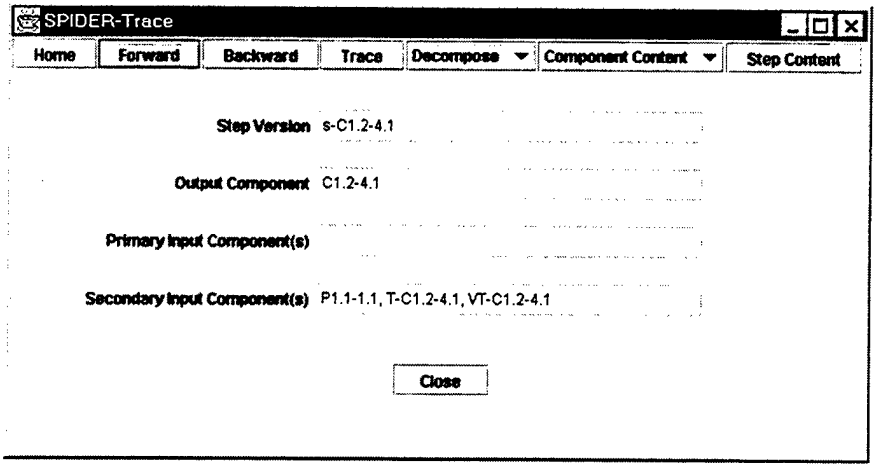


Figure 134: Trace a component in the SPIDER – trace by forward button

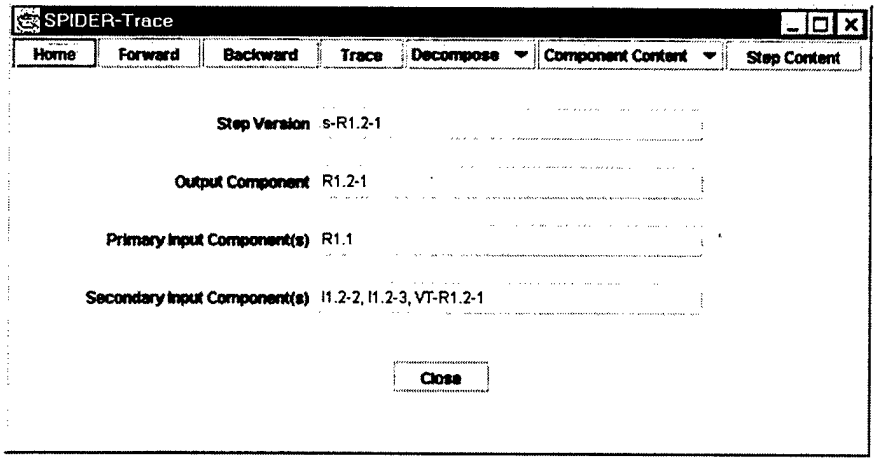


Figure 135: Trace a component in the SPIDER – trace by home button

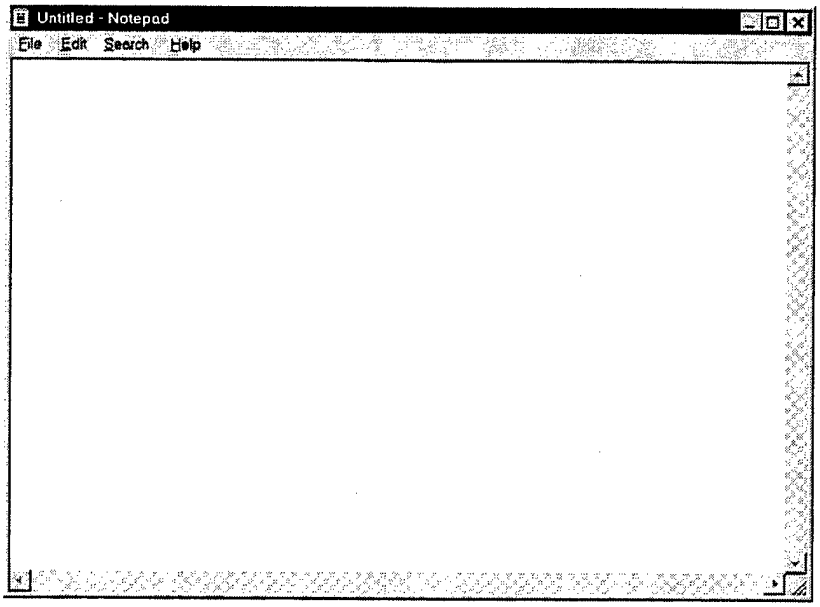


Figure 136: Tools – notepad

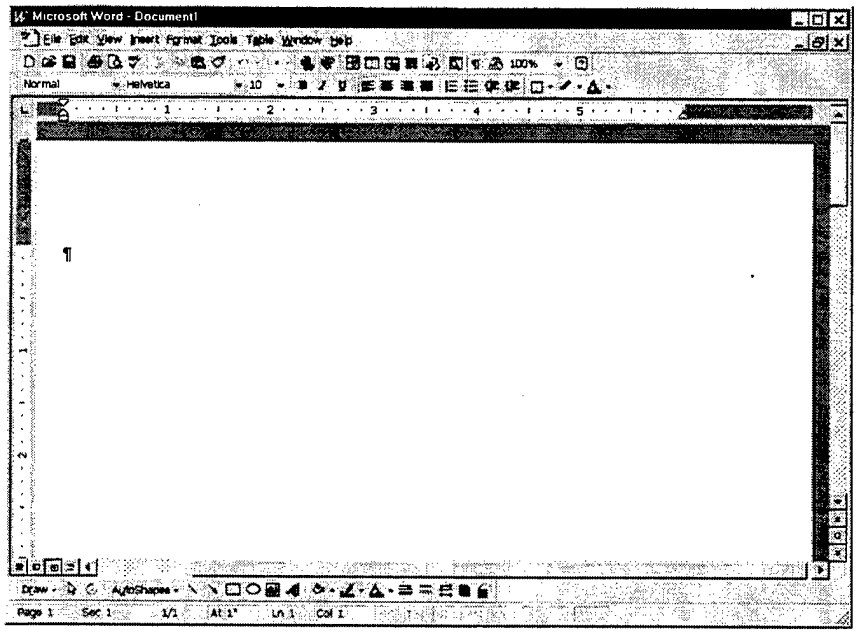


Figure 137: Tools – MS Word

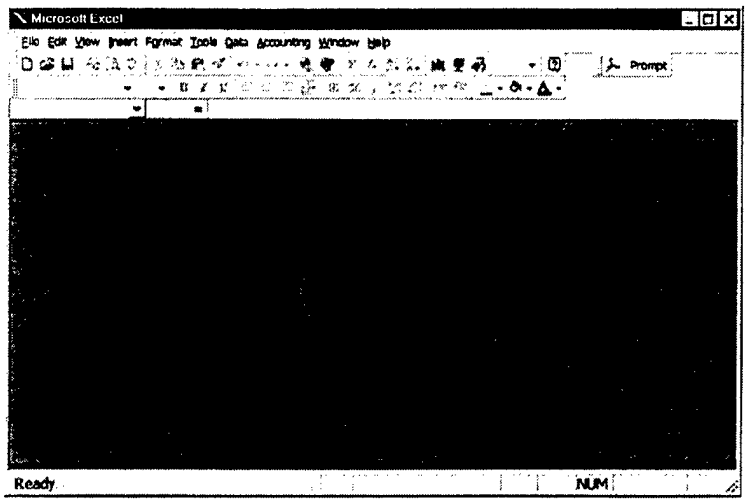


Figure 138: Tools – MS Excel

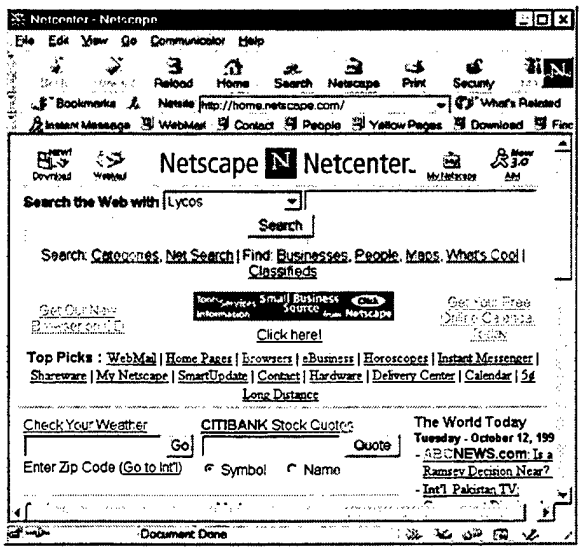


Figure 139: Tools – Netscape

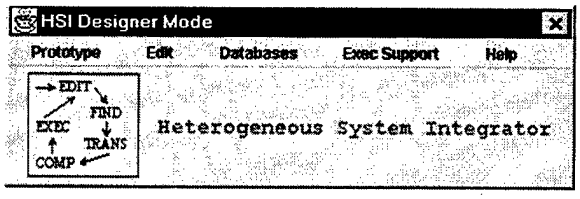


Figure 140: Tools – CAPS

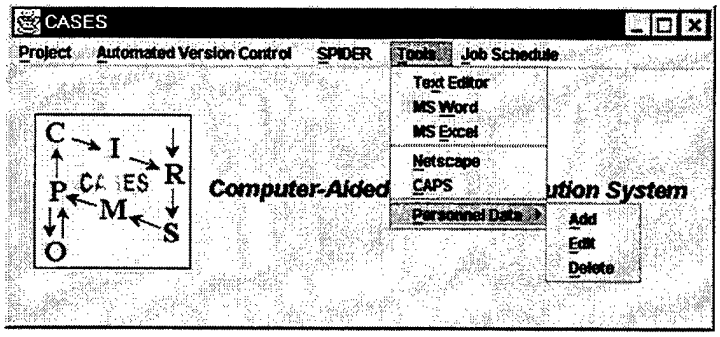


Figure 141: Tools – personnel data

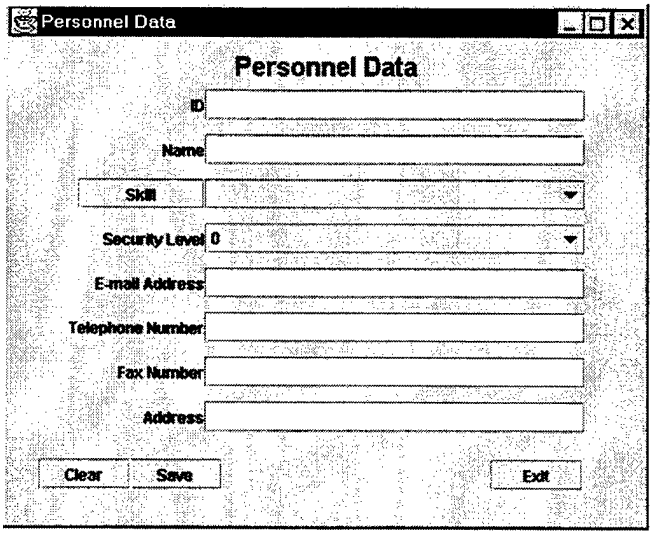


Figure 142: Add personnel data in the Tools – personnel data (1)

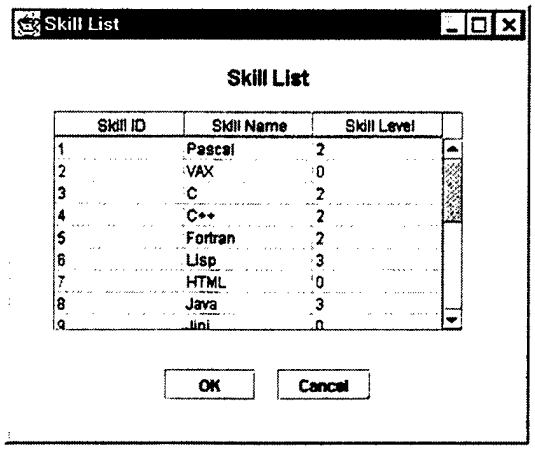


Figure 143: Add personnel data in the Tools – personnel data (2)

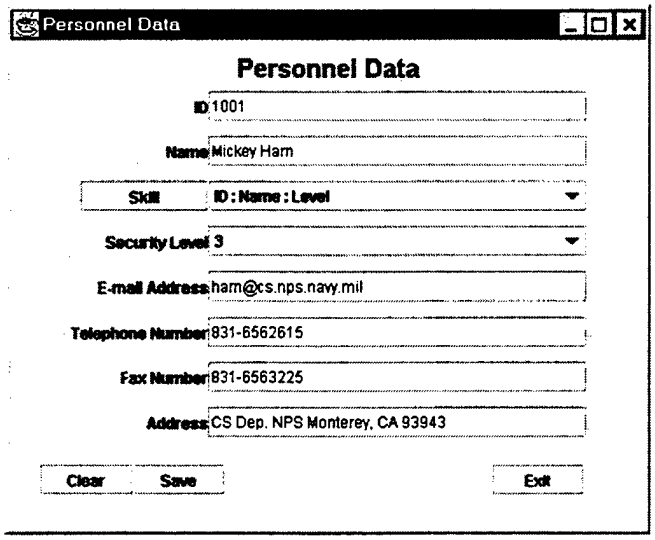


Figure 144: Add personnel data in the Tools – personnel data (3)

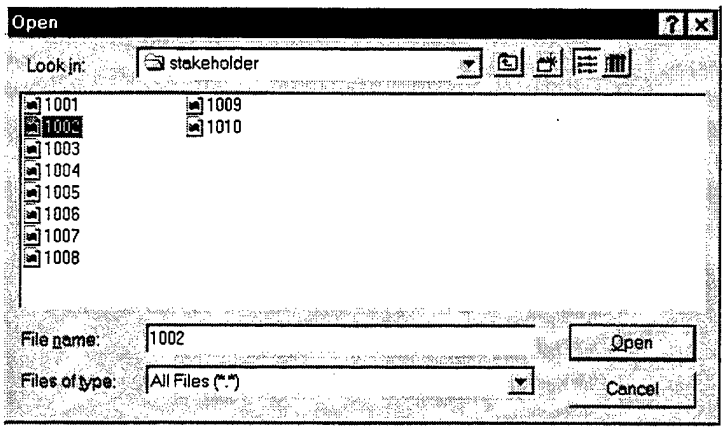


Figure 145: File chooser for editing personnel data in the Tools – personnel data

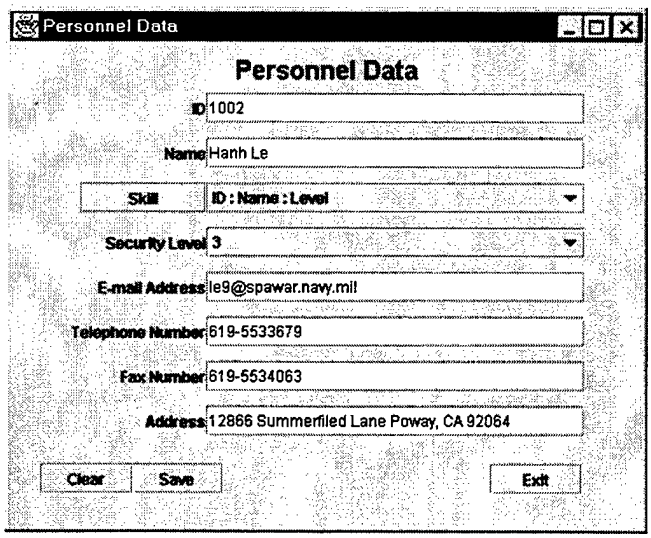


Figure 146: Edit personnel data in the Tools – personnel data

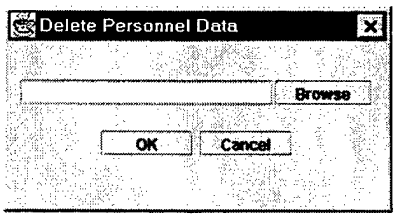


Figure 147: Delete personnel data in the Tools – personnel data (1)

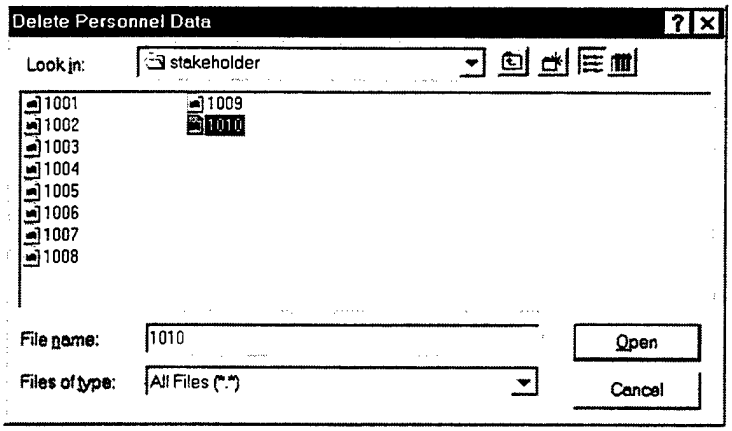


Figure 148: Delete personnel data in the Tools – personnel data (2)

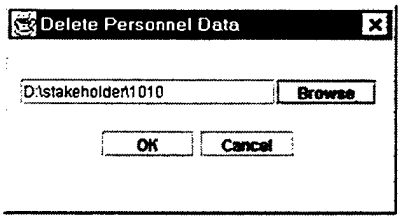


Figure 149: Delete personnel data in the Tools – personnel data (3)

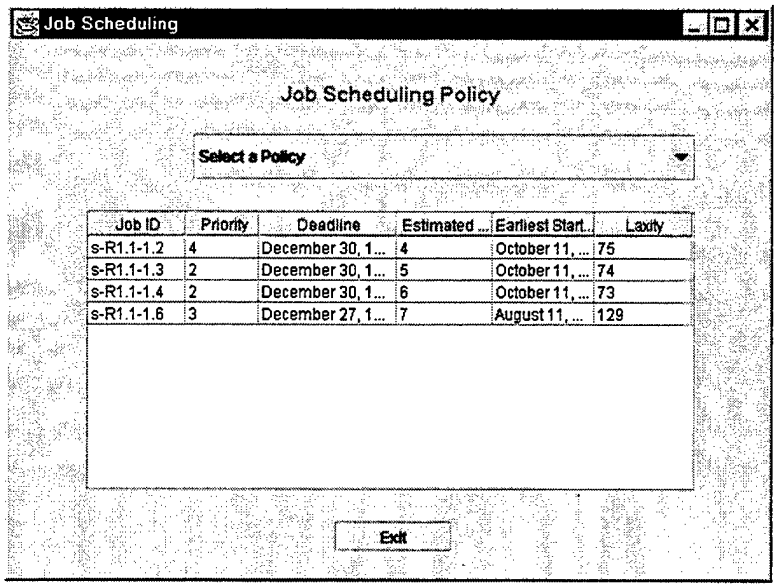


Figure 150: Job schedule – scheduling (1)

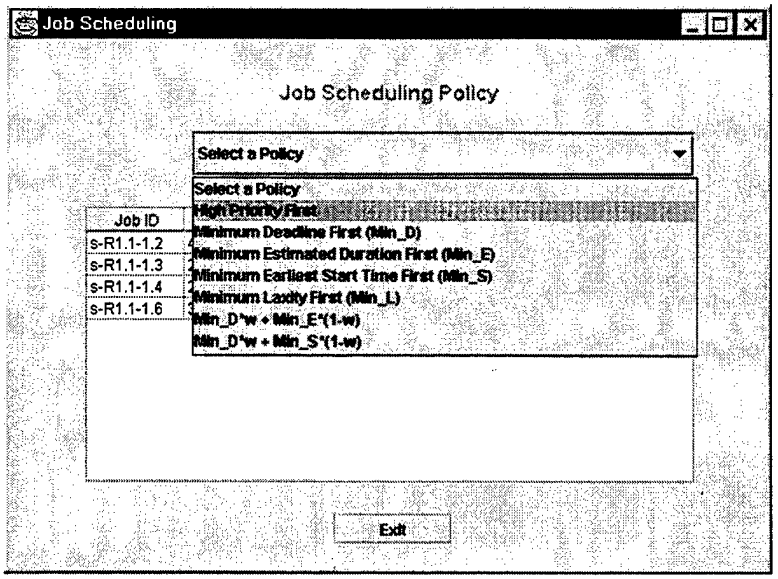


Figure 151: Job schedule – scheduling (2)

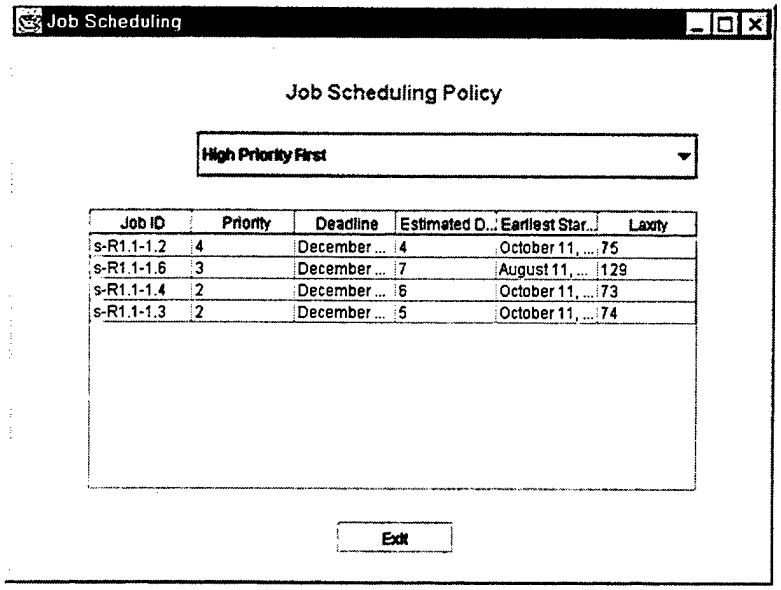


Figure 152: Job schedule – scheduling (3)

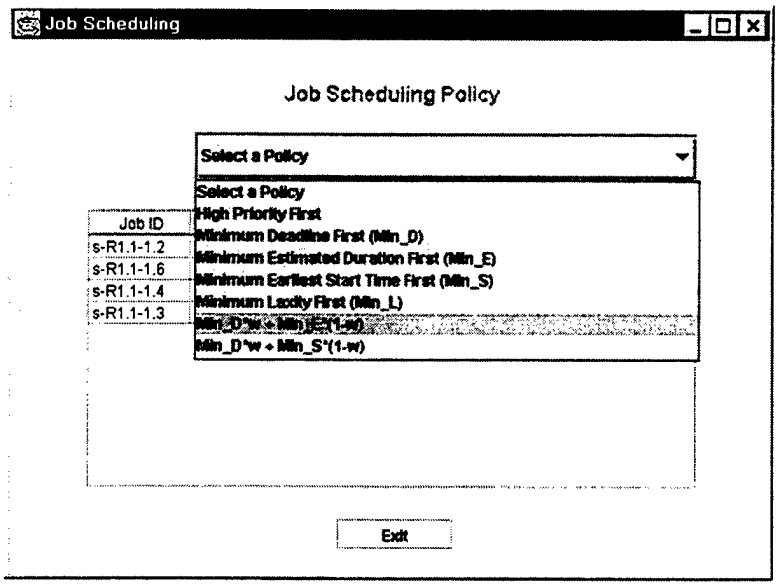


Figure 153: Job schedule – scheduling (4)

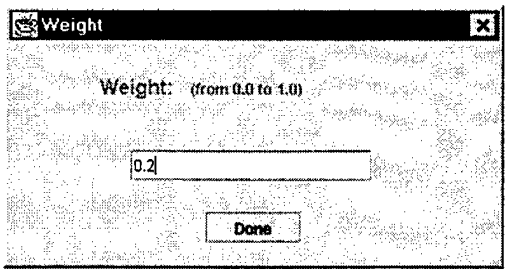
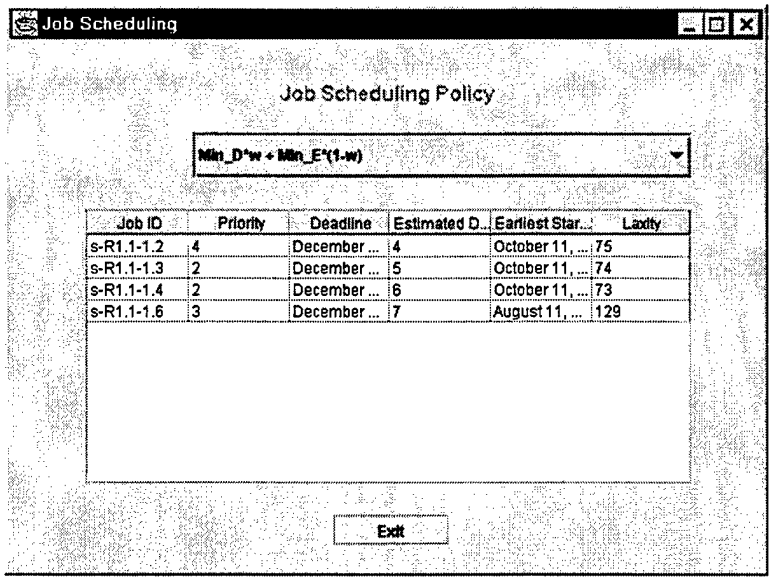


Figure 154: Job schedule – scheduling (5)



Job ID	Priority	Deadline	Estimated D...	Earliest Star...	Laxity
s-R1.1-1.2	4	December ...	4	October 11, ...	75
s-R1.1-1.3	2	December ...	5	October 11, ...	74
s-R1.1-1.4	2	December ...	6	October 11, ...	73
s-R1.1-1.6	3	December ...	7	August 11, ...	129

Figure 155: Job schedule – scheduling (6)

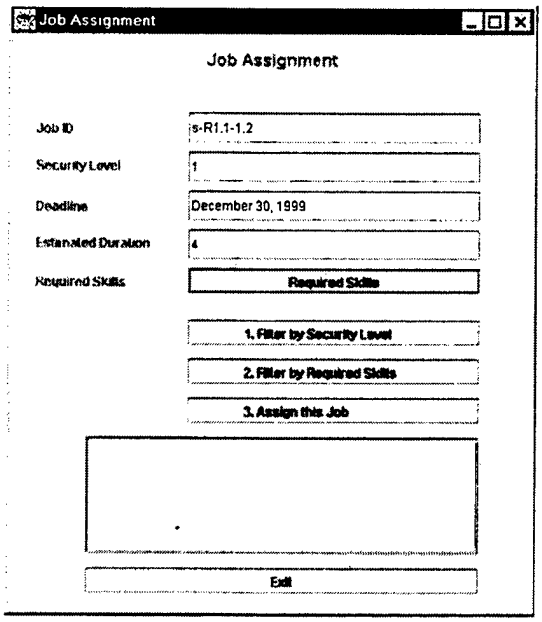


Figure 156: Job schedule – assignment (1)

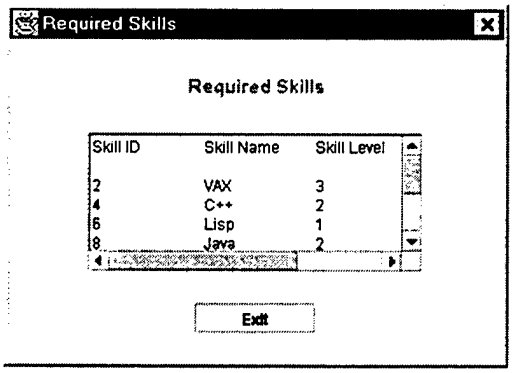


Figure 157: Job schedule – assignment (2)

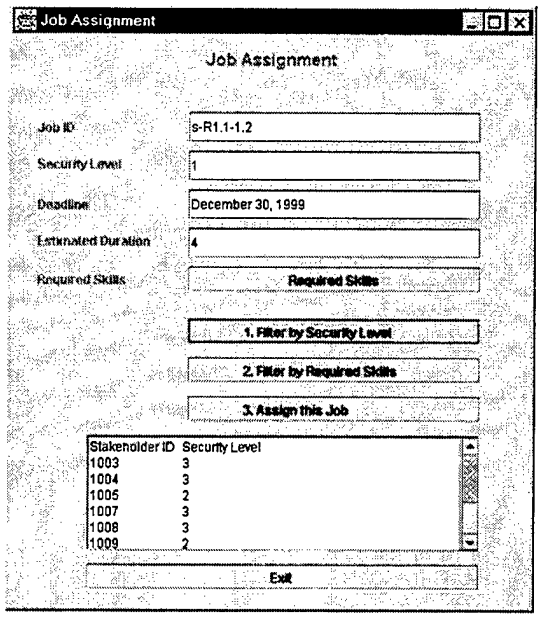


Figure 158: Job schedule – assignment (3)

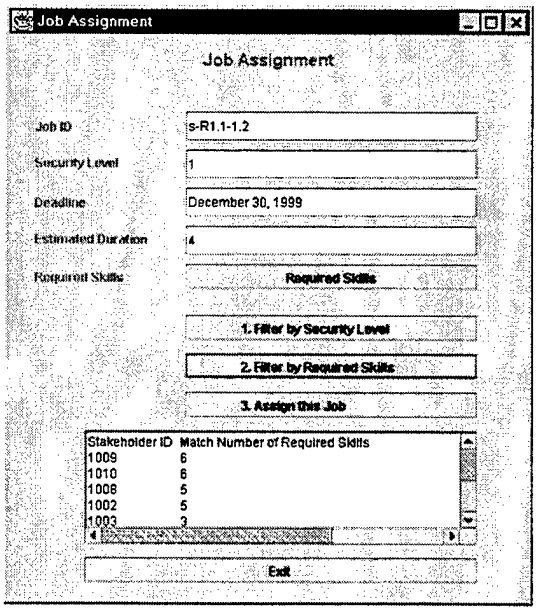


Figure 159: Job schedule – assignment (4)

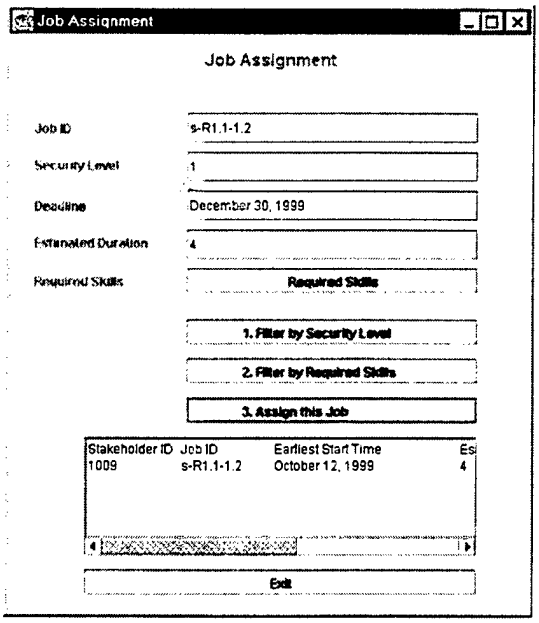


Figure 160: Job schedule – assignment (5)

APPENDIX C. SPIDERS OF C4I/MD SYSTEMS

A. Requirement analysis step: *s-R1.1*

(R1.1-1 ← s-R1.1-1 (VT-R1.1-1))

(R1.1-1.1 ← s-R1.1-1.1 (VT-R1.1-1.1))

(R1.1-1.2 ← s-R1.1-1.2 (VT-R1.1-1.2))

(R1.1-1.3 ← s-R1.1-1.3 (VT-R1.1-1.3))

(R1.1-1.4 ← s-R1.1-1.4 (VT-R1.1-1.4))

(R1.1-1.5 ← s-R1.1-1.5 (VT-R1.1-1.5))

(R1.1-1.6 ← s-R1.1-1.6 (VT-R1.1-1.6))

(R1.1-1.7 ← s-R1.1-1.7 (VT-R1.1-1.7))

(R1.1-1.8 ← s-R1.1-1.8 (VT-R1.1-1.8))

(R1.1-2 ← s-R1.1-2 (VT-R1.1-2))

(R1.1-2.1 ← s-R1.1-2.1 (VT-R1.1-2.1))

(R1.1-2.2 ← s-R1.1-2.2 (VT-R1.1-2.2))

(R1.1-2.3 ← s-R1.1-2.3 (VT-R1.1-2.3))

(R1.1-2.4 ← s-R1.1-2.4 (VT-R1.1-2.4))

(R1.1-3 ← s-R1.1-3 (VT-R1.1-3))

(R1.1-3.1 ← s-R1.1-3.1 (VT-R1.1-3.1))

(R1.1-3.2 ← s-R1.1-3.2 (VT-R1.1-3.2))

(R1.1-3.3 ← s-R1.1-3.3 (VT-R1.1-3.3))

(R1.1-3.4 ← s-R1.1-3.4 (VT-R1.1-3.4))

(R1.1-3.5 ← s-R1.1-3.5 (VT-R1.1-3.5))

B. Specification design step: *s-S1.1*

(S1.1-1 ← s-S1.1-1 (R1.1-1, R1.1-2, VT-S1.1-1))

(S1.1-1.1 ← s-S1.1-1.1 (R1.1-1.1, VT-S1.1-1.1))

(S1.1-1.2 ← s-S1.1-1.2 (R1.1-1.2, VT-S1.1-1.2))

(S1.1-1.3 ← s-S1.1-1.3 (R1.1-1.3, VT-S1.1-1.3))

(S1.1-1.4 ← s-S1.1-1.4 (R1.1-1.4, VT-S1.1-1.4))

(S1.1-1.5 ← s-S1.1-1.5 (R1.1-1.5, VT-S1.1-1.5))

(S1.1-1.6 ← s-S1.1-1.6 (R1.1-1.6, VT-S1.1-1.6))

(S1.1-1.7 ← s-S1.1-1.7 (R1.1-1.7, VT-S1.1-1.7))

(S1.1-1.8 ← s-S1.1-1.8 (R1.1-1.8, VT-S1.1-1.8))

(S1.1-1.9 ← s-S1.1-1.9 (R1.1-2.1, VT-S1.1-1.9))

(S1.1-1.10 ← s-S1.1-1.10 (R1.1-2.2, VT-S1.1-1.10))

(S1.1-1.11 ← s-S1.1-1.11 (R1.1-2.3, VT-S1.1-1.11))

(S1.1-1.12 ← s-S1.1-1.12 (R1.1-2.4, VT-S1.1-1.12))

(S1.1-1.13 ← s-S1.1-1.13 (VT-S1.1-1.13))

(S1.1-2 ← s-S1.1-2 (R1.1-3, VT-S1.1-2))

(S1.1-2.1 ← s-S1.1-2.1 (R1.1-3.1, VT-S1.1-2.1))

(S1.1-2.2 ← s-S1.1-2.2 (R1.1-3.2, VT-S1.1-2.2))

(S1.1-2.3 ← s-S1.1-2.3 (R1.1-3.3, VT-S1.1-2.3))

(S1.1-2.4 ← s-S1.1-2.4 (R1.1-3.4, VT-S1.1-2.4))

(S1.1-2.5 ← s-S1.1-2.5 (R1.1-3.5, VT-S1.1-2.5))

C. Module implementation step: *s-M1.1*

(M1.1-1 ← s-M1.1-1 (S1.1-1, VT-M1.1-1))

(M1.1-1.1 ← s-M1.1-1.1 (S1.1-1.1, VT-M1.1-1.1))

(M1.1-1.2 ← s-M1.1-1.2 (S1.1-1.2, VT-M1.1-1.2))

(M1.1-1.3 ← s-M1.1-1.3 (S1.1-1.3, VT-M1.1-1.3))

(M1.1-1.4 ← s-M1.1-1.4 (S1.1-1.4, VT-M1.1-1.4))

(M1.1-1.5 ← s-M1.1-1.5 (S1.1-1.5, VT-M1.1-1.5))

(M1.1-1.6 ← s-M1.1-1.6 (S1.1-1.6, VT-M1.1-1.6))

(M1.1-1.7 ← s-M1.1-1.7 (S1.1-1.7, VT-M1.1-1.7))

(M1.1-1.8 ← s-M1.1-1.8 (S1.1-1.8, VT-M1.1-1.8))

(M1.1-1.9 ← s-M1.1-1.9 (S1.1-1.9, VT-M1.1-1.9))

(M1.1-1.10 ← s-M1.1-1.10 (S1.1-1.10, VT-M1.1-1.10))

(M1.1-1.11 ← s-M1.1-1.11 (S1.1-1.11, VT-M1.1-1.11))

(M1.1-1.12 ← s-M1.1-1.12 (S1.1-1.12, VT-M1.1-1.12))

(M1.1-1.13 ← s-M1.1-1.13 (VT-M1.1-1.13))

(M1.1-2 ← s-M1.1-2 (S1.1-2, VT-M1.1-2))

(M1.1-2.1 ← s-M1.1-2.1 (S1.1-2.1, VT-M1.1-2.1))

(M1.1-2.2 ← s-M1.1-2.2 (S1.1-2.2, VT-M1.1-2.2))

(M1.1-2.3 ← s-M1.1-2.3 (S1.1-2.3, VT-M1.1-2.3))

(M1.1-2.4 ← s-M1.1-2.4 (S1.1-2.4, VT-M1.1-2.4))

(M1.1-2.5 ← s-M1.1-2.5 (S1.1-2.5, VT-M1.1-2.5))

D. Program integration step: *s-P1.1*

(P1.1-1 ← s-P1.1-1 (M1.1-1, VT-P1.1-1))

(P1.1-1.1 ← s-P1.1-1.1 (M1.1-1.1, VT-P1.1-1.1))

(P1.1-1.2 ← s-P1.1-1.2 (M1.1-1.2, VT-P1.1-1.2))

(P1.1-1.3 ← s-P1.1-1.3 (M1.1-1.3, VT-P1.1-1.3))

(P1.1-1.4 ← s-P1.1-1.4 (M1.1-1.4, VT-P1.1-1.4))

(P1.1-1.5 ← s-P1.1-1.5 (M1.1-1.5, VT-P1.1-1.5))

(P1.1-1.6 ← s-P1.1-1.6 (M1.1-1.6, VT-P1.1-1.6))

(P1.1-1.7 ← s-P1.1-1.7 (M1.1-1.7, VT-P1.1-1.7))

(P1.1-1.8 ← s-P1.1-1.8 (M1.1-1.8, VT-P1.1-1.8))

(P1.1-2 ← s-P1.1-2 (M1.1-1, VT-P1.1-2))

(P1.1-2.1 ← s-P1.1-2.1 (M1.1-1.9, VT-P1.1-2.1))

(P1.1-2.2 ← s-P1.1-2.2 (M1.1-1.10, VT-P1.1-2.2))

(P1.1-2.3 ← s-P1.1-2.3 (M1.1-1.11, VT-P1.1-2.3))

(P1.1-2.4 ← s-P1.1-2.4 (M1.1-1.12, VT-P1.1-2.4))

(P1.1-3 ← s-P1.1-3 (M1.1-1, VT-P1.1-3))

(P1.1-3.1 ← s-P1.1-3.1 (M1.1-1.13, VT-P1.1-2.1))

(P1.1-4 ← s-P1.1-4 (M1.1-2, VT-P1.1-4))

(P1.1-4.1 ← s-P1.1-4.1 (M1.1-2.1, VT-P1.1-4.1))

(P1.1-4.2 ← s-P1.1-4.2 (M1.1-2.2, VT-P1.1-4.2))

(P1.1-4.3 ← s-P1.1-4.3 (M1.1-2.3, VT-P1.1-4.3))

(P1.1-4.4 ← s-P1.1-4.4 (M1.1-2.4, VT-P1.1-4.4))

(P1.1-4.5 ← s-P1.1-4.5 (M1.1-2.5, VT-P1.1-4.5))

E. Software prototype demo step: *s-C1.2*

(C1.2-1 ← s-C1.2-1 (P1.1-1, P1.1-2, T-C1.2-1, VT-C1.2-1))

(C1.2-1.1 ← s-C1.2-1.1 (P1.1-1.2, P1.1-2.2, T-C1.2-1.1, VT-C1.2-1.1))

(C1.2-1.2 ← s-C1.2-1.2 (P1.1-1.6, T-C1.2-1.2, VT-C1.2-1.2))

(C1.2-2 ← s-C1.2-2 (P1.1-1, P1.1-2, T-C1.2-2, VT-C1.2-2))

(C1.2-2.1 ← s-C1.2-2.1 (P1.1-1.2, P1.1-2.2, T-C1.2-2.1, VT-C1.2-2.1))

(C1.2-2.2 ← s-C1.2-2.2 (P1.1-1.6, T-C1.2-2.2, VT-C1.2-2.2))

(C1.2-3 ← s-C1.2-3 (P1.1-1, P1.1-2, T-C1.2-3, VT-C1.2-3))

(C1.2-3.1 ← s-C1.2-3.1 (P1.1-2.2, P1.1-2.3, P1.1-2.4, P1.1-1.4, P1.1-1.5, T-C1.2-3.1, VT-C1.2-3.1))

(C1.2-3.2 ← s-C1.2-3.2 (P1.1-1.4, P1.1-1.5, P1.1-1.8, P1.1-1.2, P1.1-1.6, T-C1.2-3.2, VT-C1.2-3.2))

(C1.2-3.3 ← s-C1.2-3.3 (P1.1-2.2, P1.1-1.4, T-C1.2-3.3, VT-C1.2-3.3))

(C1.2-4 ← s-C1.2-4 (P1.1-1, P1.1-2, T-C1.2-4, VT-C1.2-4))

(C1.2-4.1 ← s-C1.2-4.1 (P1.1-1.1, T-C1.2-4.1, VT-C1.2-4.1))

(C1.2-4.2 ← s-C1.2-4.2 (P1.1-1.6, T-C1.2-4.2, VT-C1.2-4.2))

(C1.2-4.3 ← s-C1.2-4.3 (P1.1-1.2, P1.1-1.1, P1.1-1.6, T-C1.2-4.3, VT-C1.2-4.3))

(C1.2-4.4 ← s-C1.2-4.4 (P1.1-2.2, P1.1-1.2, P1.1-2.3, P1.1-1.4, P1.1-1.5, T-C1.2-4.4, VT-C1.2-4.4))

(C1.2-4.5 ← s-C1.2-4.5 (P1.1-2.2, P1.1-1.2, P1.1-2.3, P1.1-1.5, P1.1-1.6, P1.1-1.3, T-C1.2-4.5, VT-C1.2-4.5))

(C1.2-5 ← s-C1.2-5 (P1.1-1, P1.1-2, T-C1.2-5, VT-C1.2-5))

(C1.2-5.1 ← s-C1.2-5.1 (P1.1-1.1, P1.1-2.1, T-C1.2-5.1, VT-C1.2-5.1))

(C1.2-5.2 ← s-C1.2-5.2 (P1.1-1.1, P1.1-2.1, T-C1.2-5.2, VT-C1.2-5.2))

(C1.2-6 ← s-C1.2-6 (P1.1-1, P1.1-2, T-C1.2-6, VT-C1.2-6))

(C1.2-6.1 ← s-C1.2-6.1 (P1.1-1.2, P1.1-2.2, P1.1-2.1, P1.1-1.1, P1.1-1.6, T-C1.2-6.1, VT-C1.2-6.1))

(C1.2-6.2 ← s-C1.2-6.2 (P1.1-1.2, P1.1-2.2, P1.1-2.3, P1.1-2.1, P1.1-1.1,

T-C1.2-6.2, VT-C1.2-6.2))

(C1.2-6.3 ← s-C1.2-6.3 (P1.1-2.1, P1.1-1.1, T-C1.2-6.3, VT-C1.2-6.3))

F. Issue analysis step: *s-I1.2*

(I1.2-1 ← s-I1.2-1 (C1.2-1, C1.2-2, VT-I1.2-1))

(I1.2-1.1 ← s-I1.2-1.1 (C1.2-1.1, C1.2-2.1, VT-I1.2-1.1))

(I1.2-1.2 ← s-I1.2-1.2 (C1.2-1.2, C1.2-2.2, VT-I1.2-1.2))

(I1.2-1.3 ← s-I1.2-1.3 (C1.2-4.1, VT-I1.2-1.3))

(I1.2-2 ← s-I1.2-2 (C1.2-3, C1.2-4, VT-I1.2-2))

(I1.2-2.1 ← s-I1.2-2.1 (C1.2-4.3, VT-I1.2-2.1))

(I1.2-2.2 ← s-I1.2-2.2 (C1.2-4.3, VT-I1.2-2.2))

(I1.2-2.3 ← s-I1.2-2.3 (C1.2-3.3, VT-I1.2-2.3))

(I1.2-3 ← s-I1.2-3 (C1.2-3, VT-I1.2-3))

(I1.2-3.1 ← s-I1.2-3.1 (C1.2-3.3, VT-I1.2-3.1))

(I1.2-3.2 ← s-I1.2-3.2 (C1.2-3.2, VT-I1.2-3.2))

(I1.2-4 ← s-I1.2-4 (C1.2-4, C1.2-5, VT-I1.2-4))

(I1.2-4.1 ← s-I1.2-4.1 (C1.2-4.4, VT-I1.2-4.1))

(I1.2-4.2 ← s-I1.2-4.2 (C1.2-5.1, VT-I1.2-4.2))

(I1.2-4.3 ← s-I1.2-4.3 (C1.2-5.2, VT-I1.2-4.3))

(I1.2-5 ← s-I1.2-5 (C1.2-5, C1.2-6, VT-I1.2-5))

(I1.2-5.1 ← s-I1.2-5.1 (C1.2-6.1, VT-I1.2-5.1))

(I1.2-5.2 ← s-I1.2-5.2 (C1.2-6.1, VT-I1.2-5.2))

(I1.2-5.3 ← s-I1.2-5.3 (C1.2-5.1, C1.2-5.2, VT-I1.2-5.3))

(I1.2-6 ← s-I1.2-6 (C1.2-5, C1.2-6, VT-I1.2-6))

(I1.2-6.1 ← s-I1.2-6.1 (C1.2-6.2, VT-I1.2-6.1))

(I1.2-6.2 ← s-I1.2-6.2 (C1.2-6.3, VT-I1.2-6.2))

(I1.2-6.3 ← s-I1.2-6.3 (C1.2-6.1, C1.2-6.2, C1.2-6.3, VT-I1.2-6.3))

(I1.2-6.4 ← s-I1.2-6.4 (C1.2-5.1, C1.2-5.2, VT-I1.2-6.4))

(I1.2-7 ← s-I1.2-7 (C1.1, VT-I1.2-7))

(I1.2-7.1 ← s-I1.2-7.1 (C1.2-1, C1.2-2, C1.2-3, VT-I1.2-7))

(I1.2-7.2 ← s-I1.2-7.2 (C1.1-1, C1.2-2, C1.2-3, C1.2-4, VT-I1.2-7.2))

G. Requirement analysis step: *s-R1.2*

(R1.2-1 ← s-R1.2-1 (R1.1, I1.2-2, I1.2-3, VT-R1.2-1))

(R1.2-1.1 ← s-R1.2-1.1 (R1.1-1, I1.2-2.1, I1.2-2.2, VT-R1.2-1.1))

(R1.2-1.2 ← s-R1.2-1.2 (R1.1-1, I1.2-2.3, I1.2-3.1, VT-R1.2-1.2))

(R1.2-2 ← s-R1.2-2 (R1.1-1, I1.2, VT-R1.2-2))

(R1.2-2.1 ← s-R1.2-2.1 (R1.1-1.1, I1.2-3, VT-R1.2-2.1))

(R1.2-2.2 ← s-R1.2-2.2 (R1.1-1.2, I1.2-3, VT-R1.2-2.2))

(R1.2-2.3 ← s-R1.2-2.3 (R1.1-1.4, I1.2-3, VT-R1.2-2.3))

(R1.2-2.4 ← s-R1.2-2.4 (R1.1-1.5, I1.2-3, VT-R1.2-2.4))

(R1.2-2.5 ← s-R1.2-2.5 (R1.1-1.6, I1.2-3, VT-R1.2-2.5))

(R1.2-2.6 ← s-R1.2-2.6 (R1.1-1.2, I1.2-3, VT-R1.2-2.6))

(R1.2-2.7 ← s-R1.2-2.7 (R1.1-1.6, VT-R1.2-2.7))

(R1.2-2.8 ← s-R1.2-2.8 (R1.1-1.2, VT-R1.2-2.8))

H. Specification design step: *s-S1.2*

(S1.2-1 ← s-S1.2-1 (S1.1-1, R1.2-2, VT-S1.2-1))

(S1.2-1.1 ← s-S1.2-1.1 (S1.1-1.1, R1.2-2.1, VT-S1.2-1.1))

(S1.2-1.2 ← s-S1.2-1.2 (S1.1-1.2, R1.2-2.2, VT-S1.2-1.2))

(S1.2-1.3 ← s-S1.2-1.3 (S1.1-1.3, VT-S1.2-1.3))

(S1.2-1.4 ← s-S1.2-1.4 (S1.1-1.4, R1.2-2.3, VT-S1.2-1.4))

(S1.2-1.5 ← s-S1.2-1.5 (S1.1-1.5, R1.2-2.4, VT-S1.2-1.5))

(S1.2-1.6 ← s-S1.2-1.6 (S1.1-1.6, R1.2-2.5, VT-S1.2-1.6))

(S1.2-1.7 ← s-S1.2-1.7 (S1.1-1.7, VT-S1.2-1.7))

(S1.2-1.8 ← s-S1.2-1.8 (S1.1-1.8, VT-S1.2-1.8))

(S1.2-1.9 ← s-S1.2-1.9 (S1.1-1.9, VT-S1.2-1.9))

(S1.2-1.10 ← s-S1.2-1.10 (S1.1-1.10, VT-S1.2-1.10))

(S1.2-1.11 ← s-S1.2-1.11 (S1.1-1.11, VT-S1.2-1.11))

(S1.2-1.12 ← s-S1.2-1.12 (S1.1-1.12, VT-S1.2-1.12))

(S1.2-1.13 ← s-S1.2-1.13 (S1.1-1.13, VT-S1.2-1.13))

(S1.2-1.14 ← s-S1.2-1.14 (R1.2-2.6, VT-S1.2-1.14))

(S1.2-1.15 ← s-S1.2-1.15 (R1.2-2.6, R1.2-2.7, R1.2-2.8, VT-S1.2-1.15))

(S1.2-1.16 ← s-S1.2-1.16 (R1.2-2.6, R1.2-2.7, R1.2-2.8, VT-S1.2-1.16))

(S1.2-2 ← s-S1.2-2 (S1.1-2, R1.2-2, VT-S1.2-2))

(S1.2-2.1 ← s-S1.2-2.1 (S1.1-2.1, VT-S1.2-2.1))

(S1.2-2.2 ← s-S1.2-2.2 (S1.1-2.2, R1.2-2.6, VT-S1.2-2.2))

(S1.2-2.3 ← s-S1.2-2.3 (S1.1-2.3, R1.2-2.6, VT-S1.2-2.3))

(S1.2-2.4 ← s-S1.2-2.4 (S1.1-2.4, VT-S1.2-2.4))

(S1.2-2.5 ← s-S1.2-2.5 (S1.1-2.5, VT-S1.2-2.5))

(S1.2-2.6 ← s-S1.2-2.6 (R1.2-2.6, VT-S1.2-2.6))

I. Module implementation step: *s-M1.2*

(M1.2-1 ← s-M1.2-1 (M1.1-1, S1.2-1, VT-M1.2-1))

(M1.2-1.1 ← s-M1.2-1.1 (M1.1-1.1, S1.2-1.1, VT-M1.2-1.1))

(M1.2-1.2 ← s-M1.2-1.2 (M1.1-1.2, S1.2-1.2, VT-M1.2-1.2))

(M1.2-1.3 ← s-M1.2-1.3 (M1.1-1.3, S1.2-1.3, VT-M1.2-1.3))

(M1.2-1.4 ← s-M1.2-1.4 (M1.1-1.4, S1.2-1.4, VT-M1.2-1.4))

(M1.2-1.5 ← s-M1.2-1.5 (M1.1-1.5, S1.2-1.5, VT-M1.2-1.5))

(M1.2-1.6 ← s-M1.2-1.6 (M1.1-1.6, S1.2-1.6, VT-M1.2-1.6))

(M1.2-1.7 ← s-M1.2-1.7 (M1.1-1.7, S1.2-1.7, VT-M1.2-1.7))

(M1.2-1.8 ← s-M1.2-1.8 (M1.1-1.8, S1.2-1.8, VT-M1.2-1.8))

(M1.2-1.9 ← s-M1.2-1.9 (M1.1-1.9, S1.2-1.9, VT-M1.2-1.9))

(M1.2-1.10 ← s-M1.2-1.10 (M1.1-1.10, S1.2-1.10, VT-M1.2-1.10))

(M1.2-1.11 ← s-M1.2-1.11 (M1.1-1.11, S1.2-1.11, VT-M1.2-1.11))

(M1.2-1.12 ← s-M1.2-1.12 (M1.1-1.12, S1.2-1.12, VT-M1.2-1.12))

(M1.2-1.13 ← s-M1.2-1.13 (M1.1-1.13, S1.2-1.13, VT-M1.2-1.13))

(M1.2-1.14 ← s-M1.2-1.14 (S1.2-1.14, VT-M1.2-1.14))

(M1.2-1.15 ← s-M1.2-1.15 (S1.2-1.15, VT-M1.2-1.15))

(M1.2-1.16 ← s-M1.2-1.16 (S1.2-1.16, VT-M1.2-1.16))

(M1.2-2 ← s-M1.2-2 (M1.1-2, S1.2-2, VT-M1.2-2))

(M1.2-2.1 ← s-M1.2-2.1 (M1.1-2.1, S1.2-2.1, VT-M1.2-2.1))

(M1.2-2.2 ← s-M1.2-2.2 (M1.1-2.2, S1.2-2.2, VT-M1.2-2.2))

(M1.2-2.3 ← s-M1.2-2.3 (M1.1-2.3, S1.2-2.3, VT-M1.2-2.3))

(M1.2-2.4 ← s-M1.2-2.4 (M1.1-2.4, S1.2-2.4, VT-M1.2-2.4))

(M1.2-2.5 ← s-M1.2-2.5 (M1.1-2.5, S1.2-2.5, VT-M1.2-2.5))

(M1.2-2.6 ← s-M1.2-2.6 (S1.2-2.6, VT-M1.2-2.6))

J. Program integration step: *s-P1.2*

(P1.2-1 ← s-P1.2-1 (P1.1-1, M1.2-1, VT-P1.2-1))

(P1.2-1.1 ← s-P1.2-1.1 (P1.1-1.1, M1.2-1.1, VT-P1.2-1.1))

(P1.2-1.2 ← s-P1.2-1.2 (P1.1-1.2, M1.2-1.2, VT-P1.2-1.2))

(P1.2-1.3 ← s-P1.2-1.3 (P1.1-1.3, M1.2-1.3, VT-P1.2-1.3))

(P1.2-1.4 ← s-P1.2-1.4 (P1.1-1.4, M1.2-1.4, VT-P1.2-1.4))

(P1.2-1.5 ← s-P1.2-1.5 (P1.1-1.5, M1.2-1.5, VT-P1.2-1.5))

(P1.2-1.6 ← s-P1.2-1.6 (P1.1-1.6, M1.2-1.6, VT-P1.2-1.6))

(P1.2-1.7 ← s-P1.2-1.7 (P1.1-1.7, M1.2-1.7, VT-P1.2-1.7))

(P1.2-1.8 ← s-P1.2-1.8 (P1.1-1.8, M1.2-1.8, VT-P1.2-1.8))

(P1.2-1.9 ← s-P1.2-1.9 (M1.2-1.14, VT-P1.2-1.9))

(P1.2-1.10 ← s-P1.2-1.10 (M1.2-1.15, VT-P1.2-1.10))

(P1.2-1.11 ← s-P1.2-1.11 (M1.2-1.16, VT-P1.2-1.11))

(P1.2-2 ← s-P1.2-2 (P1.1-2, M1.2-1, VT-P1.2-2))

(P1.2-2.1 ← s-P1.2-2.1 (P1.1-2.1, M1.2-1.9, VT-P1.2-2.1))

(P1.2-2.2 ← s-P1.2-2.2 (P1.1-2.2, M1.2-1.10, VT-P1.2-2.2))

(P1.2-2.3 ← s-P1.2-2.3 (P1.1-2.3, M1.2-1.11, VT-P1.2-2.3))

(P1.2-2.4 ← s-P1.2-2.4 (P1.1-2.4, M1.2-1.12, VT-P1.2-2.4))

(P1.2-3 ← s-P1.2-3 (P1.1-3, M1.2-1, VT-P1.2-2))

(P1.2-3.1 ← s-P1.2-3.1 (P1.1-3.1, M1.2-1.13, VT-P1.2-2.4))

(P1.2-4 ← s-P1.2-4 (P1.1-4, M1.2-2, VT-P1.2-4))

(P1.2-4.1 ← s-P1.2-4.1 (P1.1-4.1, M1.2-2.1, VT-P1.2-4.1))

(P1.2-4.2 ← s-P1.2-4.2 (P1.1-4.2, M1.2-2.2, VT-P1.2-4.2))

(P1.2-4.3 ← s-P1.2-4.3 (P1.1-4.3, M1.2-2.3, VT-P1.2-4.3))

(P1.2-4.4 ← s-P1.2-4.4 (P1.1-4.4, M1.2-2.4, VT-P1.2-4.4))

(P1.2-4.5 ← s-P1.2-4.5 (P1.1-4.5, M1.2-2.5, VT-P1.2-4.5))

(P1.2-4.6 ← s-P1.2-4.6 (M1.2-2.6, VT-P1.2-4.6))

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. JAVA CODE OF CASES

List of Java Source Code Files

Cases.AVCCreateStepFrame 272

Cases.AVCMergingFrame 279

Cases.AVCOpenStepFrame 288

Cases.AVCSplittingFrame 294

Cases.CalendarDialog 302

Cases.CasesFrame 305

Cases.CasesTitle 328

Cases.ComponentContentFrame 329

Cases.ComponentType 340

Cases.ConnectionLinksFrame 341

Cases.DecomposeListDialog 348

Cases.DeleteDialog 352

Cases.Dependency 356

Cases.EditDecomposeFrame 358

Cases.EHL 364

Cases.I_AVC 365

Cases.I_AVCOpenStep 365

Cases.I_Cases 366

Cases.I_ComponentContent 366

Cases.I_EditDecompose 367

Cases.I_Personnel 367

Cases.I_ProjectSchema 367

Cases.I_StepContent 368

Cases.I_Trace 368

Cases.ListDialog 368

Cases.Personnel 372

Cases.PersonnelFrame 376

Cases.ProjectSchemaFrame 385

Cases.ReviewComponentContentDialog 410

Cases.SkillTableFrame 410

Cases.StepContent 418

Cases.StepContentFrame 423

Cases.StepType 435

Cases.TraceFrame 436

Cases.VersionControl 447

JobSchedule.JAboutDialog 450

JobSchedule.JDialog_jobskill 451

JobSchedule.JDialog_message 454

JobSchedule.JDialog_message1 456

JobSchedule.JDialog_weight 457

JobSchedule.JDialog_weit 459

JobSchedule.JFrame_assignjob 462

JobSchedule.JFrame_manage 472

JobSchedule.Predecessor 483

JobSchedule.Result 484

JobSchedule.Weight 484

```
package Cases;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import com.sun.java.swing.*;
import java.text.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * Automatic Version Control - Create new version number for all steps
 *    in the current project at the same time.  That means these steps always
 *    have the same number of versions.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVC
 */
////////////////////////////////////////////////////////////////////////////////

public class AVCCreateStepFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_AVC
{
  /**
   * versionVector : stores all version numbers of current project
   */
  public Vector versionVector = new Vector();

  /**
   * pathName : current path name, C:\Cases\projectName
   */
        public String pathName;

  /**
   * EHLHashtable : stores all EHL objects which retrieve from loop.cfg
file
   */
        public Hashtable EHLHashtable = new Hashtable();

  /**
   * depHashtable : stores all Dependency objects which retrieve from
dependency.cfg file
   */
        public Hashtable depHashtable = new Hashtable();


  /**
   * Creating AVCCreateStepFrame
   */
        public AVCCreateStepFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Automated Version Control - Create Step
Version");
                getContentPane().setLayout(null);
                setSize(470,280);
                setVisible(false);

        JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel9.setText("Evolution Process");
                getContentPane().add(JLabel9);
                JLabel9.setForeground(java.awt.Color.black);
                JLabel9.setBounds(15,70,140,22);
```

```
currentLabel.setHorizontalAlignment(com.sun.java.swing.SwingC
onstants.RIGHT);
currentLabel.setText("Current Variant Number");
getContentPane().add(currentLabel);
currentLabel.setForeground(java.awt.Color.black);
currentLabel.setBounds(15,110,140,22);
OKButton.setText("OK");
OKButton.setActionCommand("jbutton");
getContentPane().add(OKButton);
OKButton.setBounds(158,210,75,22);
cancelButton.setText("Cancel");
cancelButton.setActionCommand("jbutton");
getContentPane().add(cancelButton);
cancelButton.setBounds(236,210,75,22);
newVersionTextField.setEditable(false);
getContentPane().add(newVersionTextField);

newVersionTextField.setBackground(java.awt.Color.white);

newVersionTextField.setForeground(java.awt.Color.black);
newVersionTextField.setBounds(155,150,300,22);
getContentPane().add(currentVariantComboBox);
currentVariantComboBox.setBounds(155,110,300,22);

JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);

JLabel1.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
JLabel1.setText("Create Step Version: ");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel1.setBounds(10,6,190,25);
```



```
projectLabel.setVerticalAlignment(com.sun.java.swing.SwingCon
stants.BOTTOM);
projectLabel.setText("Project Label");
getContentPane().add(projectLabel);
projectLabel.setForeground(java.awt.Color.black);
projectLabel.setFont(new Font("Dialog", Font.BOLD,
18));
projectLabel.setBounds(210,6,200,25);
getContentPane().add(EHLComboBox);
EHLComboBox.setBounds(155,70,300,22);
newStepVersionButton.setText("New Step Version");
newStepVersionButton.setActionCommand("New Step
Version");
getContentPane().add(newStepVersionButton);
newStepVersionButton.setBounds(15,150,140,22);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
OKButton.addActionListener(lSymAction);
cancelButton.addActionListener(lSymAction);
SymItem lSymItem = new SymItem();
currentVariantComboBox.addItemListener(lSymItem);
EHLComboBox.addItemListener(lSymItem);
newStepVersionButton.addActionListener(lSymAction);
//}}
}

/**
 * To be called when Create Step Version Menu Item of CasesFrame
 *   receives the event from user.
 * Retrieving Dependency and EHL object from dependency.cfg and
loop.cfg files
```

```
*/
        public AVCCreateStepFrame( String pathName, String dirName ){
    this();
            this.projectLabel.setText( dirName );
            this.pathName = pathName;
    try{
      FileInputStream fileInput = new FileInputStream(
this.pathName+"\\dependency.cfg" );
      ObjectInputStream dep = new ObjectInputStream( fileInput );
      if( dep != null ){
        this.depHashtable = (Hashtable) dep.readObject();
      }
      dep.close();
      fileInput.close();

      fileInput = new FileInputStream( this.pathName+"\\loop.cfg" );
      ObjectInputStream loopIn = new ObjectInputStream( fileInput );
      Vector loopVector = new Vector();
      if( loopIn != null ){
        loopVector = (Vector)loopIn.readObject();
        if( loopVector.size() > 0 ){

          // Setting Evolution Process combobox
          this.setLoopNameComboBox( loopVector );
        }
      }
      loopIn.close();
      fileInput.close();
    }
    catch( IOException e ){ debug("IOException: "+e);
    }
    catch( ClassNotFoundException ex ){
debug("ClassNotFoundException: "+ex);
    }
        }

        public void setVisible(boolean b)
        {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
        }

        static public void main(String args[])
        {
                (new AVCCreateStepFrame()).setVisible(true);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets and menu
bar
                Insets insets = getInsets();
                com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
                int menuBarHeight = 0;
                if (menuBar != null)
                        menuBarHeight =
menuBar.getPreferredSize().height;
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
        }

        // Used by addNotify
```

```
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel currentLabel = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JTextField newVersionTextField = new
com.sun.java.swing.JTextField();
com.sun.java.swing.JComboBox currentVariantComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel projectLabel = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JComboBox EHLComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JButton newStepVersionButton = new
com.sun.java.swing.JButton();
//}}

//{{DECLARE_MENUS
//}}


class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent
event)
        {
                Object object = event.getSource();
                if (object == OKButton)
                        OKButton_actionPerformed(event);
                else if (object == cancelButton)
                        cancelButton_actionPerformed(event);
                else if (object == newStepVersionButton)

        newStepVersionButton_actionPerformed(event);
                }
        }

  /**
   * Create new directory with new version number for all steps of the
current project
   * @param event, occur when user press OK button
   */
        public void
OKButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          checkPath(this.versionVector);
    setVisible( false );
    dispose();
        }

  /**
   * Exit AVCCreateStepFrame
   * @param event, occur when user press Cancel button
   */
        public void
cancelButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    setVisible( false );
    dispose();
        }

        /**
         * Create new version number and the default version is 1.1
   * @param event, occur when user press New Version button
         */
```

```
public void
newStepVersionButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
          if( (EHLComboBox.getItemCount() > 0) &&
(EHLComboBox.getSelectedIndex() > 0) ){
              if( this.currentVariantComboBox.getItemCount() == 0 ){
                newVersionTextField.setText("1.1");
              }
              else{
                createVersionNumber(this.versionVector);
              }
    }
        }

        class SymItem implements java.awt.event.ItemListener
        {
                public void itemStateChanged(java.awt.event.ItemEvent
event)
                {
                        Object object = event.getSource();
                        if (object == currentVariantComboBox)

        currentVariantComboBox_itemStateChanged(event);
                        else if (object == EHLComboBox)

        EHLComboBox_itemStateChanged(event);
                }
        }

  /**
   * Allows a user to select all the available Evolution processes in the
combobox
   * @param event, occur when user select Evolution Process
   */
        public void
EHLComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
          if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedItem = (String)event.getItem();
        int selectedIndex = this.EHLComboBox.getSelectedIndex();

        if( selectedIndex > 0 ){
          if( this.EHLHashtable.containsKey( selectedItem ) ){
            EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
            this.versionVector = new Vector();
            this.versionVector =
tokenizeVector((String)ehl.getEHLPath());
            this.setVariantComboBox( this.versionVector) ;
          }
        }
        else if( selectedIndex == 0 ){
          this.currentVariantComboBox.removeAllItems();
          this.newVersionTextField.setText("");
        }
      }
    }

  /**
   * Allows a user to select different variants
   * @param event, occur when user select Variant number
   */
        public void
currentVariantComboBox_itemStateChanged(java.awt.event.ItemEvent
event)
        {
          if( event.getStateChange() == ItemEvent.SELECTED ){
        String currentStep = ((String)event.getItem()).trim();
        newVersionTextField.setText("");
          }
    }

  /**
   * Short cut to print the output
```

```
 * @param string : the output string
 */
      public void debug( String string ){
         System.out.println( string );
      }

/**
 * Tokenizing a string
 * @param string : tokenized string
 * @return v : vector of string without ","
 */
      public Vector tokenizeVector(String string){
  StringTokenizer st = new StringTokenizer( string, "," );
         Vector v = new Vector();
         while( st.hasMoreTokens() ){
            v.addElement( ((String)st.nextToken()).trim() );
         }
  return v;
      }

/**
 * Adding evolution processes into EHLComboBox
 * @param loopVector : vector of evolution processes
 */
      public void setLoopNameComboBox( Vector loopVector ){
  EHLComboBox.removeAllItems();
  EHLComboBox.addItem(PROCESS_TITLE);

  this.EHLHashtable = new Hashtable();

  for( int i=0; i< loopVector.size(); i++ ){
     EHL ehl = (EHL)loopVector.elementAt( i );
     String loopName = ehl.getEHLName();

     //set step Hashtable
     this.EHLHashtable.put( loopName, ehl );
     EHLComboBox.addItem(loopName);
  }
      }

/**
 * Adding variant number of the step into currentVariantComboBox
 * @param theVersionVector : vector of variant number
 */
public void setVariantComboBox( Vector theVersionVector ){
   File aFile = new File(this.pathName,
(String)theVersionVector.elementAt(0));
   if( aFile.isDirectory() ){
      String[] list = aFile.list();

      /* Clean the combo box before update it */
      this.currentVariantComboBox.removeAllItems();
      newVersionTextField.setText("");

      if( list.length > 0 ){
         Vector v = new Vector();
         for( int i=0; i<list.length; i++ ){
            String s = list[i];
            int index = s.indexOf(".");
            String sub1 = (s.substring(0,index)).trim();
            if( !v.contains(sub1) ){
               v.addElement(sub1);
            }
         }
         if( v.size() > 0 ){
            for(int j=0; j<v.size(); j++ ){
               this.currentVariantComboBox.addItem(v.elementAt(j));
            }
         }
      }
   }
}
```

```
/**
 * Create new version number for all steps in the current project
 * @param v : vector of string (e.g., 1.1, 1.2, 2.1, ...) and they
 *          represent all existing version numbers
 */
public void createVersionNumber(Vector v){
        if( v.size() > 0 ){
          try{
            /* To check aFile is existing or not */
            File aFile = new
File(this.pathName,(String)v.elementAt(0));
            if( aFile.isDirectory() ){
              String[] list = aFile.list();
          if( list.length > 0 ){
            int index = list[0].indexOf(".");
            String sub1 =
((String)currentVariantComboBox.getSelectedItem()).trim();
            String sub2 = list[0].substring(index+1);
            int theMax = Integer.parseInt(sub2);
            for( int i=1; i<list.length; i++ ){
              index = list[i].indexOf(".");
              String s = list[i].substring(0, index);
              if( s.equals(sub1)){
                sub2 = list[i].substring(index+1);
                int temp = Math.max(theMax,Integer.parseInt(sub2));
                theMax = temp;
              }
            }
            theMax++;
            String stepVersion =
((String)this.currentVariantComboBox.getSelectedItem()).trim()+"."+theM
ax;
            newVersionTextField.setText(stepVersion);
          }
            }
      }
      catch( Exception e){}
```

```
          }
}

/**
 * Go to each step (e.g., s-I, s-C, s-R, ...) and create new directory with
 *  the name is new version number
 * @param v : vector of step names
 */
public void checkPath(Vector v){
  String currentVersion = this.newVersionTextField.getText();
  if( v.size() > 0 ){
    File myFile = new File(this.pathName);
    if( myFile.exists()) {
      String[] myList = myFile.list();
      for(int m=0; m<myList.length; m++ ){
        File f = new File(myFile, (String)myList[m]);
        if( f.isDirectory() ){
          if( v.contains(f.getName()) ){
            File theFile = new File(f,currentVersion);
            theFile.mkdir();
            if( !theFile.isDirectory() ){
              theFile.mkdir();
            }
            if( theFile.isDirectory() ){
              Dependency dep = (Dependency)
this.depHashtable.get( f.getName() );
              createFiles(dep, theFile);
            }
          }
        }
      }
    }
  }
}

/**
 * Create Component Content directory and link files inside it.
```

```
    * @param dep : Dependency object of theFile and get input.p and
input.s files
    * @param theFile : current version directory
    */
   public void createFiles( Dependency dep, File theFile){
      if( dep !=  null ){
         try{
            //Create Component Content subdirectory in thte new step
            //and include txt.link, word.link, excel.link, data.link, url.link, and
caps.link
            File compContent = new File(theFile,"Component Content");
            compContent.mkdir();
            if( compContent.isDirectory() ){
               for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
                  File newFile = new File(compContent,
LINK_FILE_NAMES[i]);
                  FileWriter fileWriter = new FileWriter( newFile );
                  BufferedWriter bw = new BufferedWriter( fileWriter );
                  bw.flush();
                  bw.close();
                  fileWriter.close();
               }
            }

            FileOutputStream fileOutput = new
FileOutputStream(theFile.getAbsolutePath()+"\\input.p");
            DataOutputStream depPrimary = new DataOutputStream(
fileOutput);
            if( depPrimary != null ){
               depPrimary.writeBytes(dep.getPrimaryInput());
            }
            depPrimary.flush();
            depPrimary.close();
            fileOutput.close();

            fileOutput = new
FileOutputStream(theFile.getAbsolutePath()+"\\input.s");
            DataOutputStream depSecondary = new DataOutputStream(
fileOutput);
            if( depSecondary != null ){
               depSecondary.writeBytes(dep.getSecondaryInput());
            }
            depSecondary.flush();
            depSecondary.close();
            fileOutput.close();
         }
         catch( IOException e ){ debug("dep_IOException: "+e);
         }
      }
   }
}
```

```
package Cases;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import java.text.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * Automatic Version Control - Evolution History Merging : to merge two
existing
 *    versions, and create the new version.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVC
 */
//////////////////////////////////////////////////////////////////////////////
```

```
public class AVCMergingFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_AVC
{
  /**
   * pathName : current path name, e.g. C:\Cases\projectName,
D:\Cases\projectName, ...
   */
        public String pathName;

  /**
   * versionVector : stores all version numbers of current project
   */
  public Vector versionVector = new Vector();

  /**
   * EHLHashtable : stores all EHL objects which retrieve from loop.cfg
file
   */
        public Hashtable EHLHashtable = new Hashtable();

  /**
   * depHashtable : stores all Dependency objects which retrieve from
dependency.cfg file
   */
        public Hashtable depHashtable = new Hashtable();

  /**
   * Creating AVCMergingFrame
   */
        public AVCMergingFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Automated Version Control - Evolution History
Merging");
                getContentPane().setLayout(null);
                setSize(500,360);
                setVisible(false);

        JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel9.setText("Evolution Process");
                getContentPane().add(JLabel9);
                JLabel9.setForeground(java.awt.Color.black);
                JLabel9.setBounds(26,70,145,22);

        currentStepLabel.setHorizontalAlignment(com.sun.java.swing.Swi
ngConstants.RIGHT);
                currentStepLabel.setText("Current Step Version");
                getContentPane().add(currentStepLabel);
                currentStepLabel.setForeground(java.awt.Color.black);
                currentStepLabel.setBounds(23,110,144,22);

        JLabel11.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
                JLabel11.setText("Variant Type");
                getContentPane().add(JLabel11);
                JLabel11.setForeground(java.awt.Color.black);
                JLabel11.setBounds(23,190,145,22);
                OKButton.setText("OK");
                OKButton.setActionCommand("jbutton");
                getContentPane().add(OKButton);
                OKButton.setBounds(171,290,75,22);
                cancelButton.setText("Cancel");
                cancelButton.setActionCommand("jbutton");
                getContentPane().add(cancelButton);
```

```
cancelButton.setBounds(249,290,75,22);
getContentPane().add(currentStepComboBox);
currentStepComboBox.setBounds(173,110,300,22);

JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);

JLabel1.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
JLabel1.setText("Evolution History Merging: ");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel1.setBounds(0,6,250,25);

projectLabel.setVerticalAlignment(com.sun.java.swing.SwingCon
stants.BOTTOM);
projectLabel.setText("Project Label");
getContentPane().add(projectLabel);
projectLabel.setForeground(java.awt.Color.black);
projectLabel.setFont(new Font("Dialog", Font.BOLD,
18));
projectLabel.setBounds(250,6,250,25);
getContentPane().add(EHLComboBox);
EHLComboBox.setBounds(173,70,300,22);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel2.setText("Merged Step Version");
getContentPane().add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(23,150,145,22);
getContentPane().add(mergedStepComboBox);
mergedStepComboBox.setBounds(173,150,300,22);
newVersionTextField.setEditable(false);
getContentPane().add(newVersionTextField);
newVersionTextField.setBackground(java.awt.Color.white);

newVersionTextField.setForeground(java.awt.Color.black);
newVersionTextField.setBounds(173,230,300,22);
getContentPane().add(variantComboBox);
variantComboBox.setBounds(173,190,300,22);
newStepVersionButton.setText("New Step Version");
newStepVersionButton.setActionCommand("New Step
Version");
getContentPane().add(newStepVersionButton);
newStepVersionButton.setBounds(33,230,135,22);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
OKButton.addActionListener(lSymAction);
cancelButton.addActionListener(lSymAction);
SymItem lSymItem = new SymItem();
EHLComboBox.addItemListener(lSymItem);
newStepVersionButton.addActionListener(lSymAction);
currentStepComboBox.addItemListener(lSymItem);
mergedStepComboBox.addItemListener(lSymItem);
variantComboBox.addItemListener(lSymItem);
//}}

/* Adding item to variantComboBox */
for( int i=0; i<VARIANT_TYPES.length; i++ ){
    variantComboBox.addItem(VARIANT_TYPES[i]);
  }
  variantComboBox.setSelectedIndex( 0 );
}

/**
```

```
    * To be called when Evolution History Merging Menu Item of
CasesFrame
    *     receives the event from user.
    * Retrieving Dependency and EHL object from dependency.cfg and
loop.cfg files
    */
  public AVCMergingFrame( String pathName, String dirName ){
    this();
            this.projectLabel.setText( dirName );
            this.pathName = pathName;

    try{
      FileInputStream fileInput = new FileInputStream(
this.pathName+"\\dependency.cfg" );
      ObjectInputStream dep = new ObjectInputStream( fileInput );
      if( dep != null ){
        this.depHashtable = (Hashtable) dep.readObject();
      }
      dep.close();
      fileInput.close();
    }
    catch( IOException e ){
      debug("IOException_Dep: "+e);
    }
    catch( ClassNotFoundException ex ){
      debug("ClassNotFoundException_Dep: "+ex);
    }

    try{
      FileInputStream fileInput = new FileInputStream(
this.pathName+"\\loop.cfg" );
      ObjectInputStream loopIn = new ObjectInputStream( fileInput );
      if( loopIn != null ){
        Vector loopVector = new Vector();
        loopVector = (Vector)loopIn.readObject();
        if( loopVector.size() > 0){
          this.setLoopNameComboBox( loopVector );
        }
      }
      loopIn.close();
      fileInput.close();
    }
    catch( IOException e ){
      debug("IOException_loop: "+e);
    }
    catch( ClassNotFoundException ex ){
      debug("ClassNotFoundException_loop: "+ex);
    }
  }

    public void setVisible(boolean b)
    {
            if (b)
                    setLocation(50, 50);
            super.setVisible(b);
    }

    static public void main(String args[])
    {
            (new AVCMergingFrame()).setVisible(true);
    }

    public void addNotify()
    {
            // Record the size of the window prior to calling parents
addNotify.
            Dimension size = getSize();

            super.addNotify();

            if (frameSizeAdjusted)
                    return;
            frameSizeAdjusted = true;
```

```
		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel currentStepLabel = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel11 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JComboBox currentStepComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel projectLabel = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JComboBox EHLComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JComboBox mergedStepComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JTextField newVersionTextField = new
com.sun.java.swing.JTextField();
	com.sun.java.swing.JComboBox variantComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JButton newStepVersionButton = new
com.sun.java.swing.JButton();
	//}}

	//{{DECLARE_MENUS
	//}}


	class SymAction implements java.awt.event.ActionListener
	{
		public void actionPerformed(java.awt.event.ActionEvent
event)
		{
			Object object = event.getSource();
			if (object == OKButton)
				OKButton_actionPerformed(event);
			else if (object == cancelButton)
				cancelButton_actionPerformed(event);
			else if (object == newStepVersionButton)

	newStepVersionButton_actionPerformed(event);
		}
	}

  /**
   * Create new directory with new version number for all steps of the
current project
   * @param event, occur when user press OK button
   */
  public void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
```

```
        {
    checkPath(this.versionVector);
    setVisible( false );
    dispose();
        }

  /**
   * Exit AVCMergingFrame
   * @param event, occur when user press Cancel button
   */
        public void
cancelButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    setVisible( false );
    dispose();
        }

        /**
         * Create new version number and the default version is 1.1
   * @param event, occur when user press New Version button
         */
  public void
newStepVersionButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
          if( (EHLComboBox.getItemCount() > 0) &&
(EHLComboBox.getSelectedIndex() > 0) ){
              if( this.currentStepComboBox.getItemCount()>0 &&
                 this.mergedStepComboBox.getItemCount() > 0 ){
              createVersionNumber(variantComboBox.getSelectedIndex());
              }
              else{
                 newVersionTextField.setText("1.1");
              }
    }
        }
```

```
        class SymItem implements java.awt.event.ItemListener
        {
                public void itemStateChanged(java.awt.event.ItemEvent
event)
                {
                        Object object = event.getSource();
                        if (object == EHLComboBox)

        EHLComboBox_itemStateChanged(event);
                        else if (object == currentStepComboBox)

        currentStepComboBox_itemStateChanged(event);
                else if (object == mergedStepComboBox)

        currentStepComboBox_itemStateChanged(event);
                        else if (object == variantComboBox)

        currentStepComboBox_itemStateChanged(event);
                }
        }

  /**
   * Allows a user to select all the available Evolution processes in the
combobox
   * @param event, occur when user select Evolution Process
   */
  public void
EHLComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
                if( event.getStateChange() == ItemEvent.SELECTED ){
      String selectedItem = (String)event.getItem();
      int selectedIndex = this.EHLComboBox.getSelectedIndex();

      this.currentStepComboBox.removeAllItems();
      this.mergedStepComboBox.removeAllItems();
      newVersionTextField.setText("");
```

```
        if( selectedIndex > 0 ){
          if( this.EHLHashtable.containsKey( selectedItem ) ){
            EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
            this.versionVector = new Vector();
            this.versionVector =
tokenizeVector((String)ehl.getEHLPath());
            this.setVersionComboBoxes( this.versionVector) ;
          }
        }
      }
    }

    /**
     * Allows a user to select all the available steps in the combobox
     * @param event, occur when user select currentStepComboBox
     */
        public void
currentStepComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
      if( event.getStateChange() == ItemEvent.SELECTED ){
        newVersionTextField.setText("");
      }
        }

    /**
     * Short cut to print the output
     * @param string : the output string
     */
        public void debug( String string ){
          System.out.println( string );
        }

    /**
     * Adding evolution processes into EHLComboBox
     * @param loopVector : vector of evolution processes
     */
        public void setLoopNameComboBox( Vector loopVector ){
      EHLComboBox.removeAllItems();
      EHLComboBox.addItem(PROCESS_TITLE);

      this.EHLHashtable = new Hashtable();

      for( int i=0; i< loopVector.size(); i++ ){
        EHL ehl = (EHL)loopVector.elementAt( i );
        String loopName = ehl.getEHLName();

        //set step Hashtable
        this.EHLHashtable.put( loopName, ehl );

        this.EHLComboBox.addItem(loopName);
      }
        }

    /**
     * Adding version numbers to currentStepComboBox and
mergedStepComboBox
     * @param versionVector : vector of version numbers
     */
    public void setVersionComboBoxes( Vector versionVector ){

      File aFile = new File(this.pathName,
(String)versionVector.elementAt(0));
      if( aFile.isDirectory() ){
        String[] list = aFile.list();

        if( list.length > 0 ){
          for( int i=0; i<list.length; i++ ){
            this.currentStepComboBox.addItem(((String)list[i]).trim());
            this.mergedStepComboBox.addItem(((String)list[i]).trim());
          }
        }
      }
    }
```

```
/**
 * Tokenizing a string
 * @param string : tokenized string
 * @return v : vector of string without ","
 */
      public Vector tokenizeVector(String string){
   StringTokenizer st = new StringTokenizer( string, "," );
          Vector v = new Vector();
          while( st.hasMoreTokens() ){
             v.addElement( ((String)st.nextToken()).trim() );
          }
   return v;
      }

/**
 * Create new version number for all steps in the current project
 * @param typeIndex = 0 ==> variant is old, new version =
currentVariant.(highestVersion+1);
 *              = 1 ==> variant is new, new version =
(highestVariant+1).(highestVersion+1);
 */
public void createVersionNumber(int typeIndex){
          String current =
((String)currentStepComboBox.getSelectedItem()).trim();
          String merged =
((String)mergedStepComboBox.getSelectedItem()).trim();

          if( current.equals(merged) ){
      JOptionPane.showMessageDialog(this, " Current Step Version must
be different from Merged Step Version!",
                        "Error Message",
JOptionPane.ERROR_MESSAGE);
      return;
          }
          else{
      try{
         int index1 = current.indexOf(".");
         int index2 = merged.indexOf(".");
         String sub1 = current.substring(index1+1);
         String sub2 = merged.substring(index2+1);
         int theMax2 =
Math.max(Integer.parseInt(sub1),Integer.parseInt(sub2));
         theMax2++;
         String mergedVersion = null;

         if( typeIndex == 0 ){
            mergedVersion = current.substring(0,index1)+"."+theMax2;
         }
         else if( typeIndex == 1 ){
            index1 = current.indexOf(".");
            index2 = merged.indexOf(".");
            sub1 = current.substring(0, index1);
            sub2 = merged.substring(0, index2);
            int theMax1 =
Math.max(Integer.parseInt(sub1),Integer.parseInt(sub2));
            theMax1++;
            mergedVersion = theMax1+"."+theMax2;
         }

         for( int i=0; i<currentStepComboBox.getItemCount(); i++ ){
            String s = (String)currentStepComboBox.getItemAt(i);
            if( s.equals(mergedVersion) ){
               JOptionPane.showMessageDialog(this, mergedVersion+"
version already exists in this project!",
                        "Error Message",
JOptionPane.ERROR_MESSAGE);
               return;
            }
            else if( i==currentStepComboBox.getItemCount()-1){
               newVersionTextField.setText(mergedVersion);
            }
         }
      }
      catch( Exception e){ }
```

```
    }
  }

  /**
   * Go to each step (e.g., s-I, s-C, s-R, ...) and create new directory with
   *  the name is new version number
   * @param v : vector of step names
   */
  public void checkPath(Vector v){
    String mergedVersion = this.newVersionTextField.getText();
     if( v.size() > 0 ){
       File myFile = new File(this.pathName);
       if( myFile.exists()) {
          String[] myList = myFile.list();
          for(int m=0; m<myList.length; m++ ){
             File f = new File(myFile, (String)myList[m]);
             if( f.isDirectory() ){
                if( v.contains(f.getName()) ){
                   File theFile = new File(f,mergedVersion);
                   theFile.mkdir();
                   if( !theFile.isDirectory() ){
                      theFile.mkdir();
                   }
                   if( theFile.isDirectory() ){
                      Dependency dep = (Dependency)
this.depHashtable.get( f.getName() );
                         createFiles(dep, theFile);
                   }
                }
             }
          }
       }
     }
  }

  /**
   * Create Component Content directory and link files inside it.
   * @param dep : Dependency object of theFile and get input.p and
input.s files
   * @param theFile : current version directory
   */
  public void createFiles( Dependency dep, File theFile){
     if( dep !=  null ){
        try{
           //Create Component Content subdirectory in thte new step
           //and include txt.link, data.link, url.link, and caps.link
           File compContent = new File(theFile,"Component Content");
           compContent.mkdir();
           if( compContent.isDirectory() ){
              for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
                 File newFile = new File(compContent,
LINK_FILE_NAMES[i]);
                 FileWriter fileWriter = new FileWriter( newFile );
                 BufferedWriter bw = new BufferedWriter( fileWriter );
                 bw.flush();
                 bw.close();
                 fileWriter.close();
              }
           }

           FileOutputStream fileOutput = new
FileOutputStream(theFile.getAbsolutePath()+"\\input.p");
           DataOutputStream depPrimary = new DataOutputStream(
fileOutput);
           if( depPrimary != null ){
              depPrimary.writeBytes(dep.getPrimaryInput());
           }
           depPrimary.flush();
           depPrimary.close();
           fileOutput.close();

           fileOutput = new
FileOutputStream(theFile.getAbsolutePath()+"\\input.s");
```

```
            DataOutputStream depSecondary = new DataOutputStream(
fileOutput);
            if( depSecondary != null ){
              depSecondary.writeBytes(dep.getSecondaryInput());
            }
            depSecondary.flush();
            depSecondary.close();
            fileOutput.close();
          }
          catch( IOException e ){
            debug("dep_IOException: "+e);
          }
        }
      }
    }
```



```
package Cases;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * Automatic Version Control - Open existing version number for all steps
 *    in the current project.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVCOpenStep
 */
////////////////////////////////////////////////////////////////////////////////
public class AVCOpenStepFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_AVCOpenStep
{
  /**
   * pathName : current path name, e.g. C:\Cases\projectName,
D:\Cases\projectName, ...
   */
  public String pathName;

  /**
   * EHLHashtable : stores all EHL objects which retrieve from loop.cfg
file
   */
        public Hashtable EHLHashtable = new Hashtable();

        /**
         * oldVersionControl : VersionControl object, keeping the content
of current.cfg file
         */
        protected VersionControl oldVersionControl = null;


  /**
   * Creating AVCOpenStepFrame
   */
        public AVCOpenStepFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Automated Version Control - Open Step
Version");
```

```
            getContentPane().setLayout(null);
            setSize(450,290);
            setVisible(false);

        JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
            JLabel9.setText("Evolution Process");
            getContentPane().add(JLabel9);
            JLabel9.setForeground(java.awt.Color.black);
            JLabel9.setBounds(5,70,110,22);

        JLabel10.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
            JLabel10.setText("Step Type");
            getContentPane().add(JLabel10);
            JLabel10.setForeground(java.awt.Color.black);
            JLabel10.setBounds(5,110,110,22);

        JLabel12.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
            JLabel12.setText("Step Version");
            getContentPane().add(JLabel12);
            JLabel12.setForeground(java.awt.Color.black);
            JLabel12.setBounds(5,150,110,22);
            OKButton.setText("OK");
            OKButton.setActionCommand("jbutton");
            getContentPane().add(OKButton);
            OKButton.setBounds(148,210,75,22);
            cancelButton.setText("Cancel");
            cancelButton.setActionCommand("jbutton");
            getContentPane().add(cancelButton);
            cancelButton.setBounds(226,210,75,22);
            getContentPane().add(stepIDComboBox);
            stepIDComboBox.setBounds(117,110,300,22);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel1.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
            JLabel1.setText("Open Step Version: ");
            getContentPane().add(JLabel1);
            JLabel1.setForeground(java.awt.Color.black);
            JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
            JLabel1.setBounds(0,6,180,25);

        projectLabel.setVerticalAlignment(com.sun.java.swing.SwingCon
stants.BOTTOM);
            projectLabel.setText("Project Label");
            getContentPane().add(projectLabel);
            projectLabel.setForeground(java.awt.Color.black);
            projectLabel.setFont(new Font("Dialog", Font.BOLD,
18));
            projectLabel.setBounds(187,6,200,25);
            getContentPane().add(EHLComboBox);
            EHLComboBox.setBounds(117,70,300,22);
            getContentPane().add(versionComboBox);
            versionComboBox.setBounds(117,150,300,22);
            //}}

            //{{INIT_MENUS
            //}}

            //{{REGISTER_LISTENERS
            SymAction lSymAction = new SymAction();
            OKButton.addActionListener(lSymAction);
            cancelButton.addActionListener(lSymAction);
            SymItem lSymItem = new SymItem();
            EHLComboBox.addItemListener(lSymItem);
            stepIDComboBox.addItemListener(lSymItem);
            //}}
        }

    /**
```

```
    * To be called when Open Step Version Menu Item of CasesFrame
    *     receives the event from user.
    * Retrieving VersionControl and EHL object from current.cfg and
loop.cfg files
    */
  public AVCOpenStepFrame( String dirName, String pathName ){
    this();
          this.projectLabel.setText( dirName );
          this.pathName = pathName;

    try{
      FileInputStream fileInput = new FileInputStream(
this.pathName+"\\loop.cfg" ) ;
      ObjectInputStream loopIn = new ObjectInputStream( fileInput );
      if( loopIn != null ){
        Vector loopVector = new Vector();
        loopVector = (Vector)loopIn.readObject();
        if( loopVector.size() > 0 ){
          this.setLoopNameComboBox( loopVector );
        }
      }
      loopIn.close();
      fileInput.close();
    }
    catch( IOException e ){
      debug("IOException_loop: "+e);
    }
    catch( ClassNotFoundException ex ){
      debug("ClassNotFoundException_loop: "+ex);
    }

    try{
      FileInputStream fileInput = new FileInputStream(
this.pathName+"\\current.vsn" );
      ObjectInputStream currentIn = new ObjectInputStream( fileInput );
      if( currentIn != null ){
        VersionControl oldVersionControl =
(VersionControl)currentIn.readObject();
        if( oldVersionControl != null ){
          this.setInitialFrame(oldVersionControl);
        }
      }
      currentIn.close();
      fileInput.close();
    }
    catch( IOException e ){
      debug("IOException_currentIn: "+e);
    }
    catch( ClassNotFoundException ex ){
      debug("ClassNotFoundException_currentLoop: "+ex);
    }
  }

    public AVCOpenStepFrame(String sTitle)
    {
        this();
        setTitle(sTitle);
    }

    public void setVisible(boolean b)
    {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    static public void main(String args[])
    {
        (new AVCOpenStepFrame()).setVisible(true);
    }

    public void addNotify()
    {
```

```
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel10 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel12 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JComboBox stepIDComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel projectLabel = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JComboBox EHLComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JComboBox versionComboBox = new
com.sun.java.swing.JComboBox();
	//}}

	//{{DECLARE_MENUS
	//}}


	class SymAction implements java.awt.event.ActionListener
	{
		public void actionPerformed(java.awt.event.ActionEvent
event)
		{
			Object object = event.getSource();
			if (object == OKButton)
				OKButton_actionPerformed(event);
			else if (object == cancelButton)
				cancelButton_actionPerformed(event);
		}
	}

  /**
   * Create new directory with new version number for all steps of the
current project
   * @param event, occur when user press OK button
   */
  public void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
	{
```

```
        String currentLoop = (String)this.EHLComboBox.getSelectedItem();
        String currentStep = (String)this.stepIDComboBox.getSelectedItem();
        String currentVersion = (String)
this.versionComboBox.getSelectedItem();
        String currentStatus = "";

        VersionControl vc = new VersionControl( currentLoop, currentStep,
                                currentVersion, currentStatus );

        try{
          FileOutputStream fileOutput = new FileOutputStream(
this.pathName+"\\current.vsn" );
          ObjectOutputStream currentOut= new ObjectOutputStream(
fileOutput );
                if( currentOut != null ){
                  currentOut.writeObject( vc );
                }
                currentOut.flush();
                currentOut.close();
          fileOutput.close();
        }
        catch( IOException e1 ){
          debug("IOException_currentOut: "+e1);
        }

        setVisible( false );
        dispose();
            }

      /**
        * Exit AVCOpenStepFrame
        * @param event, occur when user press Cancel button
        */
            public void
cancelButton_actionPerformed(java.awt.event.ActionEvent event)
            {
        setVisible( false );
        dispose();
            }

            class SymItem implements java.awt.event.ItemListener
            {
                    public void itemStateChanged(java.awt.event.ItemEvent
event)
                    {
                            Object object = event.getSource();
                            if (object == EHLComboBox)

            EHLComboBox_itemStateChanged(event);
                            else if (object == stepIDComboBox)

            stepIDComboBox_itemStateChanged(event);
                    }
            }

      /**
        * Allows a user to select all the available Evolution processes in the
combobox
        * @param event, occur when user select Evolution Process
        */
            public void
EHLComboBox_itemStateChanged(java.awt.event.ItemEvent event)
            {
                    if( event.getStateChange() == ItemEvent.SELECTED ){
            String selectedItem = (String)event.getItem();
            int selectedIndex = this.EHLComboBox.getSelectedIndex();

            if( selectedIndex > 0 ){
              if( this.EHLHashtable.containsKey( selectedItem ) ){
                EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );

                StringTokenizer st = new StringTokenizer(
(String)ehl.getEHLPath(), "," );
                        Vector tokenizeVector = new Vector();
```

```
                while( st.hasMoreTokens() ){
                  tokenizeVector.addElement( st.nextToken() );
                }
            this.setStepComboBox( tokenizeVector );
          }
        }
        else if( selectedIndex == 0 ){
          this.stepIDComboBox.removeAllItems();
          this.versionComboBox.removeAllItems();
        }
      }
    }

    /**
     * Allows a user to select all the available step ID in the combobox
     * @param event, occur when user select step ID
     */
        public void
stepIDComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
                if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedStep = ((String)event.getItem()).trim();
        int selectedIndex = this.stepIDComboBox.getSelectedIndex();

        if( selectedIndex > 0 ){

          Vector versionVector = new Vector();
          File existedDir = new File( this.pathName, selectedStep );
          if( existedDir.isDirectory() ){
            String[] fileList = existedDir.list();

            for( int j=0; j<fileList.length; j++ ){
              versionVector.addElement( fileList[j] );
            }
            this.setVersionComboBox(versionVector);
          }
            }
        else if( selectedIndex == 0 ){
          this.versionComboBox.removeAllItems();
        }
            }
}

/**
 * Short cut to print the output
 * @param string : the output string
 */
        public void debug( String string ){
          System.out.println( string );
        }

/**
 * Adding evolution processes into EHLComboBox
 * @param loopVector : vector of evolution processes
 */
public void setLoopNameComboBox( Vector loopVector ){
  EHLComboBox.removeAllItems();
  EHLComboBox.addItem(PROCESS_TITLE);

  this.EHLHashtable = new Hashtable();

  for( int i=0; i< loopVector.size(); i++ ){
    EHL ehl = (EHL)loopVector.elementAt( i );

    //set step Hashtable
    this.EHLHashtable.put( ehl.getEHLName(), ehl );

    this.EHLComboBox.addItem(ehl.getEHLName());
  }
    }

/**
 * Adding step ID into stepIDComboBox
 * @param stepVector : vector of step ID
```

```
  */
 public void setStepComboBox( Vector stepVector ){
   this.stepIDComboBox.removeAllItems();
   this.stepIDComboBox.addItem(STEP_TYPE_TITLE);

   Vector versionVector = new Vector();
   for( int i=0; i< stepVector.size(); i++ ){
     String stepID = (String)stepVector.elementAt( i );
     this.stepIDComboBox.addItem( stepID.trim() );
   }

   this.setVersionComboBox( versionVector );
 }

 /**
  * Adding version numbers to versionComboBox
  * @param versionVector : vector of version numbers
  */
 public void setVersionComboBox( Vector versionVector ){
   this.versionComboBox.removeAllItems();

   for( int i=0; i < versionVector.size(); i++ ){
     String vv = (String)versionVector.elementAt( i );
     this.versionComboBox.addItem( vv.trim() );
   }
 }

 /**
  * Adding version numbers to currentStepComboBox and
mergedStepComboBox
  * @param vc : vector of VersionControl objects
  */
 public void setInitialFrame( VersionControl vc ){
   if( this.EHLComboBox.getItemCount() > 0 ){
     this.EHLComboBox.setSelectedItem(vc.getCurrentLoop());
     this.stepIDComboBox.setSelectedItem(vc.getCurrentStep());
     this.versionComboBox.setSelectedItem(vc.getCurrentVersion());
   }
 }
}
```

```
package Cases;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import com.sun.java.swing.*;
import java.text.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * Automatic Version Control - Evolution History Splitting : to split the
existing
 *    versions into new version.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVC
 */
////////////////////////////////////////////////////////////////////////////////
public class AVCSplittingFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_AVC
{
  /**
   * pathName : current path name, e.g. C:\Cases\projectName,
D:\Cases\projectName, ...
   */
        public String pathName;

  /**
```

```
 * versionVector : stores all version numbers of current project
 */
public Vector versionVector = new Vector();

/**
 * EHLHashtable : stores all EHL objects which retrieve from loop.cfg
file
 */
        public Hashtable EHLHashtable = new Hashtable();

/**
 * depHashtable : stores all Dependency objects which retrieve from
dependency.cfg file
 */
public Hashtable depHashtable = new Hashtable();


/**
 * Creating AVCSplittingFrame
 */
        public AVCSplittingFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Automated Version Control -  Evolution History
Splitting");
                getContentPane().setLayout(null);
                setSize(480,280);
                setVisible(false);

        JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel9.setText("Evolution Process");
                getContentPane().add(JLabel9);
                JLabel9.setForeground(java.awt.Color.black);
                JLabel9.setBounds(15,70,145,22);

        currentLabel.setHorizontalAlignment(com.sun.java.swing.SwingC
onstants.RIGHT);
                currentLabel.setText("Current Step Version");
                getContentPane().add(currentLabel);
                currentLabel.setForeground(java.awt.Color.black);
                currentLabel.setBounds(15,110,145,22);
                OKButton.setText("OK");
                OKButton.setActionCommand("jbutton");
                getContentPane().add(OKButton);
                OKButton.setBounds(163,210,75,22);
                cancelButton.setText("Cancel");
                cancelButton.setActionCommand("jbutton");
                getContentPane().add(cancelButton);
                cancelButton.setBounds(241,210,75,22);
                newVersionTextField.setEditable(false);
                getContentPane().add(newVersionTextField);

        newVersionTextField.setBackground(java.awt.Color.white);

        newVersionTextField.setForeground(java.awt.Color.black);
                newVersionTextField.setBounds(164,150,300,22);
                getContentPane().add(currentStepComboBox);
                currentStepComboBox.setBounds(164,110,300,22);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);

        JLabel1.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
```

```
JLabel1.setText("Evolution History Splitting:");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel1.setBounds(0,6,250,25);

projectLabel.setVerticalAlignment(com.sun.java.swing.SwingCon
stants.BOTTOM);
projectLabel.setText("Project Label");
getContentPane().add(projectLabel);
projectLabel.setForeground(java.awt.Color.black);
projectLabel.setFont(new Font("Dialog", Font.BOLD,
18));
projectLabel.setBounds(250,6,230,25);
getContentPane().add(EHLComboBox);
EHLComboBox.setBounds(164,70,300,22);
newStepVersionButton.setText("New Step Version");
newStepVersionButton.setActionCommand("New Step
Version");
getContentPane().add(newStepVersionButton);
newStepVersionButton.setBounds(25,150,135,22);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
OKButton.addActionListener(lSymAction);
cancelButton.addActionListener(lSymAction);
SymItem lSymItem = new SymItem();
currentStepComboBox.addItemListener(lSymItem);
EHLComboBox.addItemListener(lSymItem);
newStepVersionButton.addActionListener(lSymAction);
//}}
}
```

```
/**
 * To be called when Evolution History Splitting Menu Item of
CasesFrame
 *    receives the event from user.
 * Retrieving Dependency and EHL object from dependency.cfg and
loop.cfg files
 */
    public AVCSplittingFrame(String pathName, String dirName ){
  this();
      this.projectLabel.setText( dirName );
      this.pathName = pathName;
  try{
    FileInputStream fileInput = new FileInputStream(
this.pathName+"\\dependency.cfg" );
    ObjectInputStream dep = new ObjectInputStream( fileInput );
    if( dep != null ){
      this.depHashtable = (Hashtable) dep.readObject();
    }
    dep.close();
    fileInput.close();
  }
  catch( IOException e ){
    debug("IOException_Dep: "+e);
  }
  catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_Dep: "+ex);
  }

  try{
    FileInputStream fileInput = new FileInputStream(
this.pathName+"\\loop.cfg" );
    ObjectInputStream loopIn = new ObjectInputStream( fileInput );
    if( loopIn != null ){
      Vector loopVector = new Vector();
      loopVector = (Vector)loopIn.readObject();
      if( loopVector.size() > 0){
        this.setLoopNameComboBox( loopVector );
```

```
        }
      }
      loopIn.close();
      fileInput.close();
    }
    catch( IOException e ){
      debug("IOException_loop: "+e);
    }
    catch( ClassNotFoundException ex ){
      debug("ClassNotFoundException_loop: "+ex);
    }
      }

      public AVCSplittingFrame(String sTitle)
      {
            this();
            setTitle(sTitle);
      }

      public void setVisible(boolean b)
      {
            if (b)
                  setLocation(50, 50);
            super.setVisible(b);
      }

      static public void main(String args[])
      {
            (new AVCSplittingFrame()).setVisible(true);
      }

      public void addNotify()
      {
            // Record the size of the window prior to calling parents
addNotify.
            Dimension size = getSize();
            super.addNotify();

            if (frameSizeAdjusted)
                  return;
            frameSizeAdjusted = true;

            // Adjust size of frame according to the insets and menu
bar
            Insets insets = getInsets();
            com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
            int menuBarHeight = 0;
            if (menuBar != null)
                  menuBarHeight =
menuBar.getPreferredSize().height;
            setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
      }

      // Used by addNotify
      boolean frameSizeAdjusted = false;

      //{{DECLARE_CONTROLS
      com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
      com.sun.java.swing.JLabel currentLabel = new
com.sun.java.swing.JLabel();
      com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
      com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
      com.sun.java.swing.JTextField newVersionTextField = new
com.sun.java.swing.JTextField();
      com.sun.java.swing.JComboBox currentStepComboBox = new
com.sun.java.swing.JComboBox();
      com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
```

```
        com.sun.java.swing.JLabel projectLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox EHLComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JButton newStepVersionButton = new
com.sun.java.swing.JButton();
        //}}

        //{{DECLARE_MENUS
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == OKButton)
                                OKButton_actionPerformed(event);
                        else if (object == cancelButton)
                                cancelButton_actionPerformed(event);
                        else if (object == newStepVersionButton)

        newStepVersionButton_actionPerformed(event);
                }
        }

  /**
   * Create new directory with new version number for all steps of the
current project
   * @param event, occur when user press OK button
   */
        public void
OKButton_actionPerformed(java.awt.event.ActionEvent event)
        {
     checkPath(this.versionVector);
            setVisible( false );
     dispose();
        }

  /**
   * Exit AVCSplittingFrame
   * @param event, occur when user press Cancel button
   */
        public void
cancelButton_actionPerformed(java.awt.event.ActionEvent event)
        {
     setVisible( false );
     dispose();
        }

        /**
         * Create new version number and the default version is 1.1
   * @param event, occur when user press New Version button
         */
        public void
newStepVersionButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
          if( (EHLComboBox.getItemCount() > 0) &&
(EHLComboBox.getSelectedIndex() > 0) ){
            if( this.currentStepComboBox.getItemCount() == 0 ){
              newVersionTextField.setText("1.1");
            }
            else{
              createVersionNumber();
            }
     }
        }

        class SymItem implements java.awt.event.ItemListener
        {
```

```
                public void itemStateChanged(java.awt.event.ItemEvent
event)
                {
                        Object object = event.getSource();
                        if (object == currentStepComboBox)

        currentStepComboBox_itemStateChanged(event);
                        else if (object == EHLComboBox)

        EHLComboBox_itemStateChanged(event);
                }
        }

  /**
   * Allows a user to select all the available Evolution processes in the
combobox
   * @param event, occur when user select Evolution Process
   */
        public void
EHLComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
          if( event.getStateChange() == ItemEvent.SELECTED ){
      String selectedItem = (String)event.getItem();
      int selectedIndex = this.EHLComboBox.getSelectedIndex();

      if( selectedIndex > 0 ){
        if( this.EHLHashtable.containsKey( selectedItem ) ){
          EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
          this.versionVector = new Vector();
          this.versionVector =
tokenizeVector((String)ehl.getEHLPath());
          this.setVersionComboBox( this.versionVector);
        }
      }
      else if( selectedIndex == 0 ){
        this.currentStepComboBox.removeAllItems();
        this.newVersionTextField.setText("");
```

```
        }
      }
    }

    /**
     * Allows a user to select all the available steps in the combobox
     * @param event, occur when user select currentStepComboBox
     */
        public void
currentStepComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
          if( event.getStateChange() == ItemEvent.SELECTED ){
      newVersionTextField.setText("");
          }
    }

    /**
     * Short cut to print the output
     * @param string : the output string
     */
        public void debug( String string ){
          System.out.println( string );
        }

    /**
     * Tokenizing a string
     * @param string : tokenized string
     * @return v : vector of string without ","
     */
        public Vector tokenizeVector(String string){
    StringTokenizer st = new StringTokenizer( string, "," );
          Vector v = new Vector();
          while( st.hasMoreTokens() ){
            v.addElement( st.nextToken() );
          }
    return v;
        }
```

```
/**
 * Adding evolution processes into EHLComboBox
 * @param loopVector : vector of evolution processes
 */
      public void setLoopNameComboBox( Vector loopVector ){
  EHLComboBox.removeAllItems();
  EHLComboBox.addItem(PROCESS_TITLE);

  this.EHLHashtable = new Hashtable();

  for( int i=0; i< loopVector.size(); i++ ){
    EHL ehl = (EHL)loopVector.elementAt( i );
    String loopName = ehl.getEHLName();

    //set step Hashtable
    this.EHLHashtable.put( loopName, ehl );

    EHLComboBox.addItem(loopName);
  }
      }

/**
 * Adding version numbers to currentStepComboBox
 * @param versionVector : vector of version numbers
 */
public void setVersionComboBox( Vector versionVector ){
  this.currentStepComboBox.removeAllItems();
  File aFile = new File(this.pathName,
(String)versionVector.elementAt(0));
  if( aFile.isDirectory() ){
    String[] list = aFile.list();

    if( list.length > 0 ){
      for( int i=0; i<list.length; i++ ){
        this.currentStepComboBox.addItem(((String)list[i]).trim());
      }
    }
  }
}


/**
 * Create new version number for all steps in the current project
 * new version = highest version + 1.1
 */
public void createVersionNumber(){
  try{
    int variantMax = findMaxVariant() + 1;
        String current =
((String)currentStepComboBox.getSelectedItem()).trim();
    int index = current.indexOf(".");
    String sub1 = current.substring(0, index);
    String sub2 = current.substring(index+1);
    int i1 = Integer.parseInt(sub1)+1;
    int i2 = Integer.parseInt(sub2)+1;
    String stepVersion = variantMax+"."+i2;
    for( int i=0; i<currentStepComboBox.getItemCount(); i++ ){
      String s = (String)currentStepComboBox.getItemAt(i);
      if( s.equals(stepVersion) ){
        JOptionPane.showMessageDialog(this, stepVersion+" version
already exists in this project!",
                "Error Message",
JOptionPane.ERROR_MESSAGE);
        return;
      }
      else if( i==currentStepComboBox.getItemCount()-1){
        newVersionTextField.setText(stepVersion);
      }
    }
  }
  catch( Exception e){ }
}
```

```
/**
 * Go to each step (e.g., s-I, s-C, s-R, ...) and create new directory with
 *  the name is new version number
 * @param v : vector of step names
 */
public void checkPath(Vector v){
   String mergedVersion = this.newVersionTextField.getText();
    if( v.size() > 0 ){
      File myFile = new File(this.pathName);
      if( myFile.exists()) {
         String[] myList = myFile.list();
         for(int m=0; m<myList.length; m++ ){
            File f = new File(myFile, (String)myList[m]);
            if( f.isDirectory() ){
               if( v.contains(f.getName()) ){
                  File theFile = new File(f,mergedVersion);
                  theFile.mkdir();
                  if( !theFile.isDirectory() ){
                     theFile.mkdir();
                  }
                  if( theFile.isDirectory() ){
                     Dependency dep = (Dependency)
this.depHashtable.get( f.getName() );
                        createFiles(dep, theFile);
                  }
               }
            }
         }
      }
   }
}

/**
 * Create Component Content directory and link files inside it.
 * @param dep : Dependency object of theFile and get input.p and
input.s files
 * @param theFile : current version directory
 */
public void createFiles( Dependency dep, File theFile){
   if( dep != null ){
      try{
         //Create Component Content subdirectory in thte new step
         //and include txt.link, data.link, url.link, and caps.link
         File compContent = new File(theFile,"Component Content");
         compContent.mkdir();
         if( compContent.isDirectory() ){
            for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
               File newFile = new File(compContent,
LINK_FILE_NAMES[i]);
               FileWriter fileWriter = new FileWriter( newFile );
               BufferedWriter bw = new BufferedWriter( fileWriter );
               bw.flush();
               bw.close();
               fileWriter.close();
            }
         }

         FileOutputStream fileOutput = new
FileOutputStream(theFile.getAbsolutePath()+"\\input.p");
         DataOutputStream depPrimary = new DataOutputStream(
fileOutput);
         if( depPrimary != null ){
            depPrimary.writeBytes(dep.getPrimaryInput());
         }
         depPrimary.flush();
         depPrimary.close();
         fileOutput.close();

         fileOutput = new
FileOutputStream(theFile.getAbsolutePath()+"\\input.s");
         DataOutputStream depSecondary = new DataOutputStream(
fileOutput);
         if( depSecondary != null ){
            depSecondary.writeBytes(dep.getSecondaryInput());
```

```
            }
            depSecondary.flush();
            depSecondary.close();
            fileOutput.close();
         }
         catch( IOException e ){
            debug("dep_IOException: "+e);
         }
      }
   }
   /**
    * Find the maximum variant number of the step
    * @return variantMax : the maximum variant number of this process
    */
   public int findMaxVariant(){
      int variantMax = 0;
      File aFile = new File(this.pathName,
(String)versionVector.elementAt(0));
      if( aFile.isDirectory() ){
         String[] list = aFile.list();

         if( list.length > 0 ){
            Vector v = new Vector();
            for( int i=0; i<list.length; i++ ){
               String s = list[i];
               int index = s.indexOf(".");
               String sub1 = (s.substring(0,index)).trim();
               if( !v.contains(sub1) ){
                  v.addElement(sub1);
               }
            }
            if( v.size() > 0 ){
               try{
                  variantMax = Integer.parseInt((String)v.elementAt(0));
                  for(int j=1; j<v.size(); j++ ){
                     variantMax =
Math.max(variantMax,Integer.parseInt((String)v.elementAt(j)));
```

```
                  }
               }
               catch(Exception e){}
            }
         }
      }
      return variantMax;
   }
}
```

```
package Cases;

import java.awt.*;
import java.text.*;
import java.util.*;
import com.sun.java.swing.*;
import symantec.itools.awt.util.Calendar;

////////////////////////////////////////////////////////////////////////////////
/**
 * CalendarDialog : display the calendar and return the selected day as a
string
 */
////////////////////////////////////////////////////////////////////////////////
public class CalendarDialog extends com.sun.java.swing.JDialog
{
   private int index = 0;
   StepContentFrame scf = null;

         public CalendarDialog(Frame parent)
         {
                  super(parent);
```

```
        // This code is automatically generated by Visual Cafe
when you add
        // components to the visual environment. It instantiates
and initializes
        // the components. To modify the code, only use code
syntax that matches
        // what Visual Cafe can generate, or Visual Cafe may be
unable to back
        // parse your Java file into its visual environment.
        //{{INIT_CONTROLS
        setTitle("Calendar");
        setModal(true);
        getContentPane().setLayout(null);
        setSize(220,250);
        setVisible(false);
        getContentPane().add(theCalendar);
        theCalendar.setBackground(java.awt.Color.lightGray);
        theCalendar.setFont(new Font("Dialog", Font.BOLD,
10));
        theCalendar.setBounds(5,15,210,170);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("Cancel");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(109,200,73,24);
        OKButton.setText("OK");
        OKButton.setActionCommand("OK");
        getContentPane().add(OKButton);
        OKButton.setBounds(37,200,73,24);
        //}}

        //{{REGISTER_LISTENERS
        SymAction lSymAction = new SymAction();
        OKButton.addActionListener(lSymAction);
        cancelButton.addActionListener(lSymAction);
        //}}
    }
```

```
  /**
   * Receive the event from Deadline, Earliest Start Time, and Finish Time
buttons
   * from StepContentFrame
   */
  public CalendarDialog(StepContentFrame scf, String currentDate, int
index){
    this();
    this.scf = scf;
    this.index = index;

    try{
      theCalendar.setDate(currentDate);
    }
    catch(Exception e){System.out.println(e);}
  }

      public CalendarDialog()
      {
            this((Frame)null);
      }

      public CalendarDialog(String sTitle)
      {
            this();
            setTitle(sTitle);
      }

      public void setVisible(boolean b)
      {
            if (b)

      setLocation((Toolkit.getDefaultToolkit().getScreenSize().width -
                        this.getSize().width) / 2,

      (Toolkit.getDefaultToolkit().getScreenSize().height -
                        this.getSize().height) / 2);
```

```
		super.setVisible(b);
	}

	static public void main(String args[])
	{
		(new CalendarDialog()).setVisible(true);
	}

	public void addNotify()
	{
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets
		Insets insets = getInsets();
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	symantec.itools.awt.util.Calendar theCalendar = new
symantec.itools.awt.util.Calendar();
	com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
	//}}
```

```
	class SymAction implements java.awt.event.ActionListener
	{
		public void actionPerformed(java.awt.event.ActionEvent
event)
		{
			Object object = event.getSource();
			if (object == OKButton)
				OKButton_actionPerformed(event);
			else if (object == cancelButton)
				cancelButton_actionPerformed(event);
		}
	}

  /**
   * Return the selected day to the specific textfield which is respectively
with
   * the selected button
   * If index = 0 : set the selected day to earliestSTTextField
   * If index = 1 : set the selected day to deadlineTextField
   * If index = 2 : set the selected day to finishTimeTextField
   * @param event : button action event
   */
	void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
	{
	  if( this.scf != null ){
	    if( this.index == 0 ){
	      this.scf.earliestSTTextField.setText(theCalendar.getDate());
	    }
	    else if( this.index == 1 ){
	      this.scf.deadlineTextField.setText(theCalendar.getDate());
	    }
	    else if( this.index == 2 ){
	      this.scf.finishTimeTextField.setText(theCalendar.getDate());
	    }
```

```
    }
    setVisible(false);
    dispose();
        }

  /**
   * Exit CalendarDialog and return null
   */
        void cancelButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
    setVisible(false);
    dispose();
        }
}
```



```
package Cases;

/**
 * The main CASES frame.
 */

import JobSchedule.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.preview.JFileChooser;
import java.io.*;
import java.util.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * CasesFrame : main frame of Cases.  It is the main frame to connect to all
 * frames in Cases system.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Cases
 */
////////////////////////////////////////////////////////////////////////////////
public class CasesFrame extends com.sun.java.swing.JFrame implements
CasesTitle, I_Cases
{
        /**
         * fileChooser : instantiate new JFileChooser with the current
directory is Cases
         */
        public JFileChooser fileChooser = new
JFileChooser(CASESDIRECTORY);

  /**
   * projectAtomicVector : a vector contains all atomics in the project.
   * It is used in Job Schedule.
   */
  public Vector projectAtomicVector = new Vector();

  /**
   * projectStepContentVector : a vector contains all StepContent objects
in the project
   * It is used in Job Schedule
   */
  public Vector projectStepContentVector = new Vector();
```

```
/**
 * stepContentPathVector : a vector containts the paths of all step
contents in the project
 * It is used in Job Schedule
 */
public Vector stepContentPathVector = new Vector();

/**
 * personnelVector : a vector contains all Personnel objects in the project
 * It is used in Job Schedule
 */
public Vector personnelVector = new Vector();

/**
 * scheduleAtomicVector : a vector contains all StepContent objects
whose
 * status are "Scheduled"
 * It is used in Job Schedule
 */
public Vector scheduledAtomicVector = new Vector();

/**
 * vc : a VersionControl object
 */
public VersionControl vc = null;

/**
 * title : a title of the current menu item under SPIDER menu,
 * e.g. Edit, Decompose, Component Content, Step Content, or Trace
 */
public String title = null;

/**
 * projectName : a current project name, e.g. C4I, C3I, ...
 */
public String projectName = null;
```



```
/**
 * pathName : the absolute path of the current project, e.g. C:\Cases\C4I
 */
public String pathName = null;

///Job Schedule Variables
public Vector person_queue;
public Vector job_queue = new Vector();
public Vector job_queue1 = new Vector();
public Vector job_queue2 = new Vector();
public Vector job_pool = new Vector();
public int personid=0;
public int ctrl=0;

/**
 * Create CasesFrame
 */
        public CasesFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setJMenuBar(casesMenuBar);
                setTitle("CASES");
                getContentPane().setLayout(null);
                getContentPane().setForeground(java.awt.Color.blue);
                setSize(550,200);
                setVisible(false);
                //$$ casesMenuBar.move(0,348);
                projectMenu.setText("Project");
```

```
projectMenu.setActionCommand("Hypergraph");
projectMenu.setMnemonic((int)'P');
casesMenuBar.add(projectMenu);
createProjectMenuItem.setText("Create Project");
createProjectMenuItem.setActionCommand("Create
Project");
createProjectMenuItem.setMnemonic((int)'C');
projectMenu.add(createProjectMenuItem);
openProjectMenuItem.setText("Open Project");
openProjectMenuItem.setActionCommand("Open
Project");
openProjectMenuItem.setMnemonic((int)'O');
projectMenu.add(openProjectMenuItem);
deleteProjectMenuItem.setText("Delete Project");
deleteProjectMenuItem.setActionCommand("Delete
Project");
deleteProjectMenuItem.setMnemonic((int)'D');
projectMenu.add(deleteProjectMenuItem);
projectMenu.add(JSeparator1);
exitMenuItem.setText("Exit");
exitMenuItem.setActionCommand("Exit");
exitMenuItem.setMnemonic((int)'X');
projectMenu.add(exitMenuItem);
AVCMenu.setText("Automated Version Control");
AVCMenu.setActionCommand("Automated Version
Control");
AVCMenu.setMnemonic((int)'A');
casesMenuBar.add(AVCMenu);
AVCCreationMenuItem.setText("Create Step Version");
AVCCreationMenuItem.setActionCommand("New
Step");
AVCCreationMenuItem.setMnemonic((int)'C');
AVCMenu.add(AVCCreationMenuItem);
AVCOpenStepMenuItem.setText("Open Step Version");
AVCOpenStepMenuItem.setActionCommand("Open
Step");
AVCOpenStepMenuItem.setMnemonic((int)'O');
AVCMenu.add(AVCOpenStepMenuItem);
AVCMenu.add(JSeparator3);
AVCSplittingMenuItem.setText("Evolution History
Splitting");
AVCSplittingMenuItem.setActionCommand("New
Step");
AVCSplittingMenuItem.setMnemonic((int)'S');
AVCMenu.add(AVCSplittingMenuItem);
AVCMergingMenuItem.setText("Evolution History
Merging");
AVCMergingMenuItem.setActionCommand("New
Step");
AVCMergingMenuItem.setMnemonic((int)'M');
AVCMenu.add(AVCMergingMenuItem);
spiderMenu.setText("SPIDER");
spiderMenu.setActionCommand("Editor");
spiderMenu.setMnemonic((int)'S');
casesMenuBar.add(spiderMenu);
editMenuItem.setText("Edit");
editMenuItem.setActionCommand("jmenuItem");
editMenuItem.setMnemonic((int)'E');
spiderMenu.add(editMenuItem);
decomposeMenuItem.setText("Decompose");
decomposeMenuItem.setActionCommand("jmenuItem");
decomposeMenuItem.setMnemonic((int)'D');
spiderMenu.add(decomposeMenuItem);
spiderMenu.add(JSeparator4);
componentContentMenuItem.setText("Component
Content");
componentContentMenuItem.setActionCommand("jmenuItem");
componentContentMenuItem.setMnemonic((int)'C');
spiderMenu.add(componentContentMenuItem);
stepContentMenuItem.setText("Step Content");
stepContentMenuItem.setActionCommand("jmenuItem");
stepContentMenuItem.setMnemonic((int)'S');
spiderMenu.add(stepContentMenuItem);
```

```
spiderMenu.add(JSeparator2);
traceMenuItem.setText("Trace");
traceMenuItem.setActionCommand("jmenuItem");
traceMenuItem.setMnemonic((int)'T');
spiderMenu.add(traceMenuItem);
toolsMenu.setText("Tools");
toolsMenu.setActionCommand("Tools");
toolsMenu.setMnemonic((int)'T');
casesMenuBar.add(toolsMenu);
textEditorMenuItem.setText("Text Editor");
textEditorMenuItem.setActionCommand("Text Editor");
textEditorMenuItem.setMnemonic((int)'X');
toolsMenu.add(textEditorMenuItem);
MSWordMenuItem.setText("MS Word");
MSWordMenuItem.setActionCommand("Step
Database");
MSWordMenuItem.setMnemonic((int)'W');
toolsMenu.add(MSWordMenuItem);
MSExcelMenuItem.setText("MS Excel");
MSExcelMenuItem.setActionCommand("Component
Database");
MSExcelMenuItem.setMnemonic((int)'E');
toolsMenu.add(MSExcelMenuItem);
toolsMenu.add(JSeparator5);
netscapeMenuItem.setText("Netscape");
netscapeMenuItem.setActionCommand("Netscape");
netscapeMenuItem.setMnemonic((int)'N');
toolsMenu.add(netscapeMenuItem);
capsMenuItem.setText("CAPS");
capsMenuItem.setActionCommand("CAPS");
capsMenuItem.setMnemonic((int)'C');
toolsMenu.add(capsMenuItem);
toolsMenu.add(JSeparator6);
personnelDataMenu.setText("Personnel Data");
personnelDataMenu.setActionCommand("personnel
Data");
personnelDataMenu.setMnemonic((int)'P');
toolsMenu.add(personnelDataMenu);
addPersonnelMenuItem.setText("Add");
addPersonnelMenuItem.setActionCommand("Add");
addPersonnelMenuItem.setMnemonic((int)'A');
personnelDataMenu.add(addPersonnelMenuItem);
editPersonnelMenuItem.setText("Edit");
editPersonnelMenuItem.setActionCommand("Edit");
editPersonnelMenuItem.setMnemonic((int)'E');
personnelDataMenu.add(editPersonnelMenuItem);
deletePersonnelMenuItem.setText("Delete");
deletePersonnelMenuItem.setActionCommand("Delete");
deletePersonnelMenuItem.setMnemonic((int)'D');
personnelDataMenu.add(deletePersonnelMenuItem);
jobScheduleMenu.setText("Job Schedule");
jobScheduleMenu.setActionCommand("Tools");
jobScheduleMenu.setMnemonic((int)'J');
casesMenuBar.add(jobScheduleMenu);
jobManagementMenuItem.setText("Scheduling");
jobManagementMenuItem.setActionCommand("Text
Editor");
jobManagementMenuItem.setMnemonic((int)'S');
jobScheduleMenu.add(jobManagementMenuItem);
jobAssignMenuItem.setText("Assignment");
jobAssignMenuItem.setActionCommand("Component
Database");
jobAssignMenuItem.setMnemonic((int)'A');
jobScheduleMenu.add(jobAssignMenuItem);

imageLabel.setHorizontalAlignment(com.sun.java.swing.SwingCo
nstants.CENTER);
getContentPane().add(imageLabel);
imageLabel.setBackground(java.awt.Color.blue);
imageLabel.setBounds(18,35,125,125);
//$$ JPopupMenu1.move(24,348);
JLabel2.setText("Computer-Aided Software Evolution
System");
getContentPane().add(JLabel2);
```

```
JLabel2.setBackground(java.awt.Color.blue);
JLabel2.setForeground(java.awt.Color.blue);
JLabel2.setFont(new Font("Dialog",
Font.BOLD|Font.ITALIC, 18));
JLabel2.setBounds(155,80,380,30);
//$$ casesIimageIcon.move(72,348);
//$$ mainMenuBar.move(0,348);
//$$ JMenuBar1.move(0,348);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
createProjectMenuItem.addActionListener(lSymAction);
openProjectMenuItem.addActionListener(lSymAction);
deleteProjectMenuItem.addActionListener(lSymAction);
exitMenuItem.addActionListener(lSymAction);

AVCOpenStepMenuItem.addActionListener(lSymAction);
textEditorMenuItem.addActionListener(lSymAction);
MSWordMenuItem.addActionListener(lSymAction);
MSExcelMenuItem.addActionListener(lSymAction);
netscapeMenuItem.addActionListener(lSymAction);
traceMenuItem.addActionListener(lSymAction);
editMenuItem.addActionListener(lSymAction);
decomposeMenuItem.addActionListener(lSymAction);

componentContentMenuItem.addActionListener(lSymAction);
stepContentMenuItem.addActionListener(lSymAction);

jobManagementMenuItem.addActionListener(lSymAction);
jobAssignMenuItem.addActionListener(lSymAction);
addPersonnelMenuItem.addActionListener(lSymAction);
editPersonnelMenuItem.addActionListener(lSymAction);
deletePersonnelMenuItem.addActionListener(lSymAction);
capsMenuItem.addActionListener(lSymAction);
AVCCreationMenuItem.addActionListener(lSymAction);
AVCMergingMenuItem.addActionListener(lSymAction);
AVCSplittingMenuItem.addActionListener(lSymAction);
//}}

/**
 * Check and Create Cases and stakeholder directory
 */
if( CASESDIRECTORY.isDirectory() ){
  String[] fileList = CASESDIRECTORY.list();
  if( fileList.length > 0 ){
    setOpenDelete( true );
  }
  else if( fileList.length == 0 ){
    setOpenDelete( false );
  }
}
else{
  CASESDIRECTORY.mkdir();
  setOpenDelete( false );
}

if( !STAKEHOLDER.isDirectory() ){
  STAKEHOLDER.mkdir();
}
menuSetEnabled(false);

drawIcon();
}

/**
 * Adding cases.gif logo to CasesFrame
 */
public void drawIcon(){
```

```
    try{
      FileInputStream imageFile = new FileInputStream("cases.gif");
      byte[] data = new byte[3000];
      imageFile.read(data);
      ImageIcon icon = new ImageIcon(data);
      imageLabel.setIcon(icon);
    }catch (Exception e){ System.out.println(e);}
  }

      public CasesFrame(String sTitle)
      {
            this();
    setTitle(sTitle);
      }

      public void setVisible(boolean b)
      {
            if (b)
                  setLocation(50, 50);
            super.setVisible(b);
      }

      static public void main(String args[])
      {
            (new CasesFrame()).setVisible(true);
      }

      public void addNotify()
      {
            // Record the size of the window prior to calling parents
addNotify.
            Dimension size = getSize();

            super.addNotify();

            if (frameSizeAdjusted)
                  return;
            frameSizeAdjusted = true;

            // Adjust size of frame according to the insets and menu
bar
            Insets insets = getInsets();
            com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
            int menuBarHeight = 0;
            if (menuBar != null)
                  menuBarHeight =
menuBar.getPreferredSize().height;
            setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
      }

      // Used by addNotify
      boolean frameSizeAdjusted = false;

      //{{DECLARE_CONTROLS
      com.sun.java.swing.JLabel imageLabel = new
com.sun.java.swing.JLabel();
      com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
      com.sun.java.swing.JMenuBar casesMenuBar = new
com.sun.java.swing.JMenuBar();
      com.sun.java.swing.JMenu projectMenu = new
com.sun.java.swing.JMenu();
      com.sun.java.swing.JMenuItem createProjectMenuItem = new
com.sun.java.swing.JMenuItem();
      com.sun.java.swing.JMenuItem openProjectMenuItem = new
com.sun.java.swing.JMenuItem();
      com.sun.java.swing.JMenuItem deleteProjectMenuItem = new
com.sun.java.swing.JMenuItem();
      com.sun.java.swing.JSeparator JSeparator1 = new
com.sun.java.swing.JSeparator();
      com.sun.java.swing.JMenuItem exitMenuItem = new
com.sun.java.swing.JMenuItem();
```

```
com.sun.java.swing.JMenu AVCMenu = new
com.sun.java.swing.JMenu();
com.sun.java.swing.JMenuItem AVCCreationMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem AVCOpenStepMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JSeparator JSeparator3 = new
com.sun.java.swing.JSeparator();
com.sun.java.swing.JMenuItem AVCSplittingMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem AVCMergingMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenu spiderMenu = new
com.sun.java.swing.JMenu();
com.sun.java.swing.JMenuItem editMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem decomposeMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JSeparator JSeparator4 = new
com.sun.java.swing.JSeparator();
com.sun.java.swing.JMenuItem componentContentMenuItem =
new com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem stepContentMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JSeparator JSeparator2 = new
com.sun.java.swing.JSeparator();
com.sun.java.swing.JMenuItem traceMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenu toolsMenu = new
com.sun.java.swing.JMenu();
com.sun.java.swing.JMenuItem textEditorMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem MSWordMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem MSExcelMenuItem = new
com.sun.java.swing.JMenuItem();
```



```
com.sun.java.swing.JSeparator JSeparator5 = new
com.sun.java.swing.JSeparator();
com.sun.java.swing.JMenuItem netscapeMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem capsMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JSeparator JSeparator6 = new
com.sun.java.swing.JSeparator();
com.sun.java.swing.JMenu personnelDataMenu = new
com.sun.java.swing.JMenu();
com.sun.java.swing.JMenuItem addPersonnelMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem editPersonnelMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem deletePersonnelMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenu jobScheduleMenu = new
com.sun.java.swing.JMenu();
com.sun.java.swing.JMenuItem jobManagementMenuItem = new
com.sun.java.swing.JMenuItem();
com.sun.java.swing.JMenuItem jobAssignMenuItem = new
com.sun.java.swing.JMenuItem();
//}}

//{{DECLARE_MENUS
//}}


class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent
event)
        {
                Object object = event.getSource();
                if (object == createProjectMenuItem)

createProjectMenuItem_actionPerformed(event);
```

```
                else if (object == openProjectMenuItem)

openProjectMenuItem_actionPerformed(event);
                else if (object == deleteProjectMenuItem)

deleteProjectMenuItem_actionPerformed(event);
                else if (object == exitMenuItem)
                        exitMenuItem_actionPerformed(event);
                else if (object == AVCOpenStepMenuItem)

AVCOpenStepMenuItem_actionPerformed(event);

                else if (object == AVCCreationMenuItem)

AVCCreationMenuItem_actionPerformed(event);
                else if (object == AVCMergingMenuItem)

AVCMergingMenuItem_actionPerformed(event);
                else if (object == AVCSplittingMenuItem)

AVCSplittingMenuItem_actionPerformed(event);
                else if (object == editMenuItem)
                        editMenuItem_actionPerformed(event);
                else if (object == decomposeMenuItem)

decomposeMenuItem_actionPerformed(event);
                else if (object == componentContentMenuItem)

componentContentMenuItem_actionPerformed(event);
                else if (object == stepContentMenuItem)

stepContentMenuItem_actionPerformed(event);
                else if (object == traceMenuItem)

traceMenuItem_actionPerformed(event);
                else if (object == textEditorMenuItem)

textEditorMenuItem_actionPerformed(event);
                else if (object == MSWordMenuItem)

MSWordMenuItem_actionPerformed(event);
                else if (object == MSExcelMenuItem)

MSExcelMenuItem_actionPerformed(event);
                if (object == netscapeMenuItem)

netscapeMenuItem_actionPerformed(event);
                else if (object == capsMenuItem)

capsMenuItem_actionPerformed(event);
                else if (object == addPersonnelMenuItem)

addPersonnelMenuItem_actionPerformed(event);
                else if (object == editPersonnelMenuItem)

editPersonnelMenuItem_actionPerformed(event);
                else if (object == deletePersonnelMenuItem)

deletePersonnelMenuItem_actionPerformed(event);
                else if (object == jobManagementMenuItem)

jobManagementMenuItem_actionPerformed(event);
                else if (object == jobAssignMenuItem)

jobAssignMenuItem_actionPerformed(event);
        }
}

/**
 * Ask a user to enter the new project name.
 * Do not allow a duplicate project name.
 * If the name is not duplicate, create the new project and connect to
 * ProjectSchemaFrame directly.
```

```
    *
    * @param event : action event from Create Project menu item
    */
        void
createProjectMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
          this.projectName = JOptionPane.showInputDialog( this, "Project
Name",
                "Create
Project",JOptionPane.INFORMATION_MESSAGE);
      if( this.projectName != null ){
        String[] theList = CASESDIRECTORY.list();
        if( theList.length == 0 ){
          this.setOpenDelete( true );
                File newFile = new
File(CASESDIRECTORY,this.projectName);
                newFile.mkdir();
        if( newFile.isDirectory() ){
                        this.pathName = newFile.getAbsolutePath();
            (new ProjectSchemaFrame(this.projectName,
this.pathName)).setVisible(true);
            menuSetEnabled(true);
          }
        }
        else if( theList.length>0 ){
          for( int i=0; i<theList.length; i++ ){

            // Checking the duplication before create the new project
            if( this.projectName.equals(theList[i]) ){
              i= theList.length;
              JOptionPane.showMessageDialog(this, this.projectName+"
has been created.",
                        "Error Message",
JOptionPane.ERROR_MESSAGE);
              return;
            }
```

```
            else if( i==(theList.length-1) ){
              this.setOpenDelete( true );
                File newFile = new
File(CASESDIRECTORY,this.projectName);
                newFile.mkdir();
                if( newFile.isDirectory() ){
                        this.pathName = newFile.getAbsolutePath();
                    (new ProjectSchemaFrame(this.projectName,
this.pathName)).setVisible(true);
                menuSetEnabled(true);
              }
            }
          }
        }
      }
    }

  /**
    * Allow a user to select a project, then he has a choice
    * Yes : connect to ProjectSchemaFrame to edit step.cfg, component.cfg,
loop.cfg,
    * or dependency.cfg file.
    * No : go straight to main menu (Automatic Version Control, SPIDER,
Tools, or Job Schedule
    *
    * @param event : action event from Open Project menu item
    */
        void
openProjectMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
            int returnValue=-1;
          if( this.projectName != null ){
            File newFile = new
File(CASESDIRECTORY,this.projectName);
            newFile.mkdir();
            this.fileChooser = new JFileChooser(newFile);
          }
```

```
            this.fileChooser.setDialogTitle("Open Project");
        this.fileChooser.setCurrentDirectory(CASESDIRECTORY);

this.fileChooser.setFileSelectionMode(this.fileChooser.DIRECTORIES_O
NLY);
            repaint();
        returnValue = this.fileChooser.showDialog(this,"Open");
            if( returnValue == fileChooser.APPROVE_OPTION ){

                    File file = fileChooser.getCurrentDirectory();

                    this.pathName = file.getAbsolutePath();
        int i = JOptionPane.showConfirmDialog(this, "Do you want to open
Project Schema?",
                "Confirmation", JOptionPane.YES_NO_OPTION);
        if( i==0 ){
                        (new ProjectSchemaFrame(this.projectName,
this.pathName)).setVisible(true);
                    }
        menuSetEnabled(true);
    }
        }

  /**
   * Connect to DeleteDialog, where allows a user to delete a project
   * and all existing project names are in the combo box.
   * Except, the current project can not be deleted since it is occuping.
   *
   * @param event : action event from Delete Project menu item
   */
        void
deleteProjectMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
    (new DeleteDialog(this, "Delete Project")).setVisible(true);
        }
```

```
  /**
   * Exit CasesFrame without saving or any notice
   *
   * @param event : action event from Exit menu item
   */
        void exitMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
                System.exit(0);
                dispose();
        }

  /**
   * Connect to AVCCreateStepFrame
   *
   * @param event : action event from Create Step Version
   */
        void
AVCCreationMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
                (new AVCCreateStepFrame(this.pathName,
this.projectName)).setVisible(true);
        }

  /**
   * Connect to AVCOpenStepFrame
   *
   * @param event : action event from Open Step Version
   */
        void
AVCOpenStepMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
                (new AVCOpenStepFrame(this.projectName,
this.pathName)).setVisible(true);
        }
```

```
/**
 * Connect to AVCEHSplittingFrame
 *
 * @param event : action event from Evolution History Splitting
 */
        void
AVCSplittingMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
                (new AVCSplittingFrame(this.pathName,
this.projectName)).setVisible(true);
        }

/**
 * Connect to AVCEHMergingFrame
 *
 * @param event : action event from Evolution History Merging
 */
        void
AVCMergingMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
                (new AVCMergingFrame(this.pathName,
this.projectName)).setVisible(true);
        }

/**
 * Get the current versionControl object and will connect to
EditDecomposeFrame
 *
 * @param event : action event from Edit menu item
 */
        void editMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
          this.title = EDIT_TITLE;
          getVersionControl();
        }

/**
 * Get the current versionControl object and will connect to
EditDecomposeFrame
 *
 * @param event : action event from Decompose menu item
 */
        void
decomposeMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
          this.title = DECOMPOSE_TITLE;
          getVersionControl();
        }

/**
 * Get the current versionControl object and will connect to
ComponentContentFrame
 *
 * @param event : action event from Component Content menu item
 */
        void
componentContentMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
          this.title = COMPONENT_CONTENT_TITLE;
          getVersionControl();
        }

/**
 * Get the current versionControl object and will connect to
StepContentFrame
 *
 * @param event : action event from Step Content menu item
 */
```

```
        void
stepContentMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
          this.title = STEP_CONTENT_TITLE;
          getVersionControl();
        }

  /**
   * Get the current versionControl object and will connect to TraceFrame
   *
   * @param event : action event from Trace menu item
   */
        void traceMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
          this.title = TRACE_TITLE;
          getVersionControl();
        }

  /**
   * Connect to notepad editor
   *
   * @param event : action event from Text menu item
   */
        void
textEditorMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
     try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(NOTEPAD);
     }
     catch( Exception e ) {
        throw new RuntimeException(e.toString());
     }
        }

  /**
   * Connect to Microsoft Word editor
   *
   * @param event : action event from MS Word menu item
   */
        void
MSWordMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
     try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(WINWORD);
     }
     catch( Exception e ) {
        throw new RuntimeException(e.toString());
     }
        }

  /**
   * Connect to Microsoft Excel editor
   *
   * @param event : action event from MS Excel menu item
   */
        void
MSExcelMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
     try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(EXCEL);
     }
     catch( Exception e ) {
        throw new RuntimeException(e.toString());
     }
        }

  /**
   * Connect to Netscape environment
   *
   * @param event : action event from Netscape menu item
```

```
    */
        void
netscapeMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
    try {
      Runtime runtime = Runtime.getRuntime();
      runtime.exec(NETSCAPE);
    }
    catch( Exception e ) {
      throw new RuntimeException(e.toString());
    }
        }

  /**
    * Connect to Caps environment
    *
    * @param event : action event from Caps menu item
    */
        void capsMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
    try {
      Runtime runtime = Runtime.getRuntime();
      runtime.exec(CAPS);
    }
    catch( Exception e ) {
      throw new RuntimeException(e.toString());
    }
        }

  /**
    * Connect to PersonnelFrame
    *
    * @param event : action event from Personnel - Add menu item
    */
        void
addPersonnelMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
          (new PersonnelFrame()).setVisible(true);
        }

  /**
    * Go to JFileChooser to choose a file before connect to PersonnelFrame
    *
    * @param event : action event from Personnel - Edit menu item
    */
        void
editPersonnelMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
    FileDialog fd = new FileDialog(this, "Open", FileDialog.LOAD);
    fd.setDirectory(STAKEHOLDER.getAbsolutePath());
    fd.show();
    String fileName = fd.getFile();
    if( fileName != null ){
      String filePath = STAKEHOLDER.getAbsolutePath()+fileName;
          (new PersonnelFrame(filePath, "Edit")).setVisible(true);
    }
        }

  /**
    * Connect to DeleteDialog and it has a Browse button to select a file
    * under stakeholder directory
    *
    * @param event : action event from Personnel - Delete menu item
    */
        void
deletePersonnelMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
```

```
            (new DeleteDialog(this, "Delete Personnel
Data")).setVisible(true);
        }

    /**
     * Short cut to print the output
     * @param string : the output string
     */
        public void debug( String string ){
            System.out.println(string);
        }

    /**
     * Gray out Open Project and Delete Project menu items
     *    if there is no project under CASESDIRECTORY .
     * Or not gray out if CASESDIRECTORY exists at least one project
     */
        public void setOpenDelete( boolean enabled ){
            this.openProjectMenuItem.setEnabled( enabled );
            this.deleteProjectMenuItem.setEnabled( enabled );
        }

    /**
     * If this.projectName is null, flag = false
     * Else flag = true
     */
        void menuSetEnabled( boolean flag){
      this.AVCMenu.setEnabled( flag );
            this.spiderMenu.setEnabled( flag );
            this.toolsMenu.setEnabled( flag );
            this.jobScheduleMenu.setEnabled(flag);
        }

    /**
     * Get VersionControl object from current.vsn file
     */
        public void getVersionControl(){
        try{
            FileInputStream fileInput = new
FileInputStream(this.pathName+"\\current.vsn");
                ObjectInputStream current = new ObjectInputStream(
fileInput );
                if( current != null ){
                    this.vc = (VersionControl) current.readObject();
                }
                current.close();
                fileInput.close();
            }
            catch( ClassNotFoundException e ){
debug("ClassNotFoundException: "+e); }
            catch(IOException i){
        JOptionPane.showMessageDialog(this, "Can not open the
application \nsince current.vsn does not exist in this project.",
                        "Error Message",
JOptionPane.ERROR_MESSAGE);
        return;
            }

            if( this.vc != null ){
        directoryTree();
        }
            }

    /**
     * Check and locate the current step in fileChooser
     * If the selected step is not a current step, Error Message shows up
     */
        public void directoryTree(){
            String currentStep = (String)this.vc.getCurrentStep();
            String currentVersion = (String)this.vc.getCurrentVersion();
            String wholePath = this.pathName +
"\\"+currentStep+"\\"+currentVersion;
            File file = new File(wholePath);
                int returnValue=-1;
```

```
    this.fileChooser = new JFileChooser(file);
        this.fileChooser.setDialogTitle("Directory Tree");
    this.fileChooser.setCurrentDirectory(file);

this.fileChooser.setFileSelectionMode(this.fileChooser.DIRECTORIES_O
NLY);
    returnValue = this.fileChooser.showDialog(this,"Open");

        if( returnValue == this.fileChooser.APPROVE_OPTION ){

                File aFile = this.fileChooser.getCurrentDirectory();


                String absPath = aFile.getAbsolutePath();
                String output =
currentStep.substring((int)currentStep.indexOf("-")+1)+currentVersion;
                if( absPath.equals( wholePath ) ){
                  currentDir(absPath);
                }
                else if( absPath.length() > wholePath.length() ){
                  subDir(absPath);
                }
                else{
        JOptionPane.showMessageDialog(this, "The selected directory is
not the current step.",
                    "Error Message",
JOptionPane.ERROR_MESSAGE);
                }
    }
        }

  /**
   * Locate the selected step and it is an original step
   *
   * @param absPath : a string of the whole path
   */
  public void currentDir(String absPath ){
        String currentStep = (String)this.vc.getCurrentStep();
        String currentVersion = (String)this.vc.getCurrentVersion();
            String stepName = currentStep + currentVersion;
        String output =
currentStep.substring((int)currentStep.indexOf("-")+1)+currentVersion;

            if( this.title.equals(EDIT_TITLE) ){
              (new EditDecomposeFrame(this.title,stepName,
output,absPath)).setVisible(true);
            }
            else if( this.title.equals(DECOMPOSE_TITLE) ){
              int theMax = (int)findMax(absPath) +1 ;
      stepName = stepName + "-" + theMax;
              String decomposeOutput = stepName.substring(2);
              (new EditDecomposeFrame(this.title,stepName,

decomposeOutput,absPath,theMax)).setVisible(true);
            }
            else if(
this.title.equals(COMPONENT_CONTENT_TITLE) ){
              setComponentContent( stepName, absPath);
            }
            else if( this.title.equals(STEP_CONTENT_TITLE) ){
      setStepContent(null, stepName, absPath);
        }
            else if( this.title.equals(TRACE_TITLE) ){
      setTraceFrame(stepName, absPath);
        }
  }

  /**
   * Locate the selected step and it is a decompose step
   *
   * @param absPath : a string of the whole path
   */
  public void subDir(String absPath ){
        String currentStep = (String)this.vc.getCurrentStep();
```

```
            String currentVersion = (String)this.vc.getCurrentVersion();
            String wholePath = this.pathName +
"\\"+currentStep+"\\"+currentVersion;
                    String newSubString = absPath.substring(0,
wholePath.length());
            String output =
currentStep.substring((int)currentStep.indexOf("-")+1)+currentVersion;

                    int result = wholePath.compareTo(newSubString);
                    if( result == 0 ){
                      String subName =
absPath.substring(wholePath.length()+1);
                      StringTokenizer st = new
StringTokenizer(subName,"\\");
                      Vector stVector = new Vector();
                      while( st.hasMoreTokens()){
                        stVector.addElement(st.nextToken());
                      }
                      String stResult = (String)stVector.elementAt(0);
                      for( int i=1; i<stVector.size(); i++){
                        stResult =
stResult+"."+(String)stVector.elementAt(i);
                      }
                      output = output + "-" + stResult;
                      String stepName = currentStep+currentVersion+"-
"+stResult;

                      if( this.title.equals(EDIT_TITLE) ){
                      (new EditDecomposeFrame(this.title, stepName,
output,absPath)).setVisible(true);
                      }

                      else if( this.title.equals(DECOMPOSE_TITLE) ){
          int theMax = (int)findMax(absPath) + 1;
                        stepName = stepName + "." +theMax;
                        String decomposeOutput = stepName.substring(2);
                      (new EditDecomposeFrame(this.title, stepName,
decomposeOutput, absPath, theMax)).setVisible(true);
                      }

                      else if(
this.title.equals(COMPONENT_CONTENT_TITLE) ){
                        setComponentContent( stepName, absPath);
                      }
                      else if( this.title.equals(STEP_CONTENT_TITLE) ){

                        setStepContent(null, stepName, absPath);
                      }
                      else if( this.title.equals(TRACE_TITLE) ){
            setTraceFrame(stepName, absPath);
          }
                    }
                    else{
          JOptionPane.showMessageDialog(this, "The selected directory is
not the current step.",
                      "Error Message",
JOptionPane.ERROR_MESSAGE);
                    }
  }

  /**
   * Return a vector of all components of the current step
   *
   * @param stepName : the selected step
   * @param absPath : the complete path of this stepName
   * @return componentVector : a vector of strings holds component
names
   */
  public Vector getComponents(String stepName, String absPath){
                    Vector componentVector = new Vector();
    String s = stepName.substring(2);

                    componentVector.addElement(s);
```

```
                try{
                  FileInputStream fileInput = new
FileInputStream(absPath+"\\input.p");
                  DataInputStream inputP = new
DataInputStream(fileInput);
                  if( inputP != null ){
                    String sP = inputP.readLine();
                    if(( sP !=  null) && (!sP.equals("")) ){
                      tokenizer(componentVector, sP);
                    }
                  }
                  inputP.close();
                  fileInput.close();


                  fileInput = new FileInputStream(absPath+"\\input.s");
                  DataInputStream inputS = new
DataInputStream(fileInput);
                  if( inputS != null ){
                    String sS = inputS.readLine();
                    if( (sS != null) && (!sS.equals("")) ){
                      tokenizer(componentVector, sS);
                    }
                  }
                }
                catch( IOException e ){
                  debug("IOException at ComponentContent: "+e);
                }
    return componentVector;
  }

  /**
   * Connect to ComponentContentFrame
   *
   * @param stepName : selected step name, e.g., s-I, 1.1, 1, ...
   * @param absPath : the absolute path of  stepName, e.g.,F:\Cases\s-
I\1.1
   */
        public void setComponentContent( String stepName, String
absPath ){
        (new
ComponentContentFrame(this.pathName,stepName.substring(2),
getComponents(stepName, absPath),
(Vector)fileNameList())).setVisible(true);
        }

  /**
   * Connect to StepContentFrame
   *
   * @param traceFrame : instantiation of TraceFrame
   * @param stepName : selected step name, e.g., s-I, 1.1, 1, ...
   * @param absPath : the absolute path of  stepName, e.g.,F:\Cases\s-
I\1.1
   */
  public void setStepContent(TraceFrame traceFrame, String stepName,
String absPath ){
    File file = new File(absPath);
    if( file.isDirectory() ){
      if( !(file.getName()).equals(COMPONENT_CONTENT_DIR) ){
        Vector atomicsVector = new Vector();
        File temp = new File(this.pathName, this.vc.getCurrentStep());
        String[] theFiles = temp.list();
        for( int i=0; i<theFiles.length; i++ ){
          getAtomics(new File(temp, theFiles[i]), atomicsVector);
        }
            (new StepContentFrame(traceFrame, stepName, absPath,
                                  atomicsVector )).setVisible(true);

          }
      else{
        JOptionPane.showMessageDialog(this, absPath+" is not a step.
Please reselect it!",
                    "Warning
Message",JOptionPane.ERROR_MESSAGE);
```

```
            return;
          }
        }
      else{
        JOptionPane.showMessageDialog(this, absPath+" is not a step.
Please reselect it!",
                    "Warning
Message",JOptionPane.ERROR_MESSAGE);
        return;
      }
    }

    /**
     * Connect to TraceFrame
     *
     * @param stepName : selected step name, e.g., s-I, 1.1, 1, ...
     * @param absPath : the absolute path of  stepName, e.g.,F:\Cases\s-
I\1.1
     */
    public void setTraceFrame(String stepName, String absPath){
      (new TraceFrame(this, this.pathName, getComponents(stepName,
absPath), (Vector)fileNameList())).setVisible(true);
    }

    /**
     * Get all atomics of the selected step
     *
     * @param aFile : since a file is a step, aFile is a selected step
     * @param storedVector : a vector of atomics
     */
    public void getAtomics(File aFile, Vector storedVector){
      boolean isAtomic = true;
      if( (aFile.isDirectory()) &&
!((aFile.getName()).equals(COMPONENT_CONTENT_DIR)) ){
        String[] theFiles = aFile.list();
        for( int i=0; i<theFiles.length; i++ ){
          char c = (char)((String)theFiles[i]).charAt(0);
          Character theChar = new Character(c);
          if(theChar.isDigit(c)){
            File f1 = new File( aFile, theFiles[i]);
            if( (f1.isDirectory()) &&
!((f1.getName()).equals(COMPONENT_CONTENT_DIR)) ){
              isAtomic = false;
              getAtomics(new File(aFile, theFiles[i]), storedVector );
            }
          }
        }
        if( isAtomic ){
          storedVector.addElement(aFile);
        }
      }
    }

    /**
     * Find a maximum number of the selected step version number
     *
     * @return theMax : an integer, the maximum number of the selected
step
     * @param thePath : the path of selected step, e.g., F:\Cases\s-I\1.1\1\2\1
     */
        public int findMax( String thePath ){
          Vector v = new Vector();
              File f1 = new File( thePath );
              String[] list = (String[]) f1.list();
              for(int i=0; i<list.length;i++){
                File f2 = new File(thePath, list[i]);
                if( f2.isDirectory() ){
                  try{
            Integer i1 = new Integer(list[i]);
            v.addElement(i1);
                  }
                  catch(NumberFormatException n){
                  }
        }
```

```
                }
                int theMax = 0;
                for(int i=0; i<v.size(); i++){
        theMax =
Math.max(theMax,(int)((Integer)v.elementAt(i)).intValue());
                }
            return theMax;
        }

    /**
     * Token a string with delimit is ","
     *
     * @param v : a vector of tokenizer string
     * @param s : a tokenized string
     */
        public void tokenizer( Vector v, String s){
            StringTokenizer st = new StringTokenizer(s,",");
            while(st.hasMoreTokens()){
                String theString = (st.nextToken()).trim();
                v.addElement(theString);
            }
        }

    /**
     * List all steps of a project
     * @return a vector of string with all steps
     */
        public Vector fileNameList(){
            Vector v = new Vector();
      String[] theList = (String[])(new File(this.pathName)).list();
      for( int i=0; i<theList.length;i++){
        if( (new File(this.pathName+"\\"+theList[i])).isDirectory()){
          v.addElement(theList[i]);
        }
      }
      return v;
    }
```

```
    /**
     * Save all personnel object after updating
     */
        public void savePersonnelVector(Vector v){
            personnelVector = v;
            if( personnelVector != null ){
        for( int i=0; i<personnelVector.size(); i++ ){
            Personnel personnel = (Personnel)personnelVector.elementAt(i);
            if( personnel != null ){
                try{
                    FileOutputStream fileOutput = new
FileOutputStream(STAKEHOLDER+"\\"+personnel.getID());
                    ObjectOutputStream oo = new
ObjectOutputStream(fileOutput);
                    if( oo != null ){
                        oo.writeObject(personnel);
                    }
                    oo.flush();
                    oo.close();
                    fileOutput.close();
                }
                catch(IOException io){
                    JOptionPane.showMessageDialog(this, "Sorry, saving is
unsuccessful", "Error Message", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }
        }

    /**
     * Return a vector of personnel objects under stakeholder directory
     */
        public Vector getPersonnelVector(){
            personnelVector = new Vector();
            String[] list = (String[])STAKEHOLDER.list();
```

```
            if( list.length > 0 ){
               for( int i=0; i<list.length; i++ ){
                        try{
             String filePath = STAKEHOLDER.getAbsolutePath()+list[i];
                           File aFile = new File(filePath);
                           if( aFile.exists() ){
                              FileInputStream fileInput =  new
FileInputStream(aFile);
                              ObjectInputStream oi = new
ObjectInputStream(fileInput);
                              if( oi != null ){
                                 Personnel personnel =
(Personnel)oi.readObject();
                              if( personnel != null ){
                        personnelVector.addElement(personnel);
                                 }
                              }
                              oi.close();
                              fileInput.close();
            }
                           }
                           catch( IOException io ){
                              System.out.println(io);
                           }
                           catch( ClassNotFoundException c ){
                              debug("ClassNotFoundException: "+c);
                           }
               }
            }
            return personnelVector;
         }

   /**
    * Return a vector of all atomics in the selected project
    */
         public Vector getAllAtomics(){
      File aProject = new File(this.pathName);  //current project
      projectAtomicVector = new Vector();
      if( aProject.isDirectory() ){
         String[] stepList = aProject.list();  //List all the steps of the current
project
         for( int i=0; i<stepList.length; i++ ){
            File aFile = new File(aProject,stepList[i]);  //searching for steps
only
            if( aFile.isDirectory() ){
               File temp = new File(aProject, stepList[i]);
               String[] theFiles = temp.list();
               for( int j=0; j<theFiles.length; j++ ){
                  getAtomics(new File(temp, theFiles[j]),
projectAtomicVector);
               }
            }
         }
      }
            return projectAtomicVector;
         }

   /**
    * Save all stepContent objects after updating
    */
         public void saveStepContentVector(Vector v){
            projectStepContentVector = v;
            if( projectStepContentVector != null ){
               for( int i=0; i<projectStepContentVector.size(); i++ ){
            try{
               String path = (String)stepContentPathVector.elementAt(i);
               FileOutputStream fileOutput = new FileOutputStream(path);
               ObjectOutputStream oo = new
ObjectOutputStream(fileOutput);
               if( oo != null ){

oo.writeObject((StepContent)projectStepContentVector.elementAt(i));
               }
               oo.flush();
```

```
            oo.close();
            fileOutput.close();
          }
          catch(IOException io){
            debug("IOException: "+io);
          }
              }
            }
          }

  /**
   * Return a vector of stepContent objects in the selected project
   */
        public Vector getStepContentVector(){
    int stepContentIndex = 0;

    stepContentPathVector = new Vector();
            projectAtomicVector = (Vector)getAllAtomics();
            projectStepContentVector = new Vector();

            if( projectAtomicVector != null ){
        for( int j=0; j<projectAtomicVector.size(); j++ ){
          File theFile = new
File((File)projectAtomicVector.elementAt(j),"step.cnt");  //Example to get
the atomic at element 4 of projectAtomicVector
          if( theFile.exists() ){
            try{
              FileInputStream fileInput = new FileInputStream(theFile);
              ObjectInputStream oo = new ObjectInputStream(fileInput);
              if( oo != null ){
                StepContent st = (StepContent)oo.readObject();
                if( st!= null ){
                  projectStepContentVector.insertElementAt(st,
stepContentIndex); //Step content vector of all atomics in the project

stepContentPathVector.insertElementAt(theFile.getAbsolutePath(),
stepContentIndex++);
                }
              }
              oo.close();
              fileInput.close();
            }
            catch(IOException io){
              debug("IOException: "+io);
            }
            catch( ClassNotFoundException c ){
              debug("ClassNotFoundException: "+c);
            }
          }
        }
      }

      return projectStepContentVector;
    }

  /**
   * Return a vector of atomics with the status is "Scheduled"
   */
        public Vector getScheduledAtomicVector(){
          scheduledAtomicVector = new Vector();
    if( projectStepContentVector != null ){
      for( int i=0; i<projectStepContentVector.size(); i++ ){
        StepContent st =
(StepContent)projectStepContentVector.elementAt(i);
        if( st != null ){
          String status = ((String)st.getStatus()).trim();
          if( status.equals("Approved") ){
            st.setStatus("Scheduled");
            scheduledAtomicVector.addElement(st);
          }
          else if( status.equals("Scheduled") ){
            scheduledAtomicVector.addElement(st);
          }
        }
```

```
        }
      }
      return scheduledAtomicVector;
        }
  ////////////////////////////////// JOB SCHEDULE ////////////////////////////////
  /**
   * Job Management
   */
        void
jobManagementMenuItem_actionPerformed(java.awt.event.ActionEvent
event)
        {
          person_queue = (Vector)getPersonnelVector();
          job_pool = (Vector)getStepContentVector();
          for(int i=0; i<job_pool.size(); i++){
           StepContent step=(StepContent)job_pool.elementAt(i);
          }
    Predecessor();

    //cleanperson_job();//each time will kill all of person's job
    checking_person();

    if(job_queue.size()>0){
       (new JFrame_manage(job_queue)).setVisible(true);
    }
    else{
       (new JDialog_message()).setVisible(true);
    }

        }

  /**
   * Job Assignment
   */
        void
jobAssignMenuItem_actionPerformed(java.awt.event.ActionEvent event)
        {
          if(person_queue.size()>0 && job_queue.size()>0){
            (new JFrame_assignjob(this, person_queue, job_queue,
job_pool)).setVisible(true);
          }
          else{
            (new JDialog_message1()).setVisible(true);
          }
  }

        void Predecessor(){
          job_queue.removeAllElements();
          job_queue1.removeAllElements();
          job_queue2.removeAllElements();
          for(int i=0; i<job_pool.size(); i++){
            StepContent step=(StepContent)job_pool.elementAt(i);
            if(step.getStatus().equals("Approved")
||step.getStatus().equals("Scheduled")){
              job_queue1.addElement(step);
            }
          }
          for(int i=0; i<job_queue1.size(); i++){
            StepContent step=(StepContent) job_queue1.elementAt(i);
            step.setStatus("Scheduled");
            setstep(step.getStepName(), step);

            Vector v=(Vector) step.getPredecessors();

      if(v!=null){
        int n=0;
        n=checkpredecesor(step);
        if(n>0){
          this.job_queue.addElement(step);
        }
        else{
          this.job_queue2.addElement(step);
        }
      }
```

```
        else{
          this.job_queue.addElement(step);
        }
      }//end for_loop
        }

        int checkpredecesor(StepContent step){
          Vector v=(Vector) step.getPredecessors();
          if(v.size()>0){
            for(int i=0; i<v.size(); i++){
              String predecesor=(String) v.elementAt(i);
              for(int j=0; j<this.job_queue1.size(); j++){
                StepContent step2=(StepContent)
this.job_queue1.elementAt(j);
                if(predecesor.equals(step2.getStepName())){
                return 0;
                }
              }

            }//end for_loop
          }//end if
      return 1;
        }

        void checking_person(){
          for(int i=0; i<person_queue.size(); i++){
            Personnel person=(Personnel)person_queue.elementAt(i);
            if(person!=null){
              Vector major=(Vector) person.getMajorJobs();

              if(major!=null){
                for(int j=0; j<major.size(); j++){
                  if(major!=null){
                  StepContent step=(StepContent) major.elementAt(j);
                  if(step!=null){
                    if( check_status(step.getStepName())==1){
                      major.removeElement(step);
                    }
                  }
                  }//end if(major!=null)
                }//end for_loop(j)
              }

              //check minorJob second
              Vector minor=(Vector) person.getMinorJobs();
              if(minor!=null){
                for(int k=0; k<minor.size(); k++){
                  StepContent step=(StepContent) minor.elementAt(k);
                  if(step!=null){
                    if( check_status(step.getStepName())==1){
                      minor.removeElement(step);
                    }
                  }
                }//end for_loop(k)
              }
            }//end if(person!=null)

          }// end for_loop(i)
        }//end check_person

        int check_status(String stepname){
          for(int i=0; i<job_pool.size(); i++){
            if(job_pool!=null){
              StepContent step=(StepContent) job_pool.elementAt(i);
              if(step!=null){
                if(step.getStepName().equals(stepname)){
                  if(step.getStatus().equals("Completed")){
              return 1;
                  }
                }
              }
            }
          }//end for_loop
          return 0;
```

```
    }

    void cleanperson_job(){
      try{
        for(int i=0; i<person_queue.size(); i++){
          Personnel p=(Personnel) person_queue.elementAt(i);
          Vector v1=p.getMajorJobs();
          Vector v2=p.getMinorJobs();

          v1.removeAllElements();
          p.setMajorJobs(v1);
          v2.removeAllElements();
          p.setMinorJobs(v2);
          int personid=Integer.parseInt(p.getID());
          setperson_queue(personid, p);
        }
        savePersonnelVector(person_queue);
}
catch(Exception e){}
    }
    void setperson_queue(int personid, Personnel p1){
      try{
        for(int i=0; i<person_queue.size(); i++){
          Personnel p=(Personnel) person_queue.elementAt(i);
          if(personid==Integer.parseInt(p.getID())){
            person_queue.setElementAt(p1, i);
          }
        }//end for_loop
}
catch(Exception e){}
    }

    void setstep(String stepname, StepContent step1){
      for(int i=0; i<job_pool.size(); i++){
        StepContent step=(StepContent) job_pool.elementAt(i);
        if(step.getStepName().equals(stepname)){
          job_pool.setElementAt(step1, i);
        }
      }
    }
  ////////////////////////////////// END JOB SCHEDULE ////////////////////////////////

}



package Cases;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

//////////////////////////////////////////////////////////////////////////////////
/**
 * CasesTitle : contains all global variables in Cases package.
 * All the titles, directories, or the other names are located in this class
 */
//////////////////////////////////////////////////////////////////////////////////
public interface CasesTitle{

        //Locations of Cases and stakeholder directory
        static final File CASESDIRECTORY = new
File("F:\\nps\\Thesis\\CasesJobSchedule_10_30_99\\cases_le11\\data\\cases
\\");
        static final File STAKEHOLDER = new
File("F:\\nps\\Thesis\\CasesJobSchedule_10_30_99\\cases_le11\\data\\stake
holder\\");

  //Locations of netscape, notepad, winword, excel, and caps
  static final String NETSCAPE = "C:\\Program
Files\\Netscape\\Communicator\\Program\\netscape.exe";
```

```
   static final String NOTEPAD = "notepad.exe";
   static final String WINWORD = "C:\\Program Files\\Microsoft
Office\\Office\\Winword.exe";
   static final String EXCEL = "C:\\Program Files\\Microsoft
Office\\Office\\Excel.exe";
   static final String CAPS = "caps.bat";
   static final String[] EXECUTIONS = {NOTEPAD, WINWORD,
EXCEL, null, NETSCAPE, CAPS};

   //Names of files which Cases will generate
   static final String COMPONENT_CFG = "component.cfg";
   static final String CURRENT_VSN = "current.vsn";
   static final String DEPENDENCY_CFG = "dependency.cfg";
   static final String LOOP_CFG = "loop.cfg";
   static final String STEP_CFG = "step.cfg";

   //Titles of frames under Spider menu
         static final String EDIT_TITLE = "SPIDER-Edit";
         static final String DECOMPOSE_TITLE = "SPIDER-
Decompose";
         static final String COMPONENT_CONTENT_TITLE =
"SPIDER-Component Content";
         static final String STEP_CONTENT_TITLE = "SPIDER-Step
Content";
         static final String TRACE_TITLE = "SPIDER-Trace";

   //title of combo boxes
   static final String PROCESS_TITLE = "Select a Process";
         static final String STEP_TYPE_TITLE = "Select a Step Type";
   static final String COMPONENT_TYPE_TITLE = "Select a Component
Type";
   static final String LINKS_TITLE = "Select a Link Type";
   static final String SKILL_TITLE = "ID : Name : Level";
   static final String[] BUTTONS = {"Add", "Edit", "Delete"};

   //Links file name inside Component Content directory
(COMPONENT_CONTENT_DIR)
   static final String COMPONENT_CONTENT_DIR = "Component
Content";
   static final String[] LINK_FILE_NAMES = {"txt.link", "word.link",
"excel.link", "data.link", "url.link", "caps.link"};

   //two types of variant in EHL merging frame
   static final String[] VARIANT_TYPES = {"Old", "New"};


   //data is using for SkillTable
   static final int SKILL_LEVEL = 4;      //from 0 to 3
   static final int SKILL_ID = 21;        //from 1 to 20
   static final int SECURITY_LEVEL = 6;   //from 0 to 5
   static final int PRIORITY_LEVEL = 6;   //from 0 to 5
   static final String[] SKILL_LIST = {"Unix System","CAPS","TAE
Plus", "C", "C++",
                        "Ada", "Notepad", "MS Word", "MS Excel",
"Rational Rose",
                        "UML", "System Analysis", "System Design",
"Coding", "Testing",
                        "Maintenance", "Organization", "Evaluation",
"Management","Negotiation"};//20 skills name
}


package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;
import com.symantec.itools.swing.borders.LineBorder;
```

```
//////////////////////////////////////////////////////////////////////////////
/**
 * ComponentContentFrame : create/edit/delete Component Content
directory with all link
 * files for a selected step.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_ComponentContent
 */
//////////////////////////////////////////////////////////////////////////////
public class ComponentContentFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_ComponentContent
{
  public Vector fnList = new Vector();
  public String pathName = null;
  public String currentComponent = null;

  /**
   * Create ComponentContentFrame
   */
        public ComponentContentFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("SPIDER-Component Content");
                getContentPane().setLayout(null);
                setSize(650,520);
                setVisible(false);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel1.setText("CAPS Files:");
                getContentPane().add(JLabel1);
                JLabel1.setForeground(java.awt.Color.black);
                JLabel1.setBounds(30,370,90,20);

        JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel2.setText("Data Files:");
                getContentPane().add(JLabel2);
                JLabel2.setForeground(java.awt.Color.black);
                JLabel2.setBounds(30,290,90,20);

        JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel3.setText("URLs");
                getContentPane().add(JLabel3);
                JLabel3.setForeground(java.awt.Color.black);
                JLabel3.setBounds(30,330,90,20);

        JLabel4.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel4.setText("Text Files:");
                getContentPane().add(JLabel4);
                JLabel4.setForeground(java.awt.Color.black);
                JLabel4.setBounds(30,170,90,20);
                getContentPane().add(textComboBox);
                textComboBox.setBounds(120,170,500,20);
                getContentPane().add(CAPSComboBox);
                CAPSComboBox.setBounds(120,370,500,20);
                getContentPane().add(URLComboBox);
                URLComboBox.setBounds(120,330,500,20);
                getContentPane().add(saveButton);
                saveButton.setBounds(0,0,0,0);
                getContentPane().add(editButton);
```

```
editButton.setBounds(0,0,0,0);
getContentPane().add(deleteButton);
deleteButton.setBounds(0,0,0,0);
getContentPane().add(addButton);
addButton.setBounds(0,0,0,0);
cancelButton.setText("Cancel");
cancelButton.setActionCommand("Cancel");
getContentPane().add(cancelButton);
cancelButton.setBounds(564,450,73,24);

JLabel5.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel5.setText("Component Content:");
getContentPane().add(JLabel5);
JLabel5.setForeground(java.awt.Color.black);
JLabel5.setFont(new Font("Dialog", Font.BOLD, 16));
JLabel5.setBounds(0,7,170,30);
componentLabel.setText("Component Content:");
getContentPane().add(componentLabel);
componentLabel.setForeground(java.awt.Color.black);
componentLabel.setFont(new Font("Dialog",
Font.BOLD, 15));
componentLabel.setBounds(175,7,500,30);

JLabel6.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel6.setText("Available Components:");
getContentPane().add(JLabel6);
JLabel6.setForeground(java.awt.Color.black);
JLabel6.setBounds(10,60,130,20);
getContentPane().add(componentsComboBox);
componentsComboBox.setBounds(140,60,500,20);
getContentPane().add(connectButton);
connectButton.setBounds(0,0,0,0);
getContentPane().add(MSWordComboBox);
MSWordComboBox.setBounds(120,210,500,20);
getContentPane().add(MSExcelComboBox);
MSExcelComboBox.setBounds(120,250,500,20);

JLabel7.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel7.setText("MS Word Files:");
getContentPane().add(JLabel7);
JLabel7.setForeground(java.awt.Color.black);
JLabel7.setBounds(30,210,90,20);

JLabel8.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel8.setText("MS Excel Files:");
getContentPane().add(JLabel8);
JLabel8.setForeground(java.awt.Color.black);
JLabel8.setBounds(30,250,90,20);
JPanel1.setBorder(lineBorder1);
JPanel1.setLayout(null);
getContentPane().add(JPanel1);
JPanel1.setBounds(10,120,630,300);
JPanel1.add(dataComboBox);
dataComboBox.setBounds(110,168,500,20);

JLabel9.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
JLabel9.setText("Component Content Links");
JLabel9.setBorder(lineBorder1);
JPanel1.add(JLabel9);
JLabel9.setForeground(java.awt.Color.black);
JLabel9.setFont(new Font("Dialog", Font.BOLD, 15));
JLabel9.setBounds(236,0,190,24);
getContentPane().add(JPanel1);
JPanel1.setFont(new Font("Dialog", Font.BOLD, 12));
//$$ lineBorder1.move(0,521);
//}}

//{{INIT_MENUS
//}}
```

```
            //{{REGISTER_LISTENERS
            SymAction lSymAction = new SymAction();
            addButton.addActionListener(lSymAction);
            editButton.addActionListener(lSymAction);
            deleteButton.addActionListener(lSymAction);
            saveButton.addActionListener(lSymAction);
            cancelButton.addActionListener(lSymAction);
            SymItem iSymItem = new SymItem();
            componentsComboBox.addItemListener(iSymItem);
            //}}
        }

    /**
     * Constructor for CasesFrame to launch ComponentContentFrame
     */
    public ComponentContentFrame( String pathName, String stepName,
Vector itemVector, Vector fnList ){
        this();
        //Add Save button
            saveButton.setText("Save");
            saveButton.setActionCommand("Save");
            getContentPane().add(saveButton);
            saveButton.setBounds(492,450,70,24);

            //Add Edit button
            editButton.setText("Edit");
            editButton.setActionCommand("Edit");
            getContentPane().add(editButton);
            editButton.setBounds(81,450,70,24);

            //Add Delete button
            deleteButton.setText("Delete");
            deleteButton.setActionCommand("Delete");
            getContentPane().add(deleteButton);
            deleteButton.setBounds(152,450,70,24);
```



```
            //Add Add button
            addButton.setText("Add");
            addButton.setActionCommand("Add");
            getContentPane().add(addButton);
            addButton.setBounds(10,450,70,24);

        this.fnList = fnList;
        this.componentLabel.setText(stepName);
        this.pathName = pathName;
        this.componentsComboBox.addItem(COMPONENT_TYPE_TITLE);
        for( int i=0; i<itemVector.size(); i++ ){
            this.componentsComboBox.addItem( itemVector.elementAt(i) );
        }
    }

    /**
     * Constructor for TraceFrame to launch ComponentContentFrame
     */
    public ComponentContentFrame( String pathName, String stepName,
String currentItem, Vector fnList ){
        this();
        this.fnList = fnList;
        this.componentLabel.setText(stepName);
        this.pathName = pathName;
        this.componentsComboBox.addItem(currentItem);
        this.componentsComboBox.setSelectedItem(currentItem);
        this.componentsComboBox.setEnabled(false);

        //Connect button is used in trace panel
            connectButton.setText("Connect");
            getContentPane().add(connectButton);
            connectButton.setBounds(517,380,81,24);
    }

        public ComponentContentFrame(String sTitle)
        {
            this();
```

```
		setTitle(sTitle);
	}

	public void setVisible(boolean b)
	{
		if (b)
			setLocation(50, 50);
		super.setVisible(b);
	}

	static public void main(String args[])
	{
		(new ComponentContentFrame()).setVisible(true);
	}

	public void addNotify()
	{
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel4 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JComboBox textComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JComboBox CAPSComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JComboBox URLComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JButton saveButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton editButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton deleteButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton addButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JLabel JLabel5 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel componentLabel = new
com.sun.java.swing.JLabel();
```

```
com.sun.java.swing.JLabel JLabel6 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JComboBox componentsComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JButton connectButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JComboBox MSWordComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JComboBox MSExcelComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JLabel JLabel7 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel8 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JPanel JPanel1 = new
com.sun.java.swing.JPanel();
com.sun.java.swing.JComboBox dataComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
com.symantec.itools.swing.borders.LineBorder lineBorder1 = new
com.symantec.itools.swing.borders.LineBorder();
//}}

//{{DECLARE_MENUS
//}}


class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent
event)
        {
                Object object = event.getSource();
                if (object == addButton)
                        addButton_actionPerformed(event);
                else if (object == editButton)
                        editButton_actionPerformed(event);
                else if (object == deleteButton)
                        deleteButton_actionPerformed(event);
                else if (object == saveButton)
                        saveButton_actionPerformed(event);
                else if (object == cancelButton)
                        cancelButton_actionPerformed(event);
        }
}

/**
 * Launch ConnectionLinksFrame
 * Adding more connection links to a componentContent
 */
public void addButton_actionPerformed(java.awt.event.ActionEvent
event)
{
        String s =
(String)this.componentsComboBox.getSelectedItem();
    (new ConnectionLinksFrame(this, s, BUTTONS[0])).setVisible( true );
}

/**
 * Launch ConnectionLinksFrame
 * Editting existing connection links in a componentContent
 */
        public void
editButton_actionPerformed(java.awt.event.ActionEvent event)
{
        String s =
(String)this.componentsComboBox.getSelectedItem();
    (new ConnectionLinksFrame(this, s, BUTTONS[1])).setVisible( true );
}

/**
 * Launch ConnectionLinksFrame
 * Delete connection links in a componentContent
```

```
    */
        public void
deleteButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          String s =
(String)this.componentsComboBox.getSelectedItem();
     (new ConnectionLinksFrame(this, s, BUTTONS[2])).setVisible( true );
        }

  /**
   * Save all link files under Component Content directory of
   * currentComponent
   */
        public void
saveButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
       saveLinkFile( LINK_FILE_NAMES[i] );
     }
          clearComboBoxes();
          setVisible(false);
          dispose();
        }

  /**
   * Clear all items in every combo box
   * Exit ComponentContentFrame
   */
        public void
cancelButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          clearComboBoxes();
          setVisible(false);
          dispose();
        }

        class SymItem implements java.awt.event.ItemListener
        {
                public void itemStateChanged(java.awt.event.ItemEvent
event)
                {
                        Object object = event.getSource();
                        if (object == componentsComboBox)

        componentsComboBox_itemStateChanged(event);
                }
        }

  /**
   * Select an item in the combo box
   */
  public void
componentsComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
          String selectedItem = null;
     if( event.getStateChange() == event.SELECTED ){
       selectedItem = (String)event.getItem();
       if( !selectedItem.equals(COMPONENT_TYPE_TITLE) ){
         clearComboBoxes();
         setButtonEnabled( true );
         this.currentComponent = (String)checkInput(selectedItem);
         if( !this.currentComponent.equals("") ){
           File f = (File)searchPath(this.currentComponent);
           if( !f.exists() ){
             JOptionPane.showMessageDialog(this, selectedItem+" does
not exist!", "Alert Message",
                                JOptionPane.ERROR_MESSAGE);
           }
           else if( f.exists() ){
             String[] list = f.list();
             for( int j=0; j<list.length; j++ ){
               String s = (String)list[j];
               File aFile = null;
               if( s.equals(COMPONENT_CONTENT_DIR) ){
```

```
                aFile = new File(f, s);
                if( !aFile.isDirectory() ){
                  aFile.mkdir();
                  j= list.length;
                  setButtonEnabled( false );
              }
                else if( aFile.isDirectory() ){
                  j = list.length;
                  setButtonEnabled( true );
                  list = aFile.list();
                  for( int i=0; i<list.length; i++ ){
                    searchFiles(aFile, (String)list[i]);
                  }
                }
              }
              else if( !s.equals(COMPONENT_CONTENT_DIR) ){
                aFile = new File(f, COMPONENT_CONTENT_DIR);
                aFile.mkdir();
                setButtonEnabled( true );
              }
            }
          }
        }
      }
      else if( selectedItem.equals(COMPONENT_TYPE_TITLE) ){
        setButtonEnabled( false );
      }
    }
  }

/**
 * Add/Delete/Edit/Save button only active when a step is selected
 */
      public void setButtonEnabled( boolean flag ){
        addButton.setEnabled( flag );
        editButton.setEnabled( flag );
        deleteButton.setEnabled( flag );
        saveButton.setEnabled( flag );
      }

/**
 * Short cut to print the output
 * @param string : the output string
 */
      void debug( String s ){
        System.out.println(s);
      }

      /**
       * Search and get the info from link files
       *
       * @param aFile : a link file
       * @param fileType : a name of link file
       */
      public void searchFiles(File aFile, String fileType){
    File f = new File( aFile, fileType);
    try{
      BufferedReader br = null;
      String item = null;
      if( fileType.equals(LINK_FILE_NAMES[0]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
          while( (item = br.readLine()) != null ){
            this.textComboBox.addItem(item);
          }
        }
      }
      else if( fileType.equals(LINK_FILE_NAMES[1]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
          while( (item = br.readLine()) != null ){
            this.MSWordComboBox.addItem(item);
          }
        }
```

```
    }
    else if( fileType.equals(LINK_FILE_NAMES[2]) ){
      br = new BufferedReader( new FileReader(f));
      if( br != null ){
        while( (item = br.readLine()) != null ){
          this.MSExcelComboBox.addItem(item);
        }
      }
    }
    else if( fileType.equals(LINK_FILE_NAMES[3]) ){
      br = new BufferedReader( new FileReader(f));
      if( br != null ){
        while( (item = br.readLine()) != null ){
          this.dataComboBox.addItem(item);
        }
      }
    }
    else if( fileType.equals(LINK_FILE_NAMES[4]) ){
      br = new BufferedReader( new FileReader(f));
      if( br != null ){
        while( (item = br.readLine()) != null ){
          this.URLComboBox.addItem(item);
        }
      }
    }
    else if( fileType.equals(LINK_FILE_NAMES[5]) ){
      br = new BufferedReader( new FileReader(f));
      if( br != null ){
        while( (item = br.readLine()) != null ){
          this.CAPSComboBox.addItem(item);
        }
      }
    }
  }
  catch( IOException io ){
    debug("IOException: "+io);
  }
```

```
  }

/**
 * Form an absolute path for a selected component
 *
 * @param s : a selected component name
 * @return File which is formed by a string s
 */
public File searchPath( String s){
  String thePath = null;

  s = "s-"+s;
  //To convert the string of selected item into the file path
  for( int i=0; i<this.fnList.size(); i++ ){
    String fn = (String)this.fnList.elementAt(i);
    String sub = s.substring(0,fn.length());

    if( sub.equals(fn)){
      i = this.fnList.size();
      thePath= sub+"\\";
      String sub2 = s.substring(fn.length());

      char[] ca = (char[])sub2.toCharArray();
      int result = 0;
      for( int j=0; j<ca.length; j++ ){
        String s3 = ca[j]+"";
        result = s3.compareTo("-");
        if( result == 0 ){
          j=ca.length;
          StringTokenizer st = new StringTokenizer(sub2,"-");
            String before_ = st.nextToken("-");
            thePath = thePath+before_;
            String _after = st.nextToken("-");
            st = new StringTokenizer(_after,".");
            while(st.hasMoreTokens()){
              thePath = thePath+"\\"+st.nextToken();
            }
```

```
            }
            else if( (result > 0) && (j==ca.length - 1) ){
               thePath = thePath+sub2;
            }
         }
      }
   }
   File f = new File(this.pathName,thePath);
   return f;
}

/**
 * Set text file names in textComboBox
 *
 * @param v : a vector of links
 */
public void setTextLink(Vector v){
   this.textComboBox.removeAllItems();
   for( int i=0; i<v.size(); i++ ){
      this.textComboBox.addItem(v.elementAt(i));
   }
}

/**
 * Set word file names in MSWordComboBox
 *
 * @param v : a vector of links
 */
public void setWordLink(Vector v){
   this.MSWordComboBox.removeAllItems();
   for( int i=0; i<v.size(); i++ ){
      this.MSWordComboBox.addItem(v.elementAt(i));
   }
}

/**
 * Set excel file names in MSExcelComboBox
 *
 * @param v : a vector of links
 */
public void setExcelLink(Vector v){
   this.MSExcelComboBox.removeAllItems();
   for( int i=0; i<v.size(); i++ ){
      this.MSExcelComboBox.addItem(v.elementAt(i));
   }
}

/**
 * Set personnel object names in dataComboBox
 *
 * @param v : a vector of links
 */
public void setDataLink(Vector v){
   this.dataComboBox.removeAllItems();
   for( int i=0; i<v.size(); i++ ){
      this.dataComboBox.addItem(v.elementAt(i));
   }
}

/**
 * Set URL in URLComboBox
 *
 * @param v : a vector of links
 */
public void setURLLink(Vector v){
   this.URLComboBox.removeAllItems();
   for( int i=0; i<v.size(); i++ ){
      this.URLComboBox.addItem(v.elementAt(i));
   }
}

/**
 * Set caps file names in CAPSComboBox
 *
```

```
     * @param v : a vector of links
     */
    public void setCAPSLink(Vector v){
        this.CAPSComboBox.removeAllItems();
        for( int i=0; i<v.size(); i++ ){
            this.CAPSComboBox.addItem(v.elementAt(i));
        }
    }

    /**
     * Save all link files
     *
     * @param fileType : a type of file, e.g., txt.link, data.link, ...
     */
    public void saveLinkFile( String fileType){
        Vector items = new Vector();
        JComboBox comboBox = (JComboBox)findComboBox( fileType );
        for( int i=0; i<comboBox.getItemCount(); i++ ){
            items.addElement(comboBox.getItemAt(i));
        }

        File f = (File)searchPath(this.currentComponent);
        f = new
File(f.getAbsolutePath()+"\\"+COMPONENT_CONTENT_DIR+"\\"+fileT
ype);
        try{
            FileWriter fw = new FileWriter(f);
            BufferedWriter bw = new BufferedWriter( fw);
            if( bw != null ){
                for( int i=0; i<comboBox.getItemCount(); i++ ){
                    bw.write((String)comboBox.getItemAt(i)+"\n");
                }
            }
            bw.flush();
            bw.close();
            fw.close();
        }
        catch( IOException io ){
            debug("IOException: "+io);
        }
    }

    /**
     * Search a combobox matchs with fileType
     *
     * @param fileType : a type of file, e.g., txt.link, data.link, ...
     * @return JComboBox : a combo box of fileType
     */
    public JComboBox findComboBox( String fileType ){
        JComboBox comboBox = null;
        if( fileType.equals(LINK_FILE_NAMES[0]) ){
            comboBox = this.textComboBox;
        }
        else if( fileType.equals(LINK_FILE_NAMES[1]) ){
            comboBox = this.MSWordComboBox;
        }
        else if( fileType.equals(LINK_FILE_NAMES[2]) ){
            comboBox = this.MSExcelComboBox;
        }
        else if( fileType.equals(LINK_FILE_NAMES[3]) ){
            comboBox = this.dataComboBox;
        }
        else if( fileType.equals(LINK_FILE_NAMES[4]) ){
            comboBox = this.URLComboBox;
        }
        else if( fileType.equals(LINK_FILE_NAMES[5]) ){
            comboBox = this.CAPSComboBox;
        }
        return comboBox;
    }

    /**
     * Refresh all comboboxes
     */
```

```
public void clearComboBoxes(){
   textComboBox.removeAllItems();
   MSWordComboBox.removeAllItems();
   MSExcelComboBox.removeAllItems();
   dataComboBox.removeAllItems();
   URLComboBox.removeAllItems();
   CAPSComboBox.removeAllItems();
}

/**
 * Check the selected component is valid or not
 *
 * @return String : null/valid component name
 * @param selectedItem : a selected component in
componentComboBox
 */
public String checkInput(String selectedItem ){
   if( selectedItem != null ){
      int j = selectedItem.indexOf("-");
      if( j>0){
            String sub2 = (selectedItem.substring(j+1)).trim();
         if( !sub2.equals(this.componentLabel.getText()) ){
            char c = (char)sub2.charAt(0);
            boolean isLetter = (new Character(c)).isLetter(c);
            if( isLetter ){
               JOptionPane.showMessageDialog(this, selectedItem+"
is invalid component!",
                              "Error
Message",JOptionPane.ERROR_MESSAGE);
            return null;
            }
         }
         else{
            return this.componentLabel.getText();
         }
      }
   }
   return selectedItem;
      }
}
```

```
package Cases;

import java.io.Serializable;

////////////////////////////////////////////////////////////////////////////////
/**
 * ComponentType : Create a Component Type object and save it in
component.cfg file
 */
////////////////////////////////////////////////////////////////////////////////
public class ComponentType implements Serializable{

   /**
    * componentID : component type ID
    */
   private String componentID;

   /**
    * componentName : component type name
    */
   private String componentName;

   /**
    * componentDescription : component type description
    */
   private String componentDescription;

   /**
```

```
    * This ComponentType constructor is used to create a ComponentType
object
    */
   public ComponentType( String componentID, String componentName,
String componentDescription ){
      this.componentID = componentID;
      this.componentName = componentName;
      this.componentDescription = componentDescription;
   }

   public ComponentType(){}
   /**
    * @return componentID
    */
   public String getComponentID(){
      return this.componentID;
   }

   /**
    * @return componentName
    */
   public String getComponentName(){
      return this.componentName;
   }

   /**
    * @return componentDescription
    */
   public String getComponentDescription(){
      return this.componentDescription;
   }
}
```



```
package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * ConnectionLinksFrame : Create/Edit/Delete links of a selected
component
 * Launched by ComponentContentFrame's buttons(Add, Edit, or Delete)
 *
 * Implement CasesTitle where stores all global variables of Cases package
 */
//////////////////////////////////////////////////////////////////////////////
public class ConnectionLinksFrame extends com.sun.java.swing.JFrame
                                    implements CasesTitle
{
   /**
    * fd: instantiate an open fileDialog
    */
   FileDialog fd = new FileDialog(this, "Open",FileDialog.LOAD);

   /**
    * update : to check that all the changes is update yet
    */
   boolean update = true;

   /**
    * compContentFrame : get the current ComponentContentFrame
    */
   ComponentContentFrame compContentFrame = null;

   /**
    * listModel : to monitor item list
    */
```

```
DefaultListModel listModel = new DefaultListModel();

/**
 * selectedItem : current link type
 */
private String selectedItem = null;

/**
 * currentPosition : position of selected item in the itemList
 */
private int currentPosition = 0;

/**
 * button : name of button of ComponentContentFrame to launch
ConnectionLinksFrame
 */
String button = null;

/**
 * storedVector : an array of  vector of all links
 */
Vector[] storedVector = {new Vector(), new Vector(), new Vector(), new
Vector(), new Vector(), new Vector()};


/**
 * Build ConnectionLinksFrame
 */
        public ConnectionLinksFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Component Content Editor");
                getContentPane().setLayout(null);
                setSize(500,450);
                setVisible(false);

        titleLabel.setHorizontalAlignment(com.sun.java.swing.SwingCons
tants.CENTER);
                titleLabel.setText("jlabel");
                getContentPane().add(titleLabel);
                titleLabel.setForeground(java.awt.Color.black);
                titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
                titleLabel.setBounds(25,5,450,30);
                exitButton.setText("Exit");
                exitButton.setActionCommand("Cancel");
                getContentPane().add(exitButton);
                exitButton.setBounds(260,400,75,22);
                itemScrollPane.setOpaque(true);
                getContentPane().add(itemScrollPane);
                itemScrollPane.setBounds(15,165,470,200);
                itemScrollPane.getViewport().add(itemList);
                itemList.setBounds(0,0,467,197);
                updateButton.setText("Update");
                updateButton.setActionCommand("OK");
                getContentPane().add(updateButton);
                updateButton.setBounds(164,400,75,22);
                getContentPane().add(connectionComboBox);
                connectionComboBox.setBounds(115,60,270,22);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                JLabel1.setText("Connection Links");
                getContentPane().add(JLabel1);
                JLabel1.setForeground(java.awt.Color.black);
```

```
        JLabel1.setBounds(15,140,470,22);
        getContentPane().add(tempTextField);
        tempTextField.setBounds(12,100,321,22);
        tempButton.setText("jbutton");
        tempButton.setActionCommand("jbutton");
        getContentPane().add(tempButton);
        tempButton.setBounds(333,100,75,22);
        browseButton.setText("Browse");
        browseButton.setActionCommand("Browse");
        getContentPane().add(browseButton);
        browseButton.setBounds(408,100,78,22);
        //}}

        //{{INIT_MENUS
        //}}

        //{{REGISTER_LISTENERS
        SymItem lSymItem = new SymItem();
        connectionComboBox.addItemListener(lSymItem);
        SymAction lSymAction = new SymAction();
        tempButton.addActionListener(lSymAction);
        updateButton.addActionListener(lSymAction);
        exitButton.addActionListener(lSymAction);
        SymMouse aSymMouse = new SymMouse();
        itemList.addMouseListener(aSymMouse);
        browseButton.addActionListener(lSymAction);
        //}}

/**
 * set default model for itemList
 */
itemList.setModel(listModel);

/**
 * create a combobox with all type of links
 */
for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
        connectionComboBox.addItem(LINK_FILE_NAMES[i]);
    }
}

/**
 * Constructor is launched by buttons from ComponentContentFrame
 */
public ConnectionLinksFrame(ComponentContentFrame
compContentFrame, String title, String button){
    this();
    this.compContentFrame = compContentFrame;
    this.button = button;
    this.titleLabel.setText(title);
    this.tempButton.setText(button);
    setListModel();
}

        public ConnectionLinksFrame(String sTitle)
        {
                this();
                setTitle(sTitle);
        }

        public void setVisible(boolean b)
        {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
        }

        static public void main(String args[])
        {
                (new ConnectionLinksFrame()).setVisible(true);
        }

        public void addNotify()
        {
```

```
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JLabel titleLabel = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JButton exitButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JScrollPane itemScrollPane = new
com.sun.java.swing.JScrollPane();
	com.sun.java.swing.JList itemList = new
com.sun.java.swing.JList();
	com.sun.java.swing.JButton updateButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JComboBox connectionComboBox = new
com.sun.java.swing.JComboBox();
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JTextField tempTextField = new
com.sun.java.swing.JTextField();
	com.sun.java.swing.JButton tempButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton browseButton = new
com.sun.java.swing.JButton();
	//}}

	//{{DECLARE_MENUS
	//}}


	class SymItem implements java.awt.event.ItemListener
	{
		public void itemStateChanged(java.awt.event.ItemEvent
event)
		{
			Object object = event.getSource();
			if (object == connectionComboBox)

	connectionComboBox_itemStateChanged(event);
		}
	}

  /**
   * Monitor what link type is selected
   */
	void
connectionComboBox_itemStateChanged(java.awt.event.ItemEvent event)
	{
	  if( event.getStateChange() == event.SELECTED ){
	  listModel.removeAllElements();
	    this.tempTextField.setText("");
```

```
            selectedItem = (String)event.getItem();
            for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
        if( selectedItem.equals(LINK_FILE_NAMES[i]) ){
          for( int j=0; j<storedVector[i].size(); j++ ){
            listModel.addElement(storedVector[i].elementAt(j));
          }
          i= LINK_FILE_NAMES.length;
        }
      }
      }
      }

      class SymAction implements java.awt.event.ActionListener
      {
            public void actionPerformed(java.awt.event.ActionEvent
event)
            {
                  Object object = event.getSource();
                  if (object == tempButton)
                        tempButton_actionPerformed(event);
                  else if (object == updateButton)
                        updateButton_actionPerformed(event);
                  else if (object == exitButton)
                        exitButton_actionPerformed(event);
                  else if (object == browseButton)
                        browseButton_actionPerformed(event);
            }
      }

      void tempButton_actionPerformed(java.awt.event.ActionEvent
event)
      {
    String s = tempTextField.getText();
    if( !s.equals("") ){
      searchButton(s);
            this.tempTextField.setText("");
      update = false;
    }
    else{
      update = true;
    }
      }

  /**
    * Update all link comboboxes in ComponentContentFrame
    */
      void updateButton_actionPerformed(java.awt.event.ActionEvent
event)
      {
        int index = (int)connectionComboBox.getSelectedIndex();
        searchLink(index);
        String s = (String)tempTextField.getText();
        if( !s.equals("") ){
          searchButton(s);
        }
        update = true;
      }

  /**
    * Confirmination dialog to give a user one more chance to update all
    * the change, and exit ConnectionLinksFrame.
    */
      void exitButton_actionPerformed(java.awt.event.ActionEvent
event)
      {
    if( !update ){
        Object[] options = { "OK", "CANCEL" };
      int i = JOptionPane.showOptionDialog(this, "You did not update the
last change yet.  Would you like to update?",
      "Warning", JOptionPane.DEFAULT_OPTION,
JOptionPane.WARNING_MESSAGE, null, options, options[0]);
      if( i==0 ){
        updateButton_actionPerformed(event);
      }
```

```
        }
        setVisible( false );
        dispose();
            }

    /**
     * Connect to FileDialog to let user to select a file insteads of typing, and
return the
     * entire path of the file to the textfield
     */
            void browseButton_actionPerformed(java.awt.event.ActionEvent
event)
            {
        fd.show();
        String fileName = fd.getFile();
        if( fileName != null ){
           tempTextField.setText(fd.getDirectory()+fileName);
        }
            }


            class SymMouse extends java.awt.event.MouseAdapter
            {
                    public void mouseClicked(java.awt.event.MouseEvent
event)
                    {
                            Object object = event.getSource();
                            if (object == itemList)
                                    itemList_mouseClicked(event);
                    }
            }

    /**
     * Monitor which item is selected
     */
            void itemList_mouseClicked(java.awt.event.MouseEvent event)
            {
                    if( event.getClickCount() > 0) {
                       String s = (String)this.itemList.getSelectedValue();
                       this.tempTextField.setText(s);
                       this.currentPosition = (int)this.itemList.getSelectedIndex();
                    }
            }

    /**
     * Search which button (Add/Delete/Edit) from
ComponentContentFrame to
     * launch this frame.
     *
     * @param s : button title which is passed from
ComponentContentFrame
     */
            void searchButton(String s){
                    for( int i=0; i<BUTTONS.length; i++ ){
                       if( this.button.equals(BUTTONS[i]) ){
                          searchVector(s,i);
                          i=BUTTONS.length;
                       }
                    }
            }

    /**
     * Search which comboBox in ComponentContentFrame needs to update
     *
     * @param index : an index of selected item in connectionComboBox
     */
            void searchLink(int index){
      if( index == 0 ){
                    this.compContentFrame.setTextLink(storedVector[0]);
                }
      else if( index == 1 ){
                    this.compContentFrame.setWordLink(storedVector[1]);
                }
      else if( index == 2 ){
```

```
            this.compContentFrame.setExcelLink(storedVector[2]);
        }
    else if( index == 3 ){
            this.compContentFrame.setDataLink(storedVector[3]);
        }
    else if( index == 4 ){
            this.compContentFrame.setURLLink(storedVector[4]);
        }
    else if( index == 5 ){
            this.compContentFrame.setCAPSLink(storedVector[5]);
        }
    }

  /**
   * Set the content of a selected link file into the list
   */
  void setListModel(){
    if( this.compContentFrame != null ){
      for( int j=0; j<LINK_FILE_NAMES.length; j++ ){
        JComboBox cb =
(JComboBox)this.compContentFrame.findComboBox(LINK_FILE_NAME
S[j]);
        int c =cb.getItemCount();
        for( int i=0; i<c; i++ ){
          storedVector[j].addElement(cb.getItemAt(i));
        }
      }
      for( int j=0; j<storedVector[0].size(); j++ ){
        listModel.addElement(storedVector[0].elementAt(j));
      }
    }
  }

  /**
   * Search which vector is respective with the selected link type
   *
   * @param s : a link type
   * @param index : the index of selected link type and it use to match
with
   * the element in array vector, storedVector.
   */
  void searchVector(String s, int index){
    for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
      if( selectedItem.equals(LINK_FILE_NAMES[i]) ){
        if( index==0 ){
          if( !listModel.contains(s) ){
            listModel.addElement(s);
            storedVector[i].addElement(s);
            }
            else{
              JOptionPane.showMessageDialog(this, s+" is already
in the list!",
                                "Warning Message",
JOptionPane.ERROR_MESSAGE);
            }
        }
        else if( index==1 ){
          listModel.removeElementAt(this.currentPosition);
          listModel.insertElementAt(s,this.currentPosition);
          storedVector[i].removeElementAt(this.currentPosition);
          storedVector[i].insertElementAt(s,this.currentPosition);
        }
        else if( index==2 ){
          if( listModel.contains(s) ){
            listModel.removeElementAt(this.currentPosition);
            storedVector[i].removeElementAt(this.currentPosition);
            }
            else{
              JOptionPane.showMessageDialog(this, s+" is not in
the list!",
                                "Warning Message",
JOptionPane.ERROR_MESSAGE);
            }
        }
```

```
            i= LINK_FILE_NAMES.length;
        }
    }
  }
}
```

```
package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * DecomposeListDialog : a dialog stores all decomposed components of a
selected step.
 * It is launched by decomposeComboBox of TraceFrame
 * It has 2 lists, components and decomposed components
 */
//////////////////////////////////////////////////////////////////////////////
public class DecomposeListDialog extends com.sun.java.swing.JDialog
{
  /**
   * Parent frame of this dialog
   */
  TraceFrame tf = null;

  /**
   * Monitor all components of the selected step
   */
  DefaultListModel componentListModel = new DefaultListModel();

  /**
   * Monitor all decomposed components of the selected component
   */
  DefaultListModel decomposeListModel = new DefaultListModel();

  /**
   * Build DecomposeListDialog
   */
        public DecomposeListDialog(JFrame parent)
        {
                super(parent);

                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setModal(true);
                getContentPane().setLayout(null);
                setSize(585,400);
                setVisible(false);
                componentScrollPane.setOpaque(true);
                getContentPane().add(componentScrollPane);
                componentScrollPane.setBounds(15,90,270,230);
                componentScrollPane.getViewport().add(componentList);
                componentList.setBounds(0,0,267,227);
                OKButton.setText("OK");
                OKButton.setActionCommand("OK");
                getContentPane().add(OKButton);
                OKButton.setBounds(207,340,75,22);
                cancelButton.setText("Cancel");
                cancelButton.setActionCommand("Cancel");
```

```
getContentPane().add(cancelButton);
cancelButton.setBounds(303,340,75,22);

titleLabel.setHorizontalAlignment(com.sun.java.swing.SwingCons
tants.CENTER);
titleLabel.setText("jlabel");
getContentPane().add(titleLabel);
titleLabel.setForeground(java.awt.Color.black);
titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
titleLabel.setBounds(42,5,500,30);
decomposeScrollPane.setOpaque(true);
getContentPane().add(decomposeScrollPane);
decomposeScrollPane.setBounds(300,90,270,230);

decomposeScrollPane.getViewport().add(decomposeList);
decomposeList.setBounds(0,0,267,227);

JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);

JLabel1.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
JLabel1.setText("Component");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setBounds(15,68,270,22);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);

JLabel2.setVerticalAlignment(com.sun.java.swing.SwingConstant
s.BOTTOM);
JLabel2.setText("Decompose");
getContentPane().add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(300,68,270,22);
//}}
```



```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
OKButton.addActionListener(lSymAction);
cancelButton.addActionListener(lSymAction);
SymMouse aSymMouse = new SymMouse();
componentList.addMouseListener(aSymMouse);
//}}

/**
 * setModel for componentList and decomposeList
 */
componentList.setModel(componentListModel);
decomposeList.setModel(decomposeListModel);
}

/**
 * TraceFrame use this constructor to launch DecomposeListDialog
 */
public DecomposeListDialog(TraceFrame tf, String title, Vector
itemVector)
{
this((JFrame)tf);
this.tf = tf;
setTitle(title);
this.titleLabel.setText(title);
for( int i=0; i<itemVector.size(); i++ ){

componentListModel.addElement(itemVector.elementAt(i));
}
}

public void setVisible(boolean b)
{
if (b)
  setLocation(50, 50);
super.setVisible(b);
```

```
	}

	public void addNotify()
	{
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets
		Insets insets = getInsets();
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JScrollPane componentScrollPane = new
com.sun.java.swing.JScrollPane();
	com.sun.java.swing.JList componentList = new
com.sun.java.swing.JList();
	com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JLabel titleLabel = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JScrollPane decomposeScrollPane = new
com.sun.java.swing.JScrollPane();
	com.sun.java.swing.JList decomposeList = new
com.sun.java.swing.JList();
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
	//}}


	class SymAction implements java.awt.event.ActionListener
	{
		public void actionPerformed(java.awt.event.ActionEvent
event)
		{
			Object object = event.getSource();
			if (object == OKButton)
				OKButton_actionPerformed(event);
			else if (object == cancelButton)
				cancelButton_actionPerformed(event);
		}
	}

  /**
   * Return a selected decompose step to TraceFrame
   */
	void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
	{
		String selectedItem =
(String)this.decomposeList.getSelectedValue();
		if( selectedItem != null ){
	if( this.tf != null ){
		String s = this.tf.convertToThePath(selectedItem);
		if( s!= null ){
			this.tf.setSelectedItem(s, selectedItem);
			setVisible(false);
			dispose();
```

```
            }
        }
    }
    else{
        setVisible(false);
        dispose();
    }
        }

/**
 * Exit DecomposeListDialog
 */
        void cancelButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
    setVisible(false);
    dispose();
        }

        class SymMouse extends java.awt.event.MouseAdapter
        {
                public void mouseReleased(java.awt.event.MouseEvent
event)
                {
                        Object object = event.getSource();
                        if (object == componentList)
                                componentList_mouseReleased(event);
                }
        }

/**
 * View all decomposed components of the selected component
 */
        void componentList_mouseReleased(java.awt.event.MouseEvent
event)
        {
    if(event.getClickCount() > 0){
        decomposeListModel.removeAllElements();
        if( this.tf != null ){
            String currentComponent =
this.tf.checkSelection((String)this.componentList.getSelectedValue());
            if( !currentComponent.equals("") ){
                String currentPath =
this.tf.convertToThePath(currentComponent);
                if( (currentPath != null) && (!currentPath.equals("")) ){
                    File f = new File(currentPath);
                    if( f.exists() ){
                        Vector decomposedSteps = new Vector();
                        decomposedSteps =
(Vector)findSteps(currentComponent,new File(currentPath));
                        if( (decomposedSteps != null) &&
(decomposedSteps.size() > 0) ){
                                for( int i=0; i<decomposedSteps.size(); i++
){

decomposeListModel.addElement(decomposedSteps.elementAt(i));
                                }
                        }
                        else{
                            JOptionPane.showMessageDialog(this, currentPath+"
step is an atomic component.  Please reselect it!",
                                "Warning
Message",JOptionPane.ERROR_MESSAGE);
                            return;
                        }
                    }
                }
            }
        }
    }
        }

        /**
         * Find all decomposed components of the selected component
```

```
        *
        * @param temp : a string of selected component name
        * @param f : a file with a name temp
        * @return decomposes vector contains all decomposed
components of the component temp
        */
  Vector findSteps(String temp, File f){
    Vector decomposes = new Vector();
    String[] list = (String[])f.list();
    for(int i=0; i<list.length; i++ ){
      File f2 = new File(f, list[i]);
      if( f2.isDirectory() ){
        try{
          int num = Integer.parseInt(f2.getName());
          int index = temp.indexOf("-");
          String temp2 = null;
          if( index > 0 ){
            temp2 = temp+"."+f2.getName();
          }
          else{
            temp2 = temp+"-"+f2.getName();
          }
          decomposes.addElement(temp2);
          findSteps(temp2,f2);
        }
        catch(Exception e){ }
      }
    }
    return decomposes;
  }
}
```

```
package Cases;

import java.awt.*;
import java.io.*;
import com.sun.java.swing.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * DeleteDialog : a dialog uses to delete a personnel object and
 * a project
 */
////////////////////////////////////////////////////////////////////////////////
public class DeleteDialog extends com.sun.java.swing.JDialog
{
  /**
   * fileDialog : use to get a personnel objects under stakeholder directory
   */
  FileDialog fileDialog = null;

  /**
   * deleteProject : a flag to define a purpose of using this dialog
   */
  boolean deleteProject = false;

  /**
   * casesFrame : a parent frame launch this dialog from two menu items,
   *          Project --> Delete Project and
   *          Tools --> Personnel --> Delete
   */
  CasesFrame casesFrame = null;

  /**
   * casesDirectory : the entire path of current project, eg.
F:\Cases\projectName\
   */
  public String casesDirectory = null;
```

```
/**
 *
 */
        public DeleteDialog(Frame parent)
        {
                super(parent);

                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Delete  Project");
                setModal(true);
                getContentPane().setLayout(null);
                setSize(300,130);
                setVisible(false);
                OKButton.setText("OK");
                OKButton.setActionCommand("OK");
                getContentPane().add(OKButton);
                OKButton.setBounds(74,70,73,20);
                cancelButton.setText("Cancel");
                cancelButton.setActionCommand("Cancel");
                getContentPane().add(cancelButton);
                cancelButton.setBounds(152,70,73,20);
                //}}

                //{{REGISTER_LISTENERS
                SymAction lSymAction = new SymAction();
                OKButton.addActionListener(lSymAction);
                cancelButton.addActionListener(lSymAction);
                browseButton.addActionListener(lSymAction);
                SymItem lSymItem = new SymItem();
                projectComboBox.addItemListener(lSymItem);
                //}}
        }

        public DeleteDialog()
        {
                this((Frame)null);
        }

    /**
     * CasesFrame uses this constructor to launch this dialog
     *
     * @param casesFrame : current CasesFrame
     * @param title : "Delete Project" or "Delete Personnel Data"
     */
        public DeleteDialog(CasesFrame casesFrame, String title){
          this(title);
          this.casesFrame = casesFrame;
          if( title.equals("Delete Project") ){
                getContentPane().add(projectComboBox);
                projectComboBox.setBounds(161,30,128,20);
                projectTextField.setEditable(false);
                getContentPane().add(projectTextField);
                projectTextField.setBackground(java.awt.Color.white);
                projectTextField.setBounds(10,30,150,20);

            casesDirectory =
this.casesFrame.CASESDIRECTORY.getAbsolutePath()+"\\";
            projectTextField.setText(casesDirectory);

            String[] list = this.casesFrame.CASESDIRECTORY.list();

            if( list.length > 0 ){
              for(int i=0; i<list.length; i++ ){
                File aFile = new File(casesDirectory, list[i]);
                if( aFile.isDirectory() ){
```

```
                projectComboBox.addItem(list[i]);
            }
        }
    }
    deleteProject = true;
  }
  else if( title.equals("Delete Personnel Data") ){
fileDialog = new FileDialog(this.casesFrame, title);
            getContentPane().add(filenameTextField);
            filenameTextField.setBounds(10,30,200,20);
            browseButton.setText("Browse");
            browseButton.setActionCommand("Browse");
            getContentPane().add(browseButton);
            browseButton.setBounds(211,30,78,20);

    deleteProject = false;
  }
}

public DeleteDialog(String sTitle)
{
        this();
        setTitle(sTitle);
}

public void setVisible(boolean b)
{
        if (b)
                setLocation(50, 50);
        super.setVisible(b);
}

static public void main(String args[])
{
        (new DeleteDialog()).setVisible(true);
}
```

```
        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets
                Insets insets = getInsets();
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JTextField filenameTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton browseButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JComboBox projectComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JTextField projectTextField = new
com.sun.java.swing.JTextField();
        //}}
```

```
class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent
event)
        {
                Object object = event.getSource();
                if (object == OKButton)
                        OKButton_actionPerformed(event);
                else if (object == cancelButton)
                        cancelButton_actionPerformed(event);
                else if (object == browseButton)
                        browseButton_actionPerformed(event);
        }
}

/**
 * To check deleteProject or deletePersonnel before delete it
 */
        void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
  if( deleteProject ){
          File aFile = new File(projectTextField.getText());
    removeAll(aFile);
  }
  else{
    File aFile = new File(filenameTextField.getText());
    aFile.delete();
  }
  setVisible(false);
  dispose();
        }

/**
 * Exit DeleteDialog
 */
```



```
        void cancelButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
  setVisible(false);
  dispose();
        }

/**
 * Show fileDialog with current directory is stakeholder and list all
personnel file
 */
        void browseButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
  fileDialog.setMode(FileDialog.LOAD);

fileDialog.setDirectory(this.casesFrame.STAKEHOLDER.getAbsolutePath
());
  fileDialog.show();
  if( fileDialog.getFile() != null ){

filenameTextField.setText(fileDialog.getDirectory()+fileDialog.getFile());
  }
  else{
    filenameTextField.setText("");
  }
}

        class SymItem implements java.awt.event.ItemListener
        {
                public void itemStateChanged(java.awt.event.ItemEvent
event)
                {
                        Object object = event.getSource();
                        if (object == projectComboBox)

        projectComboBox_itemStateChanged(event);
```

```
                }
            }

    /**
     * projectComboBox contains all project names under Cases directory
     * a user allows to select a project which they want to delete
     */
            void
projectComboBox_itemStateChanged(java.awt.event.ItemEvent event)
            {
        if( event.getStateChange() == event.SELECTED ){
            String s = (String)event.getItem();

            projectTextField.setText(casesDirectory+s);
        }
            }

    /**
     * Remove all files under the deleting project before delete the project
folder
     *
     * @param aFile : the deleting project folder
     */
    void removeAll(File aFile){
                String[] l = aFile.list();
                for( int i=0; i<l.length; i++ ){
                    File file = new File(aFile, l[i]);
            if( file.isDirectory() ){
                removeAll(file);
                    }
                    else{
                        file.delete();
                    }
                }
                aFile.delete();
    }
}
```

```
package Cases;

/**
 * Dependency object which is used to save in the dependency.cfg file
 */

import java.io.Serializable;

//////////////////////////////////////////////////////////////////////////////
/**
 * Dependency : Create a Dependency object and save it in dependency.cfg
file
 */
//////////////////////////////////////////////////////////////////////////////
public class Dependency implements Serializable{

    /**
     * loopName : evolution process name
     */
    private String loopName = null;

    /**
     * step : step type name
     */
    private String step = null;

    /**
     * outputComponent : output component of the step type
     */
    private String outputComponent = null;

    /**
```

```
  * primaryInput : primary input of the step type
  */
 private String primaryInput = null;

 /**
  * secondaryInput : secondary input of the step type
  */
 private String secondaryInput = null;;

 /**
  * This Dependency constructor is used to create a Dependency object
  */
 public Dependency( String loopName, String step, String
outputComponent, String primaryInput, String secondaryInput ){
    this.loopName = loopName;
    this.step = step;
    this.outputComponent = outputComponent;
    this.primaryInput = primaryInput;
    this.secondaryInput = secondaryInput;
 }

 public Dependency() { }

 /**
  * @return loopName : evolution process name
  */
 public String getLoopName(){
    return this.loopName;
 }

 /**
  * @return step : step type name
  */
 public String getStep(){
    return this.step;
 }

 /**
  * @return outputComponent : output component of the step type
  */
 public String getOutputComponent(){
    return this.outputComponent;
 }

 /**
  * @return primaryInput : primary input of the step type
  */
 public String getPrimaryInput(){
    return this.primaryInput;
 }

 /**
  * @return secondaryInput : secondary input of the step type
  */
 public String getSecondaryInput(){
    return this.secondaryInput;
 }

 /**
  * Set secondary input for this Dependency object
  *
  * @param secondaryInput : secondary input component
  */
 public void setSecondaryInput( String secondaryInput ){
    this.secondaryInput = secondaryInput;
 }
}
```

```
package Cases;

import java.awt.*;
import java.io.*;
import java.util.*;
import com.sun.java.swing.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * EditDecomposeFrame : use to edit/decompose a selected step
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_EditDecompose
 */
//////////////////////////////////////////////////////////////////////////////
public class EditDecomposeFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_EditDecompose
{
  /**
   * pathName : an absolute path of a selected step
   */
  String pathName = null;

  /**
   * newPath : an absolute path after decompose a selected step
   */
  String newPath = null;

  /**
   * isEdit : a flag to keep track the purpose of using this frame
   */
  boolean isEdit = false;

  /**
   * Build EditDecomposeFrame
   */
        public EditDecomposeFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Edit");
                getContentPane().setLayout(null);
                setSize(470,305);
                setVisible(false);

        titleLabel.setVerticalTextPosition(com.sun.java.swing.SwingConst
ants.BOTTOM);

        titleLabel.setHorizontalAlignment(com.sun.java.swing.SwingCons
tants.RIGHT);

        titleLabel.setVerticalAlignment(com.sun.java.swing.SwingConsta
nts.BOTTOM);
                titleLabel.setText("TESTING: ");
                getContentPane().add(titleLabel);
                titleLabel.setForeground(java.awt.Color.black);
                titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
                titleLabel.setBounds(3,6,120,30);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel1.setText("Step Version");
                getContentPane().add(JLabel1);
                JLabel1.setForeground(java.awt.Color.black);
                JLabel1.setBounds(7,54,175,22);
                stepTextField.setEditable(false);
```

```
getContentPane().add(stepTextField);
stepTextField.setBackground(java.awt.Color.white);
stepTextField.setBounds(187,54,270,22);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel2.setText("Output Component");
getContentPane().add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(7,96,175,22);
outputTextField.setEditable(false);
getContentPane().add(outputTextField);
outputTextField.setBackground(java.awt.Color.white);
outputTextField.setBounds(187,96,270,22);

JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel3.setText("Primary Input Component(s)");
getContentPane().add(JLabel3);
JLabel3.setForeground(java.awt.Color.black);
JLabel3.setBounds(7,138,175,22);
getContentPane().add(primaryTextField);
primaryTextField.setBounds(187,138,270,22);

JLabel4.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel4.setText("Secondary Input Component(s)");
getContentPane().add(JLabel4);
JLabel4.setForeground(java.awt.Color.black);
JLabel4.setBounds(7,180,175,22);
getContentPane().add(secondaryTextField);
secondaryTextField.setBounds(187,180,270,22);

selectedLabel.setVerticalTextPosition(com.sun.java.swing.SwingC
onstants.BOTTOM);
selectedLabel.setVerticalAlignment(com.sun.java.swing.SwingCo
nstants.BOTTOM);
selectedLabel.setText("Selected Label");
getContentPane().add(selectedLabel);
selectedLabel.setBackground(new
java.awt.Color(204,204,204));
selectedLabel.setForeground(java.awt.Color.black);
selectedLabel.setFont(new Font("Dialog", Font.BOLD,
16));
selectedLabel.setBounds(130,6,290,30);
saveButton.setText("Save");
saveButton.setActionCommand("Save");
getContentPane().add(saveButton);
saveButton.setBounds(12,258,75,22);
exitButton.setText("Cancel");
exitButton.setActionCommand("Save");
getContentPane().add(exitButton);
exitButton.setBounds(372,258,75,22);
getContentPane().add(deleteButton);
deleteButton.setBounds(0,0,0,0);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
saveButton.addActionListener(lSymAction);
deleteButton.addActionListener(lSymAction);
exitButton.addActionListener(lSymAction);
//}}
}
/**
 * CasesFrame launch this frame through SPIDER --> Edit
 */
public EditDecomposeFrame(String sTitle, String stepName,
```

```
                    String output, String pathName)
        {
                this();
    //Set Delete button for Edit frame
                deleteButton.setText("Delete");
                deleteButton.setActionCommand("Delete");
                getContentPane().add(deleteButton);
                deleteButton.setBounds(84,258,75,22);

                setTitle(sTitle);
                this.titleLabel.setText( "Edit : " );
                this.selectedLabel.setText( pathName );
                this.stepTextField.setText( stepName );
                this.outputTextField.setText( output );
                this.pathName = pathName;

                try{
                  File f = new File(pathName+"\\input.p");
                  if( f.exists() ){
                    DataInputStream primIn = new DataInputStream(
new FileInputStream(f));
                    if( primIn != null ){
                      this.primaryTextField.setText(
(String)primIn.readLine() );
                    }
        }
        f = new File(pathName+"\\input.s");
        if( f.exists() ){
                    DataInputStream secondIn = new DataInputStream(
new FileInputStream(f));
                    if( secondIn != null ){
                      this.secondaryTextField.setText(
(String)secondIn.readLine() );
                    }
        }
                }
                catch( IOException io ){ System.out.println(io);}
```



```
                newPath = this.pathName;
                isEdit = true;
        }

  /**
   * CasesFrame launch this frame through SPIDER --> Decompose
   */
        public EditDecomposeFrame(String sTitle, String stepName,
                    String output, String pathName, int subDir)
        {
                this();

                setTitle(sTitle);
                this.titleLabel.setText( "Decompose:" );
                this.selectedLabel.setText( pathName);
                this.stepTextField.setText( stepName );
                this.outputTextField.setText( output );
                this.pathName = pathName;

                try{
                  File f = new File(pathName+"\\input.p");
                  if( f.exists() ){
                    DataInputStream primIn = new DataInputStream(
new FileInputStream(f));
                    if( primIn != null ){
                      this.primaryTextField.setText(
(String)primIn.readLine() );
                    }
        }
        f = new File(pathName+"\\input.s");
        if( f.exists() ){
                    DataInputStream secondIn = new DataInputStream(
new FileInputStream(f));
                    if( secondIn != null ){
                      this.secondaryTextField.setText(
(String)secondIn.readLine() );
```

```
                }
        }
                }
                catch( IOException io ){ System.out.println(io);}

                newPath = this.pathName + "\\" + subDir;
        }

        public void setVisible(boolean b)
        {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
        }

        static public void main(String args[])
        {
                (new EditDecomposeFrame()).setVisible(true);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets and menu
bar
                Insets insets = getInsets();
                com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
                int menuBarHeight = 0;
                if (menuBar != null)
                        menuBarHeight =
menuBar.getPreferredSize().height;
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JLabel titleLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField stepTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField outputTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField primaryTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel4 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField secondaryTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel selectedLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JButton saveButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton exitButton = new
com.sun.java.swing.JButton();
```

```
        com.sun.java.swing.JButton deleteButton = new
com.sun.java.swing.JButton();
        //}}

        //{{DECLARE_MENUS
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == saveButton)
                                saveButton_actionPerformed(event);
                        else if (object == deleteButton)
                                deleteButton_actionPerformed(event);
                        else if (object == exitButton)
                                exitButton_actionPerformed(event);
                }
        }

  /**
   * Save all the creations or the changes after decomposing and editting
   */
        public void
saveButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          File dir = new File(newPath);
          if( !dir.exists() ){
            dir.mkdir();
          }

          if( dir.isDirectory() ){
      try{
        if( !isEdit ){
                File oldFile = new File(this.pathName,
COMPONENT_CONTENT_DIR);
                File newFile = new
File(newPath,COMPONENT_CONTENT_DIR);
            newFile.mkdir();
            if( oldFile.exists() && newFile.isDirectory() ){
              copyFile(oldFile, newFile);
            }
          }
              if( checkInputs(secondaryTextField.getText())) {
            FileOutputStream fileOutput = new FileOutputStream(new
File(dir, "\\input.p"));
            DataOutputStream depPrimary = new DataOutputStream(
fileOutput);
            if( depPrimary != null ){
              if( !primaryTextField.getText().equals("") ){
                depPrimary.writeBytes(primaryTextField.getText());
              }
            }
            depPrimary.flush();
            depPrimary.close();
            fileOutput.close();

            fileOutput = new FileOutputStream(new File(dir,"\\input.s" ));
            DataOutputStream depSecondary = new DataOutputStream(
fileOutput );
            if( depSecondary != null ){
              if( !secondaryTextField.getText().equals("") ){
                depSecondary.writeBytes(secondaryTextField.getText());
              }
            }
            depSecondary.flush();
            depSecondary.close();
            fileOutput.close();
            exitButton_actionPerformed(null);
          }
        }
```

```
        catch(FileNotFoundException fex ){ System.out.println(fex);}
        catch(IOException e){System.out.println(e);}
    }
        }

  /**
   * Delete this selected step
   */
        void deleteButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
            int result = JOptionPane.showConfirmDialog( this, "All the files
and subdirectories will be deleted! Would you like to continue?",
                        "Confirm
Message",JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
    //result = 0 ==> yes
    //result = 1 ==> no
    if( result == 0 ){
        File aFile = new File(this.pathName);
        if( aFile.exists() ){
            removeAll(aFile);
        }
    }
    exitButton_actionPerformed(null);
        }

  /**
   * Exit EditDecomposeFrame
   */
        public void
exitButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            this.setEnabled(false);
    dispose();
        }
```

```
  /**
   * Copy all link files under Component Content directory into the new
decomposed step
   *
   * @param oldFile : old link files of a parent file
   * @param newFile : new link files of new atomic
   */
        public void copyFile( File oldFile, File newFile ){
            try{
                for( int i=0; i< LINK_FILE_NAMES.length; i++ ){
            File oldLink = new File(oldFile, LINK_FILE_NAMES[i]);
            File newLink = new File(newFile, LINK_FILE_NAMES[i]);
            FileWriter fileWriter = new FileWriter (newLink);
            BufferedWriter bw = new BufferedWriter( fileWriter );
            if( bw != null ){
                String line = null;
                FileReader fileReader = new FileReader(oldLink);
                BufferedReader br = new BufferedReader( fileReader);
                if( br != null ){
                    while( (line=br.readLine()) != null ){
                        bw.write(line);
                    }
                }
                br.close();
                fileReader.close();
            }
            bw.flush();
            bw.close();
            fileWriter.close();
        }
    }
    catch( FileNotFoundException f){System.out.println(f);}
    catch(IOException io ){System.out.println(io);}
        }

  /**
   * Check all components in secondary textfield are valid or not
```

```
    * @param secondary : a string of secondaryTextField
    * @return boolean : valid or invalide input
    */
        public boolean checkInputs(String secondary ){
          String output = outputTextField.getText();
          String sub2 = null;

          if( secondary != null ){
            StringTokenizer st = new StringTokenizer(secondary, ",");
            while( st.hasMoreTokens() ){
              String s = (String)st.nextToken();
        int j = s.indexOf("-");
        if( j>0){
                sub2 = (s.substring(j+1)).trim();
          if( !sub2.equals(output) ){
                char c = (char)sub2.charAt(0);
                boolean isLetter = (new Character(c)).isLetter(c);
                if( isLetter ){
                  JOptionPane.showMessageDialog(this, s+" is
invalid component!",
                              "Error
Message",JOptionPane.ERROR_MESSAGE);
              return false;
              }
            }
          }
        }
      }
          return true;
        }

  /**
    * Remove all files under the deleting the selected step before delete the
step folder
    * @param aFile : the deleting the selected step folder
    */
  void removeAll(File aFile){
              String[] l = aFile.list();
              for( int i=0; i<l.length; i++ ){
                File file = new File(aFile, l[i]);
          if( file.isDirectory() ){
            removeAll(file);
              }
              else{
                file.delete();
              }
              }
              aFile.delete();
    }
  }
```

```
package Cases;

import java.io.Serializable;

//////////////////////////////////////////////////////////////////////////////
/**
 * EHL : Create a EHL object and save it in loop.cfg file
 */
//////////////////////////////////////////////////////////////////////////////
public class EHL implements Serializable{

  /**
   * EHLName : evolution history process name
   */
```

```
    private String EHLName;

    /**
     * EHLPath : a string of step types in the evolution process
     */
    private String EHLPath;

    /**
     * This EHL constructor is used to create a EHL object
     */
    public EHL( String EHLName, String EHLPath ){
      if( EHLName == null ){
        EHLName = "";
      }

      if( EHLPath == null ){
        EHLPath = "";
      }
      this.EHLName = EHLName;
      this.EHLPath = EHLPath;
    }

    public EHL() { }
    /**
     * @return EHLName : evolution history process name
     */
    public String getEHLName(){
      return this.EHLName;
    }

    /**
     * @return EHLPath : a string of step types in the evolution process
     */
    public String getEHLPath(){
      return this.EHLPath;
    }
  }
```

```
package Cases;

import java.util.*;
import java.io.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * I_AVC : an interface for 3 classes - AVCCreateStepFrame
 *                                    - AVCMergingFrame
 *                                    - AVCSplittingFrame
 */
////////////////////////////////////////////////////////////////////////////////
public interface I_AVC {
  public void checkPath(Vector v);
  public void createFiles( Dependency dep, File theFile);
        public void setLoopNameComboBox( Vector loopVector );
        public Vector tokenizeVector(String string);
}
package Cases;

import java.util.*;
import java.io.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * I_AVCOpenStep : an interface for AVCOpenStepFrame
 */
////////////////////////////////////////////////////////////////////////////////
public interface I_AVCOpenStep{
  public void setInitialFrame( VersionControl vc );
  public void setLoopNameComboBox( Vector loopVector );
  public void setStepComboBox( Vector stepVector );
```

```
   public void setVersionComboBox( Vector versionVector );

}
```

```
package Cases;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * I_Cases : an interface for CasesFrame
 */
//////////////////////////////////////////////////////////////////////////////
public interface I_Cases{
        public void getVersionControl();
        public void directoryTree();
   public void currentDir(String absPath );
   public void subDir(String absPath );
   public Vector getComponents(String stepName, String absPath);
        public void setComponentContent( String stepName, String
absPath );
   public void setStepContent(TraceFrame traceFrame, String stepName,
String absPath );
   public void setTraceFrame(String stepName, String absPath);
   public void getAtomics(File aFile, Vector storedVector);
        public int findMax( String thePath );
        public void tokenizer( Vector v, String s);
        public Vector fileNameList();
        public void savePersonnelVector(Vector v);
        public Vector getPersonnelVector();
        public Vector getAllAtomics();
        public void saveStepContentVector(Vector v);
        public Vector getStepContentVector();
        public Vector getScheduledAtomicVector();
}
```

```
package Cases;

import java.util.*;
import java.io.*;
import com.sun.java.swing.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * I_ComponentContent : an interface for ComponentContentFrame
 */
//////////////////////////////////////////////////////////////////////////////
public interface I_ComponentContent{
   public String checkInput(String selectedItem );
   public JComboBox findComboBox( String fileType );
   public void saveLinkFile( String fileType);
        public void searchFiles(File aFile, String fileType);
   public File searchPath( String s);
   public void setTextLink(Vector v);
   public void setWordLink(Vector v);
   public void setExcelLink(Vector v);
   public void setDataLink(Vector v);
   public void setURLLink(Vector v);
   public void setCAPSLink(Vector v);
}
```

```
package Cases;

import java.io.*;

////////////////////////////////////////////////////////////////////////////
/**
 * I_EditDecompose : an interface for EditDecomposeFrame
 */
////////////////////////////////////////////////////////////////////////////
public interface I_EditDecompose{
        public boolean checkInputs(String secondary );
        public void copyFile( File oldFile, File newFile );
}
```

```
package Cases;

import java.util.*;

////////////////////////////////////////////////////////////////////////////
/**
 * I_Personnel : an interface for PersonnelFrame
 */
////////////////////////////////////////////////////////////////////////////
public interface I_Personnel{
        public void setInitial(Personnel personnel);
        public Personnel getPersonnelData();
  public void setSkillComboBox(Vector v);
}
```

```
package Cases;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;

////////////////////////////////////////////////////////////////////////////
/**
 * I_ProjectSchema : an interface for ProjectSchemaFrame
 */
////////////////////////////////////////////////////////////////////////////
public interface I_ProjectSchema{
  public void readDepFile();
  public void readInputFiles( String pathName );
  public void setCompComboBox( Vector compVector );
  public void setCompInfo( ComponentType ct );
  public void setDepenEHLComboBox( Vector EHLVector );
  public void setDependStepComboBox( Vector stepVector );
  public void setDepInfo( String selectedDepStep);
  public void setEHLComboBox( Vector EHLVector );
  public void setEHLInfo( EHL ehl );
  public void setListModel( Vector stepVector );
  public void setStepComboBox( Vector stepVector );
        public void setItemList(Object[] objs);
}
```

```
package Cases;

import java.util.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * I_StepContent : an interface for StepContentFrame
 */
////////////////////////////////////////////////////////////////////////////////
public interface I_StepContent{
        public StepContent getStepContent();
        public void setInitial( StepContent stepContent );
  public void setPredecessorComboBox(Object[] oa);
        public void setReadOnly();
        public void setStakeHolders();
  public void setSkillComboBox(Vector v);
}
```

```
package Cases;

import java.util.*;
import java.io.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * I_Trace : an interface for TraceFrame
 */
////////////////////////////////////////////////////////////////////////////////
public interface I_Trace{
  public String checkSelection(String selectedItem );
  public String convertToThePath(String s);
  public File searchFilePath( String s);
        public void searchFiles(File aFile, String fileType);
  public void searchIndex();
  public void searchInputFiles( String thePath );
  public void searchPath( String s);
        public void setComponentContent(String selectedItem, File f);
  public void setHistory( String s);
  public void setInitial(Vector componentsVector);
  public void setSelectedItem(String currentPath, String selectedItem);
        public void tokenizer( Vector v, String s);
}
```

```
package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * ListDialog : Use to list components.  It is launched by
StepContentFrame,
 *          TraceFrame, and ProjectSchemaFrame.
 *
 * Implement CasesTitle where stores all global variables of Cases package
```

```
 */
//////////////////////////////////////////////////////////////////////////////
public class ListDialog extends com.sun.java.swing.JDialog implements
CasesTitle
{
  /**
   * tf : trace frame, one of owner frames
   */
  TraceFrame tf = null;

  /**
   * scf : step component frame, one of owner frames
   */
  StepContentFrame scf = null;

  /**
   * psf : project schema frame, one of owner frames
   */
  ProjectSchemaFrame psf = null;

  /**
   * listModel : model of the itemList
   */
  DefaultListModel listModel = new DefaultListModel();

  /**
   * index : index of component content or trace button from TraceFrame
   */
  int index = 0;

  /**
   * Buil ListDialog
   */
        public ListDialog(JFrame parent)
        {
                super(parent);
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setModal(true);
                getContentPane().setLayout(null);
                setSize(300,400);
                setVisible(false);
                getContentPane().add(itemScrollPane);
                itemScrollPane.setBounds(15,70,270,250);
                itemScrollPane.getViewport().add(itemList);
                itemList.setBounds(0,0,267,247);
                OKButton.setText("OK");
                getContentPane().add(OKButton);
                OKButton.setBounds(64,340,75,24);
                cancelButton.setText("Cancel");
                getContentPane().add(cancelButton);
                cancelButton.setBounds(160,340,75,24);

        titleLabel.setHorizontalAlignment(com.sun.java.swing.SwingCons
tants.CENTER);
                titleLabel.setText("jlabel");
                getContentPane().add(titleLabel);
                titleLabel.setForeground(java.awt.Color.black);
                titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
                titleLabel.setBounds(1,10,298,30);
                //}}

                //{{REGISTER_LISTENERS
                SymAction lSymAction = new SymAction();
                OKButton.addActionListener(lSymAction);
```

```
                cancelButton.addActionListener(lSymAction);
                //}}

                /**
                 * Set model for the itemList
                 */
                itemList.setModel(listModel);
        }

    /**
     * StepContentFrame launchs this dialog with its components list
     *
     * @param scf : step content frame, owner frame
     * @param title : title of this dialog when step content frame launchs it
     * @param itemVector : list of components from step content frame
     */
        public ListDialog(StepContentFrame scf, String title, Vector
itemVector)
        {
                this((JFrame)scf);
                this.scf = scf;
                setTitle(title);
                this.titleLabel.setText(title);
                for( int i=0; i<itemVector.size(); i++ ){
                   listModel.addElement(itemVector.elementAt(i));
                }
        }

    /**
     * TraceFrame launchs this dialog with its components list
     *
     * @param tf : trace frame, owner frame
     * @param title : title of this dialog when trace frame launchs it
     * @param itemVector : list of components from trace frame
     * @param index : index of button from trace frame, eg, 0:trace button,
or
     *                                1:component content buton
     */
        public ListDialog(TraceFrame tf, String title, Vector itemVector,
int index)
        {
                this((JFrame)tf);
                this.tf = tf;
                this.index = index;
                setTitle(title);
                this.titleLabel.setText(title);
                for( int i=0; i<itemVector.size(); i++ ){
                   listModel.addElement(itemVector.elementAt(i));
                }
        }

    /**
     * ProjectSchemaFrame launchs this dialog with its components list
     *
     * @param psf : project schema frame, owner frame
     * @param title : title of this dialog when project schema frame launchs
it
     * @param itemVector : list of components from project schema frame
     */
        public ListDialog(ProjectSchemaFrame psf, String title, Vector
itemVector)
        {
                this((JFrame)psf);
                this.psf = psf;
                setTitle(title);
                this.titleLabel.setText(title);
                for( int i=0; i<itemVector.size(); i++ ){

listModel.addElement(((ComponentType)itemVector.elementAt(i)).getCom
ponentID());
                }
        }

        public void setVisible(boolean b)
```

```
        {
                if (b){
                   this.setLocationRelativeTo(this.scf);
                }
                super.setVisible(b);
        }


        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets
                Insets insets = getInsets();
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JScrollPane itemScrollPane = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JList itemList = new
com.sun.java.swing.JList();
        com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel titleLabel = new
com.sun.java.swing.JLabel();
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == OKButton)
                                OKButton_actionPerformed(event);
                        else if (object == cancelButton)
                                cancelButton_actionPerformed(event);
                }
        }

  /**
   * Return selected item(s) to the owner frame
   */
        void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
     if( this.scf != null ){

this.scf.setPredecessorComboBox(this.itemList.getSelectedValues());
        setVisible(false);
        dispose();
     }
     else if( this.tf != null ){
        String selectedItem = (String)this.itemList.getSelectedValue();
        if( index == 0 ){
           String s = this.tf.convertToThePath(selectedItem);
           if( s!= null ){
```

```
            this.tf.setSelectedItem(s, selectedItem);
            setVisible(false);
            dispose();
          }
        }
        else if( index == 1 ){
          String currentComponent =
(String)this.tf.checkSelection(selectedItem);
          if( !currentComponent.equals("") ){
            File f = (File)this.tf.searchFilePath(currentComponent);
            if( !f.exists() ){
              JOptionPane.showMessageDialog(this, selectedItem+" does
not exist!", "Alert Message",
                                JOptionPane.ERROR_MESSAGE);
            }
            else{
              String s = this.tf.convertToThePath(currentComponent);
              if( s!= null ){
                this.tf.setSelectedItem(s, selectedItem);
                setVisible(false);
                dispose();
                this.tf.setComponentContent(selectedItem, f);
              }
              setVisible(false);
              dispose();
            }
          }

        }

      }
      else if( this.psf != null ){
        this.psf.setItemList(this.itemList.getSelectedValues());
        setVisible(false);
        dispose();
      }
        }
```

```
  /**
   * Exit ListDialog
   */
        void cancelButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
      setVisible(false);
      dispose();
        }
}
```

```
package Cases;

/**
 * Personnel Data object which is used to save
 * under stakeholder directory
 */

import java.io.Serializable;
import java.util.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * Personnel : Create a Personnel object and save it in a file with
 *          the name is Personnel's ID under stakeholder directory
```

```
 */
////////////////////////////////////////////////////////////////////////////////
public class Personnel implements Serializable{

  /**
   * ID : personnel ID
   */
  private String ID = null;

  /**
   * name : name of a person
   */
  private String name = null;

  /**
   * skill : vector of the person's skills
   */
  private Vector skill = new Vector();

  /**
   * securityLevel : security level of the person
   */
  private int  securityLevel = 0;

  /**
   * email : e-mail address of the person
   */
  private String  email = null;

  /**
   * telephone : telephone number of the person
   */
  private String  telephone = null;

  /**
   * fax : fax number of the person
   */
  private String  fax = null;

  /**
   * address : address of the person
   */
  private String address = null;

  /**
   * majorJobs : major job list for this person
   */
  private Vector majorJobs = new Vector();

  /**
   * minorJobs : minor job list for this person
   */
  private Vector minorJobs = new Vector();

  /**
   * This Personnel constructor is used to create a Personnel object
   */
  public Personnel(String ID, String name, Vector skill,
                   int securityLevel, String email, String telephone,
                   String fax, String address){
    this();
    this.ID = ID;
    this.name = name;
    this.skill = skill;
    this.securityLevel = securityLevel;
    this.email = email;
    this.telephone = telephone;
    this.fax = fax;
    this.address = address;
  }

  /**
   * Empty Personnel constructor
   */
```

```
public Personnel(){
}

/**
 * Set ID for this personnel object
 *
 * @param s : string of ID
 */
public void setID(String s){
   this.ID = s;
}

/**
 * ID of this personnel object
 *
 * @return ID : this person's ID
 */
public String getID(){
   return this.ID;
}

/**
 * Set name for this personnel object
 *
 * @param s : this person's name
 */
public void setName(String s){
   this.name = s;
}

/**
 * The person's name
 *
 * @return name : the name of person
 */
public String getName(){
   return this.name;
}

/**
 * Set skills for the person
 *
 * @param v : list of person's skill
 */
public void setSkill(Vector v){
   this.skill = v;
}

/**
 * Skills of the person
 *
 * @return skill : a vector of skills
 */
public Vector getSkill(){
   return this.skill;
}

/**
 * Set security level for the person
 *
 * @param i : security level
 */
public void setSecurityLevel(int i){
   this.securityLevel = i;
}

/**
 * Security level of the person
 *
 * @return securityLevel : security level
 */
public int getSecurityLevel(){
   return this.securityLevel;
}
```

```
/**
 * Set email for the person
 *
 * @param s : email address of the person
 */
public void setEmail(String s){
   this.email = s;
}

/**
 * Email address of the person
 *
 * @return email : email address of the person
 */
public String getEmail(){
   return this.email;
}

/**
 * Set telephone number of the person
 *
 * @param s : a string of telephone number
 */
public void setTelephone(String s){
   this.telephone = s;
}

/**
 * Telephone number of the person
 *
 * @return telephone : a string of telephone number
 */
public String getTelephone(){
   return this.telephone;
}
```

```
/**
 * Set fax number for the person
 *
 * @param s : a string of fax number
 */
public void setFax(String s){
   this.fax = s;
}
/**
 * Fax number of the person
 *
 * @return s : a string of fax number
 */
public String getFax(){
   return this.fax;
}

/**
 * Set address for the person
 *
 * @param s : a string of address
 */
public void setAddress(String s){
   this.address = s;
}

/**
 * Address of the person
 *
 * @return address : a string of address
 */
public String getAddress(){
   return this.address;
}
/**
 * Set the list of minor jobs for the person
 *
```

```
  * @param v : a list of minor jobs
  */
 public void setMinorJobs(Vector v){
    this.minorJobs = v;
    for( int i=0; i<v.size(); i++ ){
       System.out.println(i+" : MINORJOB = "+v.elementAt(i));
    }
 }
 /**
  * The list of minor jobs for the person
  *
  * @return minorJobs : a vector of minor jobs
  */
 public Vector getMinorJobs(){
    return this.minorJobs;
 }
 /**
  * Set the list of major jobs for the person
  *
  * @param v: a vector of major jobs
  */
 public void setMajorJobs(Vector v){
    this.majorJobs = v;
    for( int i=0; i<v.size(); i++ ){
       System.out.println(i+" : MAJORJOB = "+v.elementAt(i));
    }
 }
 /**
  * The list of major jobs for the person
  *
  * @return majorJobs : a vector of major jobs
  */
 public Vector getMajorJobs(){
    return this.majorJobs;
 }
}
```

```
package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;
import symantec.itools.awt.MultiList;
import com.symantec.itools.swing.borders.EtchedBorder;


////////////////////////////////////////////////////////////////////////////////
/**
 * PersonnelFrame : to create, edit, and view personnel object
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Personnel
 */
////////////////////////////////////////////////////////////////////////////////
public class PersonnelFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_Personnel
{
   /**
    * selectedSkill : contains all selected skill from SkillTableFrame
    */
   public Vector selectedSkill = new Vector();

   /**
    * view : a flag to define the purpose of using this frame, eg. view or edit
    */
   boolean view = false;

   /**
    * edit : a flag to define the purpose of using this frame, eg. view or edit
    */
   boolean edit = false;
```

```
Personnel personnel = null;

/**
 * listHeadings : the 3 column headings of job multi list
 */
String[] listHeadings = {"Job Name", "Real Start Time", "Estimated
Duration"};

/**
 * Build PersonnelFrame and CasesFrrame launch this to create
personnel object
 */
        public PersonnelFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Personnel Data");
                getContentPane().setLayout(null);
    setVisible(false);
        setSize(500,540);

        JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel2.setText("ID");
                getContentPane().add(JLabel2);
                JLabel2.setForeground(java.awt.Color.black);
                JLabel2.setBounds(45,40,110,24);

        JLabel5.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel5.setText("E-mail Address");
                getContentPane().add(JLabel5);
                JLabel5.setForeground(java.awt.Color.black);
                JLabel5.setBounds(45,180,110,24);

        JLabel6.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel6.setText("Fax Number");
                getContentPane().add(JLabel6);
                JLabel6.setForeground(java.awt.Color.black);
                JLabel6.setBounds(45,250,110,24);

        JLabel7.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel7.setText("Address");
                getContentPane().add(JLabel7);
                JLabel7.setForeground(java.awt.Color.black);
                JLabel7.setBounds(45,285,110,24);

        JLabel8.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel8.setText("Name");
                getContentPane().add(JLabel8);
                JLabel8.setForeground(java.awt.Color.black);
                JLabel8.setBounds(45,75,110,24);

        JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
                JLabel9.setText("Security Level");
                getContentPane().add(JLabel9);
                JLabel9.setForeground(java.awt.Color.black);
                JLabel9.setBounds(45,145,110,24);
```

```
JLabel10.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel10.setText("Telephone Number");
getContentPane().add(JLabel10);
JLabel10.setForeground(java.awt.Color.black);
JLabel10.setBounds(45,215,110,24);

JLabel11.setHorizontalTextPosition(com.sun.java.swing.SwingCo
nstants.CENTER);

JLabel11.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.CENTER);
JLabel11.setText("Personnel Data");
getContentPane().add(JLabel11);
JLabel11.setForeground(java.awt.Color.black);
JLabel11.setFont(new Font("Dialog", Font.BOLD, 20));
JLabel11.setBounds(0,2,500,40);
getContentPane().add(IDTextField);
IDTextField.setBackground(java.awt.Color.white);
IDTextField.setForeground(java.awt.Color.black);
IDTextField.setBounds(155,40,300,24);
getContentPane().add(nameTextField);
nameTextField.setBackground(java.awt.Color.white);
nameTextField.setForeground(java.awt.Color.black);
nameTextField.setBounds(155,75,300,24);
getContentPane().add(clearButton);
clearButton.setBounds(0,0,0,0);
skillButton.setText("Skill");
skillButton.setActionCommand("Skill");
getContentPane().add(skillButton);
skillButton.setBounds(55,110,100,24);
exitButton.setText("Exit");
exitButton.setActionCommand("Exit");
getContentPane().add(exitButton);
exitButton.setBounds(0,0,0,0);
getContentPane().add(saveButton);
saveButton.setBounds(0,0,0,0);
getContentPane().add(skillComboBox);
skillComboBox.setBounds(155,110,300,24);
getContentPane().add(securityComboBox);
securityComboBox.setBounds(155,145,300,24);
getContentPane().add(emailTextField);
emailTextField.setBackground(java.awt.Color.white);
emailTextField.setForeground(java.awt.Color.black);
emailTextField.setBounds(155,180,300,24);
getContentPane().add(telephoneTextField);
telephoneTextField.setBackground(java.awt.Color.white);
telephoneTextField.setForeground(java.awt.Color.black);
telephoneTextField.setBounds(155,215,300,24);
getContentPane().add(addressTextField);
addressTextField.setBackground(java.awt.Color.white);
addressTextField.setForeground(java.awt.Color.black);
addressTextField.setBounds(155,285,300,24);
getContentPane().add(faxTextField);
faxTextField.setBackground(java.awt.Color.white);
faxTextField.setForeground(java.awt.Color.black);
faxTextField.setBounds(155,250,300,24);
getContentPane().add(JLabel1);
JLabel1.setBounds(0,0,0,0);
getContentPane().add(jobsScrollPane);
jobsScrollPane.setBounds(0,0,0,0);
getContentPane().add(JLabel3);
JLabel3.setBounds(0,0,0,0);
getContentPane().add(jobScrollPane);
jobScrollPane.setBounds(0,0,0,0);
onHandsPanel.setLayout(new
FlowLayout(FlowLayout.CENTER,5,5));
getContentPane().add(onHandsPanel);
onHandsPanel.setBounds(0,0,0,0);
getContentPane().add(minorRadioButton);
minorRadioButton.setBounds(0,0,0,0);
getContentPane().add(majorRadioButton);
majorRadioButton.setBounds(0,0,0,0);
```

```
            //$$ etchedBorder1.move(0,541);
            //}}

            //{{INIT_MENUS
            //}}

            //{{REGISTER_LISTENERS
            SymAction lSymAction = new SymAction();
            skillButton.addActionListener(lSymAction);
            clearButton.addActionListener(lSymAction);
            saveButton.addActionListener(lSymAction);
            exitButton.addActionListener(lSymAction);

            SymItem lSymItem = new SymItem();
            minorRadioButton.addItemListener(lSymItem);
            majorRadioButton.addItemListener(lSymItem);
            //}}

    //Set security level combobox from 0..5
    for( int i=0; i<SECURITY_LEVEL; i++ ){
      securityComboBox.addItem(i+"");
    }
      setSize(500,380);
            clearButton.setText("Clear");
            clearButton.setActionCommand("Delete");
            getContentPane().add(clearButton);
            clearButton.setBounds(24,330,73,24);
            saveButton.setText("Save");
            saveButton.setActionCommand("Save");
            getContentPane().add(saveButton);
            saveButton.setBounds(96,330,73,24);
            exitButton.setBounds(380,330,73,24);
  }

  /**
   * CasesFrame launch this to edit and view personnel object
   *
   * @param selectedFile : a selected file name
   * @param request : purpose of using this frame, eg. View or Edit
   */
  public PersonnelFrame(String selectedFile, String request){
    this();
            try{
              File aFile = new File(selectedFile);
              if( aFile.exists() ){
                FileInputStream fileInput = new
FileInputStream(aFile);
                ObjectInputStream oi = new ObjectInputStream(
fileInput );
                if( oi != null ){
                  personnel = (Personnel)oi.readObject();
                }
                oi.close();
                fileInput.close();
        }
            }
            catch( IOException io ){
              System.out.println("IOException: "+io);
            }
            catch( ClassNotFoundException c ){
        System.out.println("ClassNotFoundException: "+c);
            }
            if( request.equals("View") ){
              view = true;
            setSize(500,540);
              clearButton.setBounds(0,0,0,0);
              saveButton.setBounds(0,0,0,0);

              onHandsPanel.setBorder(etchedBorder1);
              onHandsPanel.setLayout(null);
              getContentPane().add(onHandsPanel);
              onHandsPanel.setBounds(155,320,300,30);
              minorRadioButton.setText("Minor Jobs");
              minorRadioButton.setSelected(true);
```

```
                onHandsPanel.add(minorRadioButton);
                minorRadioButton.setBounds(170,3,130,24);
                majorRadioButton.setText("Major Jobs");
                onHandsPanel.add(majorRadioButton);
                majorRadioButton.setBounds(20,3,130,24);
                majorRadioButton.setSelected(true);


JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConstants.RIG
HT);
                JLabel3.setText("On-hand Jobs");
                getContentPane().add(JLabel3);
                JLabel3.setForeground(java.awt.Color.black);
                JLabel3.setBounds(45,320,110,27);
                getContentPane().add(jobScrollPane);
                jobScrollPane.setBounds(45,355,410,130);
                jobScrollPane.getViewport().add(jobMultiList);
                jobMultiList.setBackground(java.awt.Color.white);
                jobMultiList.setBounds(0,0,407,127);
                exitButton.setBounds(384,495,73,24);

                try{
                  jobMultiList.setNumberOfCols(3);
        jobMultiList.setHeadings(listHeadings);
      }
      catch(Exception e){System.out.println(e);}
              }
              else if( request.equals("Edit") ){
                edit = true;
              }
              if( personnel != null ){
                setInitial(personnel);
              }
  }

      public PersonnelFrame(String sTitle)
      {
                this();
                setTitle(sTitle);
      }

      public void setVisible(boolean b)
      {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
      }

      static public void main(String args[])
      {
                (new PersonnelFrame()).setVisible(true);
      }

      public void addNotify()
      {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets and menu
bar
                Insets insets = getInsets();
                com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
                int menuBarHeight = 0;
                if (menuBar != null)
                        menuBarHeight =
menuBar.getPreferredSize().height;
```

```
            setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        //symantec.itools.awt.MultiList jobMultiList = new
symantec.itools.awt.MultiList();
        MultiList jobMultiList = new MultiList();
        com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel5 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel6 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel7 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel8 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel10 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel11 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField IDTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField nameTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JButton clearButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton skillButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton exitButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton saveButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JComboBox skillComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JComboBox securityComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JTextField emailTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField telephoneTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField addressTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField faxTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JScrollPane jobsScrollPane = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JScrollPane jobScrollPane = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JPanel onHandsPanel = new
com.sun.java.swing.JPanel();
        com.sun.java.swing.JRadioButton minorRadioButton = new
com.sun.java.swing.JRadioButton();
        com.sun.java.swing.JRadioButton majorRadioButton = new
com.sun.java.swing.JRadioButton();
        com.symantec.itools.swing.borders.EtchedBorder etchedBorder1 =
new com.symantec.itools.swing.borders.EtchedBorder();
        //}}

        //{{DECLARE_MENUS
        //}}

    /**
     * Add all selected skills from SkillTableFrame to skillComboBox
```

```
 *
 * @param v : a selected skill vector
 */
public void setSkillComboBox(Vector v){
  this.selectedSkill = new Vector();
  this.skillComboBox.removeAllItems();
  if( v.size() > 0 ){
        this.skillComboBox.addItem("ID : Name : Level");
          for( int i=0; i<v.size(); i++ ){
            this.skillComboBox.addItem(v.elementAt(i));
      this.selectedSkill.addElement(v.elementAt(i));
          }
  }
      }

      class SymAction implements java.awt.event.ActionListener
      {
            public void actionPerformed(java.awt.event.ActionEvent
event)
            {
                  Object object = event.getSource();
                  if (object == skillButton)
                        skillButton_actionPerformed(event);
                  else if (object == clearButton)
                        clearButton_actionPerformed(event);
                  else if (object == saveButton)
                        saveButton_actionPerformed(event);
                  else if (object == exitButton)
                        exitButton_actionPerformed(event);
            }
      }

/**
 * Launch SkillTableFrame
 */
      public void
skillButton_actionPerformed(java.awt.event.ActionEvent event)
      {
        (new SkillTableFrame(this)).setVisible(true);
  }

/**
 * Get a personnel object and save it under stakeholder
 */
      void saveButton_actionPerformed(java.awt.event.ActionEvent
event)
      {
    Personnel personnel = (Personnel)getPersonnelData();
    if( personnel != null ){
      try{
        FileOutputStream fileOutput = new
FileOutputStream(STAKEHOLDER+"\\"+IDTextField.getText());
        ObjectOutputStream oo = new ObjectOutputStream(fileOutput);
        if( oo != null ){
          oo.writeObject(personnel);
        }
        oo.flush();
        oo.close();
        fileOutput.close();
      }
      catch(IOException io){
        JOptionPane.showMessageDialog(this, "Sorry, saving is
unsuccessful", "Error Message", JOptionPane.ERROR_MESSAGE);
      }
      setVisible( false );
      dispose();
    }
      }

/**
 * Refresh all the textfields and combo boxes
 */
      void clearButton_actionPerformed(java.awt.event.ActionEvent
event)
```

```
        {
      //clear all fields
      IDTextField.setText("");
      nameTextField.setText("");
      skillComboBox.removeAllItems();
      securityComboBox.setSelectedIndex(0);
      emailTextField.setText("");
      telephoneTextField.setText("");
      addressTextField.setText("");
      faxTextField.setText("");
        }

  /**
   * Exit PersonnelFrame
   */
        void exitButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
     setVisible( false);
     dispose();
        }

  /**
   * Create and return a personnel object
   *
   * @return personnel object
   */
        public Personnel getPersonnelData(){
     Personnel personnel = new Personnel();
          try{
            String ID = IDTextField.getText();
            String name = nameTextField.getText();
            int security =
Integer.parseInt((String)securityComboBox.getSelectedItem());
            String email = emailTextField.getText();
            String tel = telephoneTextField.getText();
            String fax = faxTextField.getText();
            String address = addressTextField.getText();

            if( ID.equals("") ||
              name.equals("") ||
              selectedSkill.size() == 0 ||
              email.equals("") ||
              tel.equals("") ||
              fax.equals("") ||
              address.equals("") ){
        JOptionPane.showMessageDialog(this, "You did not complete all
the fields", "Error Message", JOptionPane.ERROR_MESSAGE);
        return null;
            }
            else{
              personnel.setID(ID);
              personnel.setName(name);
              personnel.setSkill(selectedSkill);
              personnel.setSecurityLevel(security);
              personnel.setEmail(email);
              personnel.setTelephone(tel);
              personnel.setAddress(address);
              personnel.setFax(fax);
            }
    }
    catch(Exception e){}
        return personnel;
        }

  /**
   * Set initial frame when it is launched to view and edit
   *
   * @param personnel : a personnel object
   */
        public void setInitial(Personnel personnel){
          if( personnel != null ){
    IDTextField.setText(personnel.getID());
    nameTextField.setText(personnel.getName());
```

```
        setSkillComboBox(personnel.getSkill());

securityComboBox.setSelectedItem(""+personnel.getSecurityLevel());
        emailTextField.setText(personnel.getEmail());
        telephoneTextField.setText(personnel.getTelephone());
        addressTextField.setText(personnel.getAddress());
        faxTextField.setText(personnel.getFax());
            }
            if( view ){
        IDTextField.setEditable(false);
        nameTextField.setEditable(false);
        skillButton.setEnabled(false);
        securityComboBox.setEnabled(false);
        emailTextField.setEditable(false);
        telephoneTextField.setEditable(false);
        addressTextField.setEditable(false);
        faxTextField.setEditable(false);
            }
        }

    class SymItem implements java.awt.event.ItemListener
        {
            public void itemStateChanged(java.awt.event.ItemEvent event)
                {
                        Object object = event.getSource();
                        if (object == minorRadioButton)

        minorRadioButton_itemStateChanged(event);
                        else if (object == majorRadioButton)

        majorRadioButton_itemStateChanged(event);
                }
        }

    /**
     * View minor jobs of selected personnel object to the list
     */
```



```
    public void
minorRadioButton_itemStateChanged(java.awt.event.ItemEvent event)
        {
      if( event.getStateChange() == event.SELECTED ){
        majorRadioButton.setSelected(false);
        jobMultiList.clear();
        Vector v = new Vector();
        if( personnel != null ){
          v = (Vector)personnel.getMinorJobs();
          for( int j=0; j<v.size(); j++ ){
            StepContent sc = (StepContent)v.elementAt(j);
            System.out.println("stepNamme = "+sc.getStepName());
            jobMultiList.addTextCell(j, 0, sc.getStepName());
            jobMultiList.addTextCell(j, 1, sc.getRealStartTime());
            jobMultiList.addTextCell(j, 2, ""+sc.getDuration());
          }
        }
      }
        }

    /**
     * View major jobs of selected personnel object to the list
     */
        public void
majorRadioButton_itemStateChanged(java.awt.event.ItemEvent event)
        {
      if( event.getStateChange() == event.SELECTED ){
        minorRadioButton.setSelected(false);
        jobMultiList.clear();
        Vector v = new Vector();
        if( personnel != null ){
          v = (Vector)personnel.getMajorJobs();
          for( int j=0; j<v.size(); j++ ){
            StepContent sc = (StepContent)v.elementAt(j);
            System.out.println("stepNamme = "+sc.getStepName());
            jobMultiList.addTextCell(j, 0, sc.getStepName());
            jobMultiList.addTextCell(j, 1, sc.getRealStartTime());
```

```
                jobMultiList.addTextCell(j, 2, ""+sc.getDuration());
            }
        }
    }
        }
}
```



```
package Cases;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;

////////////////////////////////////////////////////////////////////////////////
/**
 * ProjectSchemaFrame : allows a user to create step types, component
types,
 * evolution history loop, and dependency which are stored in step.cfg,
 * component.cfg, loop.cfg, and dependency.cfg respectively
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_ProjectSchema
 */
////////////////////////////////////////////////////////////////////////////////
public class ProjectSchemaFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_ProjectSchema
{
  /**
   * saveTab : flags of 4 tabs and to confirm that every tab should be saved
before exit the frame
   */
  boolean[] saveTab = {true, true, true, true};

  /**
   * stepVector : contains all step type objects
   */
  public Vector stepVector = new Vector();

  /**
   * compVector : contains all component type objects
   */
  public Vector compVector = new Vector();

  /**
   * EHLVector : contains all EHL objects
   */
  public Vector EHLVector = new Vector();

  /**
   * stepHashtable : with the keys and the objects are step type IDs and
step type objects
   */
  public Hashtable stepHashtable = new Hashtable();

  /**
```

```
    * compHashtable : with the keys and the objects are component type
IDs and component type objects
    */
   public Hashtable compHashtable = new Hashtable();

   /**
    * EHLHashtable : with the keys and the objects are EHL IDs and EHL
objects
    */
   public Hashtable EHLHashtable = new Hashtable();

   /**
    * depenHashtable : with the keys and the objects are dependency IDs
and dependency objects
    */
   public Hashtable depenHashtable = new Hashtable();

   /**
    * testIndex :  to keep track the secondary input of each dependency
object to be set correctly
    */
   public int testIndex = 0;

   /**
    * selectedStepIndex : to make sure that getting the correct step type
object
    */
   public int selectedStepIndex;

   /**
    * selectedCompIndex : to make sure that getting the correct component
type object
    */
   public int selectedCompIndex;

   /**
    * selectedEHLIndex : to make sure that getting the correct EHL object
    */
   public int selectedEHLIndex;

   /**
    * selectedDepenStepIndex : to make sure that getting the correct
dependency object
    */
   public int selectedDepenStepIndex;

   /**
    * pathName : the path of the current project
    */
   public String pathName;

   /**
    * listModel : monitor all step types in EHL panel
    */
   public DefaultListModel listModel = new DefaultListModel();

   /**
    * Build ProjectSchemaFrame
    */
        public ProjectSchemaFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Project Schema");
                getContentPane().setLayout(null);
                setSize(600,450);
```

```
setVisible(false);
mainPanel.setLayout(new GridLayout(1,1,0,0));
getContentPane().add(mainPanel);
mainPanel.setBounds(10,0,580,360);
mainPanel.add(configManagTabbedPane);

configManagTabbedPane.setForeground(java.awt.Color.black);
configManagTabbedPane.setBounds(0,0,580,360);
stepTypePanel.setLayout(null);
configManagTabbedPane.add(stepTypePanel);
stepTypePanel.setBounds(2,27,575,330);

JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel1.setText("Step Type ID");
stepTypePanel.add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setBounds(62,70,130,22);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel2.setText("Step Type Name");
stepTypePanel.add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(62,110,130,22);

JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel3.setText("Step Type Description");
stepTypePanel.add(JLabel3);
JLabel3.setForeground(java.awt.Color.black);
JLabel3.setBounds(62,190,130,22);

JLabel4.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel4.setText("Existing Step Types");
stepTypePanel.add(JLabel4);
JLabel4.setForeground(java.awt.Color.black);
JLabel4.setBounds(62,150,130,22);
stepAddButton.setText("Add");
stepAddButton.setActionCommand("jbutton");
stepTypePanel.add(stepAddButton);
stepAddButton.setBounds(12,297,75,22);
stepDeleteButton.setText("Delete");
stepDeleteButton.setActionCommand("jbutton");
stepTypePanel.add(stepDeleteButton);
stepDeleteButton.setBounds(168,297,75,22);
stepClearButton.setText("Clear");
stepClearButton.setActionCommand("jbutton");
stepTypePanel.add(stepClearButton);
stepClearButton.setBounds(246,297,75,22);
stepEditButton.setText("Edit");
stepEditButton.setActionCommand("jbutton");
stepTypePanel.add(stepEditButton);
stepEditButton.setBounds(90,297,75,22);
stepTypePanel.add(stepIDTextField);
stepIDTextField.setBounds(195,70,300,22);
stepTypePanel.add(stepNameTextField);
stepNameTextField.setBounds(195,110,300,22);
stepTypePanel.add(existedStepComboBox);
existedStepComboBox.setBounds(195,150,300,22);
stepDescriptionTextArea.setLineWrap(true);
stepDescriptionTextArea.setWrapStyleWord(true);
stepTypePanel.add(stepDescriptionTextArea);
stepDescriptionTextArea.setBounds(195,190,300,80);
stepSaveButton.setText("Save");
stepSaveButton.setActionCommand("Save");
stepTypePanel.add(stepSaveButton);
stepSaveButton.setBounds(492,297,75,22);

JLabel16.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel16.setText("Project Label:");
stepTypePanel.add(JLabel16);
```

```
JLabel16.setForeground(java.awt.Color.black);
JLabel16.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel16.setBounds(2,6,280,24);
stepLabel.setText("Label");
stepTypePanel.add(stepLabel);
stepLabel.setForeground(java.awt.Color.black);
stepLabel.setFont(new Font("Dialog", Font.BOLD, 18));
stepLabel.setBounds(292,6,280,24);
componentTypePanel.setLayout(null);
configManagTabbedPane.add(componentTypePanel);
componentTypePanel.setBackground(new
java.awt.Color(204,204,204));
componentTypePanel.setBounds(2,27,575,330);
componentTypePanel.setVisible(false);

JLabel5.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel5.setText("Component Type ID");
componentTypePanel.add(JLabel5);
JLabel5.setForeground(java.awt.Color.black);
JLabel5.setBounds(75,70,145,22);

JLabel6.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel6.setText("Component Type Name");
componentTypePanel.add(JLabel6);
JLabel6.setForeground(java.awt.Color.black);
JLabel6.setBounds(75,110,144,22);

JLabel7.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel7.setText("Component Type Description");
componentTypePanel.add(JLabel7);
JLabel7.setForeground(java.awt.Color.black);
JLabel7.setBounds(50,190,170,22);

JLabel8.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel8.setText("Existing Component Types");
componentTypePanel.add(JLabel8);
JLabel8.setForeground(java.awt.Color.black);
JLabel8.setBounds(50,150,170,22);
compAddButton.setText("Add");
compAddButton.setActionCommand("jbutton");
componentTypePanel.add(compAddButton);
compAddButton.setBounds(12,297,75,22);
compDeleteButton.setText("Delete");
compDeleteButton.setActionCommand("jbutton");
componentTypePanel.add(compDeleteButton);
compDeleteButton.setBounds(168,297,75,22);
compClearButton.setText("Clear");
compClearButton.setActionCommand("jbutton");
componentTypePanel.add(compClearButton);
compClearButton.setBounds(246,297,75,22);
compEditButton.setText("Edit");
compEditButton.setActionCommand("jbutton");
componentTypePanel.add(compEditButton);
compEditButton.setBounds(90,297,75,22);
componentTypePanel.add(compIDTextField);
compIDTextField.setBounds(225,70,300,22);
componentTypePanel.add(compNameTextField);
compNameTextField.setBounds(225,110,300,22);
componentTypePanel.add(existedCompComboBox);
existedCompComboBox.setBounds(225,150,300,22);
compDescriptionTextArea.setLineWrap(true);
compDescriptionTextArea.setWrapStyleWord(true);
componentTypePanel.add(compDescriptionTextArea);
compDescriptionTextArea.setBounds(225,190,300,80);
compSaveButton.setText("Save");
compSaveButton.setActionCommand("Save");
componentTypePanel.add(compSaveButton);
compSaveButton.setBounds(492,297,75,22);
```

```
JLabel17.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel17.setText("Project Label:");
componentTypePanel.add(JLabel17);
JLabel17.setForeground(java.awt.Color.black);
JLabel17.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel17.setBounds(0,6,280,24);
compLabel.setText("Label");
componentTypePanel.add(compLabel);
compLabel.setForeground(java.awt.Color.black);
compLabel.setFont(new Font("Dialog", Font.BOLD, 18));
compLabel.setBounds(294,6,280,24);
EHLPanel.setDoubleBuffered(false);
EHLPanel.setLayout(null);
configManagTabbedPane.add(EHLPanel);
EHLPanel.setBounds(2,27,575,330);
EHLPanel.setVisible(false);

JLabel13.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel13.setText("Evolution Process Name");
EHLPanel.add(JLabel13);
JLabel13.setForeground(java.awt.Color.black);
JLabel13.setBounds(60,70,150,22);

JLabel14.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel14.setText("Evolution Process");
EHLPanel.add(JLabel14);
JLabel14.setForeground(java.awt.Color.black);
JLabel14.setBounds(60,210,150,22);
EHLAddButton.setText("Add");
EHLAddButton.setActionCommand("jbutton");
EHLPanel.add(EHLAddButton);
EHLAddButton.setBounds(10,297,75,22);
EHLDeleteButton.setText("Delete");
EHLDeleteButton.setActionCommand("jbutton");
EHLPanel.add(EHLDeleteButton);
EHLDeleteButton.setBounds(166,297,75,22);
EHLClearButton.setText("Clear");
EHLClearButton.setActionCommand("jbutton");
EHLPanel.add(EHLClearButton);
EHLClearButton.setBounds(244,297,75,22);
EHLEditButton.setText("Edit");
EHLEditButton.setActionCommand("jbutton");
EHLPanel.add(EHLEditButton);
EHLEditButton.setBounds(88,297,75,22);
EHLPanel.add(EHLNameTextField);
EHLNameTextField.setBounds(215,70,300,22);
EHLPathTextField.setDoubleBuffered(true);
EHLPathTextField.setEditable(false);
EHLPanel.add(EHLPathTextField);
EHLPathTextField.setBackground(java.awt.Color.white);
EHLPathTextField.setForeground(java.awt.Color.black);
EHLPathTextField.setBounds(215,210,300,22);
EHLDoneButton.setText("Done");
EHLDoneButton.setActionCommand("Done");
EHLPanel.add(EHLDoneButton);
EHLDoneButton.setBounds(493,297,75,22);

label.setHorizontalAlignment(com.sun.java.swing.SwingConstants
.RIGHT);
label.setText("Project Label:");
EHLPanel.add(label);
label.setForeground(java.awt.Color.black);
label.setFont(new Font("Dialog", Font.BOLD, 18));
label.setBounds(2,6,280,24);
projectLabel.setText("Label");
EHLPanel.add(projectLabel);
projectLabel.setForeground(java.awt.Color.black);
projectLabel.setFont(new Font("Dialog", Font.BOLD,
18));
projectLabel.setBounds(294,6,280,24);
```

```
JLabel15.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel15.setText("Existing Evolution Process");
EHLPanel.add(JLabel15);
JLabel15.setForeground(java.awt.Color.black);
JLabel15.setBounds(60,250,150,22);
EHLPanel.add(existedEHLComboBox);
existedEHLComboBox.setBounds(215,250,300,22);

JLabel20.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel20.setText("Step Types");
JLabel20.setNextFocusableComponent(compAddButton);
EHLPanel.add(JLabel20);
JLabel20.setForeground(java.awt.Color.black);
JLabel20.setBounds(60,110,150,22);
stepTypesScrollPane.setOpaque(true);
EHLPanel.add(stepTypesScrollPane);
stepTypesScrollPane.setBounds(215,110,300,80);
stepTypesScrollPane.getViewport().add(stepTypesList);
stepTypesList.setBounds(0,0,297,77);
dependencyPanel.setLayout(null);
configManagTabbedPane.add(dependencyPanel);
dependencyPanel.setBounds(2,27,575,330);
dependencyPanel.setVisible(false);

JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel9.setText("Step Types");
dependencyPanel.add(JLabel9);
JLabel9.setForeground(java.awt.Color.black);
JLabel9.setBounds(30,110,240,22);

JLabel10.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel10.setText("Output Component Type");
dependencyPanel.add(JLabel10);
JLabel10.setForeground(java.awt.Color.black);
JLabel10.setBounds(30,150,240,22);

JLabel12.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel12.setText("Primary Input Component Type");
dependencyPanel.add(JLabel12);
JLabel12.setForeground(java.awt.Color.black);
JLabel12.setBounds(30,190,240,22);
depenOKButton.setText("OK");
depenOKButton.setActionCommand("jbutton");
dependencyPanel.add(depenOKButton);
depenOKButton.setBounds(390,290,75,22);
depenCancelButton.setText("Cancel");
depenCancelButton.setActionCommand("jbutton");
dependencyPanel.add(depenCancelButton);
depenCancelButton.setBounds(470,290,75,22);
depenSecondaryTextField.setEditable(false);
dependencyPanel.add(depenSecondaryTextField);

depenSecondaryTextField.setBackground(java.awt.Color.white);
depenSecondaryTextField.setBounds(275,230,270,22);
depenOutputTextField.setEditable(false);
dependencyPanel.add(depenOutputTextField);

depenOutputTextField.setBackground(java.awt.Color.white);

depenOutputTextField.setForeground(java.awt.Color.black);
depenOutputTextField.setFont(new Font("SansSerif",
Font.PLAIN, 12));
depenOutputTextField.setBounds(275,150,270,22);
dependencyPanel.add(depenStepComboBox);
depenStepComboBox.setBounds(275,110,270,22);
depenPrimaryTextField.setEditable(false);
dependencyPanel.add(depenPrimaryTextField);
```

```
depenPrimaryTextField.setBackground(java.awt.Color.white);

depenPrimaryTextField.setForeground(java.awt.Color.black);
        depenPrimaryTextField.setBounds(275,190,270,22);

JLabel18.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
        JLabel18.setText("Project Label:");
        dependencyPanel.add(JLabel18);
        JLabel18.setForeground(java.awt.Color.black);
        JLabel18.setFont(new Font("Dialog", Font.BOLD, 18));
        JLabel18.setBounds(0,6,280,24);
        depLabel.setText("Label");
        dependencyPanel.add(depLabel);
        depLabel.setForeground(java.awt.Color.black);
        depLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        depLabel.setBounds(294,6,280,24);

JLabel19.setHorizontalTextPosition(com.sun.java.swing.SwingCo
nstants.CENTER);

JLabel19.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
        JLabel19.setText("Evolution Process");
        dependencyPanel.add(JLabel19);
        JLabel19.setForeground(java.awt.Color.black);
        JLabel19.setBounds(30,70,240,22);
        dependencyPanel.add(depenEHLComboBox);
        depenEHLComboBox.setBounds(275,70,270,22);
        secondaryButton.setText("Secondary Input Component
Type(s)");
        secondaryButton.setActionCommand("Secondary Input
Component Type(s)");
        dependencyPanel.add(secondaryButton);
        secondaryButton.setBounds(30,230,240,22);
        configManagTabbedPane.setSelectedIndex(0);
```

```
configManagTabbedPane.setSelectedComponent(stepTypePanel);
        try {
                configManagTabbedPane.setTitleAt(0,"Step
Type");
        }
        catch(ArrayIndexOutOfBoundsException e) {
System.out.println(e);
        }

configManagTabbedPane.setTitleAt(1,"Component Type");

configManagTabbedPane.setTitleAt(2,"Evolution Process");

configManagTabbedPane.setTitleAt(3,"Dependency");
        doneButton.setText("Finish");
        doneButton.setActionCommand("OK");
        getContentPane().add(doneButton);
        doneButton.setBounds(262,390,75,22);
        //}}

        //{{INIT_MENUS
        //}}

        //{{REGISTER_LISTENERS
        SymAction lSymAction = new SymAction();
        stepAddButton.addActionListener(lSymAction);
        stepEditButton.addActionListener(lSymAction);
        stepDeleteButton.addActionListener(lSymAction);
        stepClearButton.addActionListener(lSymAction);
        compAddButton.addActionListener(lSymAction);
        compEditButton.addActionListener(lSymAction);
        compDeleteButton.addActionListener(lSymAction);
        compClearButton.addActionListener(lSymAction);
        EHLAddButton.addActionListener(lSymAction);
        EHLEditButton.addActionListener(lSymAction);
        EHLDeleteButton.addActionListener(lSymAction);
```

```
EHLClearButton.addActionListener(lSymAction);
depenOKButton.addActionListener(lSymAction);
depenCancelButton.addActionListener(lSymAction);
doneButton.addActionListener(lSymAction);
EHLDoneButton.addActionListener(lSymAction);
stepSaveButton.addActionListener(lSymAction);
compSaveButton.addActionListener(lSymAction);
SymItem lSymItem = new SymItem();
depenStepComboBox.addItemListener(lSymItem);
depenEHLComboBox.addItemListener(lSymItem);
existedStepComboBox.addItemListener(lSymItem);
existedCompComboBox.addItemListener(lSymItem);
existedEHLComboBox.addItemListener(lSymItem);
SymMouse aSymMouse = new SymMouse();
stepTypesList.addMouseListener(aSymMouse);
secondaryButton.addActionListener(lSymAction);
//}}

stepTypesList.setModel( listModel );
setEnabled(3, false);
}

/**
 * Is launched when a user wants to create or edit step.cfg,
component.cfg, loop.cfg, or dependency.cfg
 *
 * @param selectedProject : current project name
 * @param pathName : the path of the current project
 */
public ProjectSchemaFrame( String selectedProject, String pathName ){
  this();
  this.setProjectLabel( selectedProject );
  this.pathName = pathName;
  this.readInputFiles( this.pathName );
}

public void setVisible(boolean b)
{
        if (b)
                setLocation(50, 50);
        super.setVisible(b);
}

static public void main(String args[])
{
        (new ProjectSchemaFrame()).setVisible(true);
}

public void addNotify()
{
        // Record the size of the window prior to calling parents
addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
                return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu
bar
        Insets insets = getInsets();
        com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
                menuBarHeight =
menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
}

// Used by addNotify
```

```
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
com.sun.java.swing.JPanel mainPanel = new
com.sun.java.swing.JPanel();
com.sun.java.swing.JTabbedPane configManagTabbedPane = new
com.sun.java.swing.JTabbedPane();
com.sun.java.swing.JPanel stepTypePanel = new
com.sun.java.swing.JPanel();
com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel4 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JButton stepAddButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton stepDeleteButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton stepClearButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton stepEditButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JTextField stepIDTextField = new
com.sun.java.swing.JTextField();
com.sun.java.swing.JTextField stepNameTextField = new
com.sun.java.swing.JTextField();
com.sun.java.swing.JComboBox existedStepComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JTextArea stepDescriptionTextArea = new
com.sun.java.swing.JTextArea();
com.sun.java.swing.JButton stepSaveButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JLabel JLabel16 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel stepLabel = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JPanel componentTypePanel = new
com.sun.java.swing.JPanel();
com.sun.java.swing.JLabel JLabel5 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel6 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel7 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel8 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JButton compAddButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton compDeleteButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton compClearButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JButton compEditButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JTextField compIDTextField = new
com.sun.java.swing.JTextField();
com.sun.java.swing.JTextField compNameTextField = new
com.sun.java.swing.JTextField();
com.sun.java.swing.JComboBox existedCompComboBox = new
com.sun.java.swing.JComboBox();
com.sun.java.swing.JTextArea compDescriptionTextArea = new
com.sun.java.swing.JTextArea();
com.sun.java.swing.JButton compSaveButton = new
com.sun.java.swing.JButton();
com.sun.java.swing.JLabel JLabel17 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel compLabel = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JPanel EHLPanel = new
com.sun.java.swing.JPanel();
```

```
        com.sun.java.swing.JLabel JLabel13 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel14 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JButton EHLAddButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton EHLDeleteButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton EHLClearButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton EHLEditButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JTextField EHLNameTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField EHLPathTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JButton EHLDoneButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel label = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel projectLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel15 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox existedEHLComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel20 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JScrollPane stepTypesScrollPane = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JList stepTypesList = new
com.sun.java.swing.JList();
        com.sun.java.swing.JPanel dependencyPanel = new
com.sun.java.swing.JPanel();
        com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel10 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel12 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JButton depenOKButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton depenCancelButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JTextField depenSecondaryTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField depenOutputTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JComboBox depenStepComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JTextField depenPrimaryTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel18 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel depLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel19 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox depenEHLComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JButton secondaryButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton doneButton = new
com.sun.java.swing.JButton();
        //}}

        //{{DECLARE_MENUS
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
```

```
public void actionPerformed(java.awt.event.ActionEvent
event)
{
        Object object = event.getSource();
        if (object == stepAddButton)

stepAddButton_actionPerformed(event);
        else if (object == stepEditButton)

stepEditButton_actionPerformed(event);
        else if (object == stepDeleteButton)

stepDeleteButton_actionPerformed(event);
        else if (object == stepClearButton)

stepClearButton_actionPerformed(event);
        else if (object == compAddButton)

compAddButton_actionPerformed(event);
        else if (object == compEditButton)

compEditButton_actionPerformed(event);
        else if (object == compDeleteButton)

compDeleteButton_actionPerformed(event);
        else if (object == compClearButton)

compClearButton_actionPerformed(event);
        else if (object == EHLAddButton)

EHLAddButton_actionPerformed(event);
        else if (object == EHLEditButton)

EHLEditButton_actionPerformed(event);
        else if (object == EHLDeleteButton)

EHLDeleteButton_actionPerformed(event);
        else if (object == EHLClearButton)

EHLClearButton_actionPerformed(event);
        else if (object == depenOKButton)

depenOKButton_actionPerformed(event);
        else if (object == depenCancelButton)

depenCancelButton_actionPerformed(event);
        else if (object == doneButton)
                doneButton_actionPerformed(event);
        else if (object == EHLDoneButton)

EHLDoneButton_actionPerformed(event);
        else if (object == stepSaveButton)

stepSaveButton_actionPerformed(event);
        if (object == compSaveButton)

compSaveButton_actionPerformed(event);
        if (object == secondaryButton)

secondaryButton_actionPerformed(event);
        }
}

/**
 * Set the current project name for each panel's title
 *
 * @param selectedProject : the current project name
 */
public void setProjectLabel( String selectedProject ){
  this.projectLabel.setText( selectedProject );
  this.stepLabel.setText( selectedProject );
  this.compLabel.setText( selectedProject );
  this.depLabel.setText( selectedProject );
}
```

```
/**
 * Add new step type object into combo box and stepVector
 */
        public void
stepAddButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    String stepID = stepIDTextField.getText();
    String stepName = stepNameTextField.getText();
    String stepDescription = stepDescriptionTextArea.getText();
        StepType stepType = new StepType( stepID, stepName,
stepDescription );

        this.stepVector.addElement( stepType );
        this.setListModel( this.stepVector );

          this.setStepComboBox( this.stepVector );
          saveTab[0] = false;

          //Clean up the frame
          this.stepClearButton_actionPerformed( null );
        }

/**
 * Edit selected step type object and delete before add it into combo box
and stepVector
 */
        public void
stepEditButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          if( this.selectedStepIndex > 0 ){
      String stepID = stepIDTextField.getText();
      String stepName = stepNameTextField.getText();
      String stepDescription = stepDescriptionTextArea.getText();
        StepType stepType = new StepType( stepID, stepName,
stepDescription );
```



```
          this.stepVector.setElementAt( stepType, this.selectedStepIndex-
1 );
          this.setListModel( this.stepVector );
            this.setStepComboBox( this.stepVector );
          saveTab[0] = false;
      }

          //Clean up the frame
          this.stepClearButton_actionPerformed( null );
  }

  /**
   * Delete selected step type object and remove it from combo box and
stepVector
   */
        public void
stepDeleteButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          if( this.selectedStepIndex > 0 ){
            this.stepVector.removeElementAt( this.selectedStepIndex - 1
);
        this.setListModel( this.stepVector );
            this.setStepComboBox( this.stepVector );
          saveTab[0] = false;
      }

          //Clean up the frame
          this.stepClearButton_actionPerformed( null );
        }

  /**
   * Refresh all text fields in step panel
   */
        public void
stepClearButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          this.stepIDTextField.setText("");
```

```
            this.stepNameTextField.setText("");
            this.stepDescriptionTextArea.setText("");
            if( this.existedStepComboBox.getItemCount() > 0 ){
              this.existedStepComboBox.setSelectedIndex(0);
            }
          }

  /**
   * Save all step type objects in stepVector in step.cfg file
   */
        public void
stepSaveButton_actionPerformed(java.awt.event.ActionEvent event)
        {
      try{
        FileOutputStream stepFileOut = new FileOutputStream(
this.pathName+"\\step.cfg" );
        ObjectOutputStream stepOut= new ObjectOutputStream(
stepFileOut );

              if( this.stepVector.size()> 0 ){
                if(stepOut != null ){
                  stepOut.writeObject( this.stepVector );
                  this.setStepComboBox( this.stepVector );
                    saveTab[0] = true;
          }
                stepOut.flush();
                stepOut.close();
                stepFileOut.close();
              }
      }
      catch( FileNotFoundException fe ){
        debug("FileNotFoundException: "+fe);
      }
      catch( IOException e ){
        debug("IOException: "+e);
      }
```



```
      //Clean up the frame
            this.stepClearButton_actionPerformed( null );
          }

  /**
   * Add new component type object into combo box and compVector
   */
        public void
compAddButton_actionPerformed(java.awt.event.ActionEvent event)
        {
      String compID = compIDTextField.getText();
      String compName = compNameTextField.getText();
      String compDescription = compDescriptionTextArea.getText();

        ComponentType compType = new ComponentType( compID,
compName, compDescription );
        this.compVector.addElement( compType );

            this.setCompComboBox( this.compVector );
            saveTab[1] = false;

            //Clean up the frame
            this.stepClearButton_actionPerformed( null );
  }

  /**
   * Edit selected component type object and delete before add it into
combo box and compVector
   */
        public void
compEditButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            if( this.selectedCompIndex > 0 ){
      String compID = compIDTextField.getText();
      String compName = compNameTextField.getText();
      String compDescription = compDescriptionTextArea.getText();
```

```
            ComponentType compType = new ComponentType( compID,
compName, compDescription );
            this.compVector.setElementAt( compType,
this.selectedCompIndex-1 );

              this.setCompComboBox( this.compVector );
            saveTab[1] = false;
        }

            //Clean up the frame
            this.compClearButton_actionPerformed( null );
          }

    /**
     * Delete selected component type object and remove it from combo box
and compVector
     */
          public void
compDeleteButton_actionPerformed(java.awt.event.ActionEvent event)
          {
            if( this.selectedCompIndex > 0 ){
              this.compVector.removeElementAt( this.selectedCompIndex
- 1 );
              this.setCompComboBox( this.compVector );
            saveTab[1] = false;
        }

            //Clean up the frame
            this.compClearButton_actionPerformed( null );
          }

    /**
     * Refresh all text fields in component panel
     */
          public void
compClearButton_actionPerformed(java.awt.event.ActionEvent event)
          {
            this.compIDTextField.setText("");
            this.compNameTextField.setText("");
            this.compDescriptionTextArea.setText("");
            if( this.existedCompComboBox.getItemCount() > 0 ){
              this.existedCompComboBox.setSelectedIndex(0);
            }
          }

    /**
     * Save all component type objects in compVector in component.cfg file
     */
          public void
compSaveButton_actionPerformed(java.awt.event.ActionEvent event)
          {
        try{
          FileOutputStream compFileOut = new FileOutputStream(
this.pathName+"\\component.cfg");
          ObjectOutputStream compOut= new ObjectOutputStream(
compFileOut );
              if( this.compVector.size()> 0 ){
                if(compOut != null ){
                  compOut.writeObject(this.compVector);
                    saveTab[1] = true;
            this.setCompComboBox( this.compVector );
                }
                compOut.flush();
                compOut.close();
                compFileOut.close();
              }
        }
        catch( FileNotFoundException fe ){
          debug("FileNotFoundException: "+fe);
        }
        catch( IOException e ){
          debug("IOException: "+e);
        }
        //Clear the Frame
```

```
            this.compClearButton_actionPerformed( null );
        }

    /**
     * Add new EHL object into combo box and EHLVector
     */
        public void
EHLAddButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    String EHLName = EHLNameTextField.getText();
    String EHLPath = EHLPathTextField.getText();

    StringTokenizer st = new StringTokenizer( EHLPath, "," );
            Vector tokenizeVector = new Vector();
            while( st.hasMoreTokens() ){
              tokenizeVector.addElement( st.nextToken() );
            }

    if( tokenizeVector.size()< 2 ){
      JOptionPane.showMessageDialog(this, "The loop must have at least
2 steps!",
                              "Error", JOptionPane.ERROR_MESSAGE);
      return;
    }
    else{
            EHL ehl = new EHL( EHLName, EHLPath );
            this.EHLVector.addElement( ehl );
              this.setEHLComboBox( this.EHLVector );
            saveTab[2] = false;

              //Clean up the frame
              this.EHLClearButton_actionPerformed( null );
    }
        }

    /**
     * Edit selected EHL object and delete before add it into combo box and
EHLVector
     */
        public void
EHLEditButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            if( this.selectedEHLIndex > 0 ){
        String EHLName = EHLNameTextField.getText();
        String EHLPath = EHLPathTextField.getText();
            EHL ehl = new EHL( EHLName, EHLPath );
            this.EHLVector.setElementAt( ehl, this.selectedEHLIndex-1 );
              this.setEHLComboBox( this.EHLVector );
            saveTab[2] = false;
    }

            //Clean up the frame
            this.EHLClearButton_actionPerformed( null );
        }

    /**
     * Delete selected EHL object and remove it from combo box and
EHLVector
     */
        public void
EHLDeleteButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            if( this.selectedEHLIndex > 0 ){
              this.EHLVector.removeElementAt( this.selectedEHLIndex - 1
);
              this.setEHLComboBox( this.EHLVector );
            saveTab[2] = false;
    }

            //Clean up the frame
            this.EHLClearButton_actionPerformed( null );
        }
```

```
/**
 * Refresh all text fields in EHL panel
 */
        public void
EHLClearButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          this.EHLNameTextField.setText("");
          this.EHLPathTextField.setText("");
        }

/**
 * Save all EHL objects in EHLVector in loop.cfg file and go directly
into dependency panel
 */
        public void
EHLDoneButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    try{
      FileOutputStream EHLFileOut = new FileOutputStream(
this.pathName+"\\loop.cfg" );
      ObjectOutputStream EHLOut = new ObjectOutputStream(
EHLFileOut );

            if( this.EHLVector.size()> 0 ){
              if(EHLOut != null ){
              EHLOut.writeObject( this.EHLVector );
          saveTab[2] = true;
           this.readDepFile();
                this.setDepenEHLComboBox( this.EHLVector);

                //Only use for EHL tabbedPane
                this.setEnabled( 3, true);
                this.configManagTabbedPane.setSelectedIndex( 3 );

                this.setEnabled( 0, false );
                this.setEnabled( 1, false );
                this.setEnabled( 2, false );
                }
            EHLOut.flush();
            EHLOut.close();
            EHLFileOut.close();
              }
    }
    catch( FileNotFoundException fe ){
      debug("FileNotFoundException: "+fe);
    }
    catch( IOException e ){
      debug("IOException: "+e);
    }

    //Clean up the frame
          this.EHLClearButton_actionPerformed( null );
        }

/**
 * Save all dependency objects in depenHashtable in dependency.cfg file
 * Before saving it, check all secondary inputs of each dependency to be
set or not.
 * The warning message will show up if one of dependency objects
 * didn't set the secondary input
 */
        public void
depenOKButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    saveLastDep();

          Vector v = new Vector();
          Vector v2 = new Vector();
    Dependency dep = null;

    Enumeration enum = (Enumeration)listModel.elements();
    while( enum.hasMoreElements() ){
      if(
!this.depenHashtable.containsKey((String)enum.nextElement())){
```

```
                v.addElement("");
                v2.addElement("");
            }
        }
        enum = null;
        try{
            FileOutputStream depFileOut = new FileOutputStream(
this.pathName+"\\dependency.cfg" );
            ObjectOutputStream dependencyOut = new ObjectOutputStream(
depFileOut );
            if( this.depenHashtable.size() > 0 ){
                    enum = this.depenHashtable.elements();
                    while( enum.hasMoreElements() ){
                        Dependency d = (Dependency)enum.nextElement();
                if( d!= null ){
                    v.addElement(((String)d.getSecondaryInput()).trim());
                        v2.addElement(((String)d.getStep()).trim());
                    }
                    else{
                        v.addElement("");
                        v2.addElement("");
                    }
                    }

                    for( int i=0; i<v.size(); i++ ){
                        if( ((String)v.elementAt(i)).equals("") ){
                            if(
!((String)v2.elementAt(i)).equals(STEP_TYPE_TITLE) ){
                        JOptionPane.showMessageDialog(this, "Can not process
since \nsome steps do not have the secondary \n input component types!",
                                        "Error",
JOptionPane.ERROR_MESSAGE);
                        return;
                    }
                        }
                    }
            if( dependencyOut != null ){
                        dependencyOut.writeObject( this.depenHashtable );
                    saveTab[3] = true;
                        }
                        dependencyOut.flush();
                        dependencyOut.close();
                        depFileOut.close();
                    }
        }
        catch( FileNotFoundException fe ){
            debug("FileNotFoundException_DepFileOut: "+fe);
        }
        catch( IOException e ){
            debug("IOException: "+e);
        }

                this.setEnabled( 3, false);
                this.configManagTabbedPane.setSelectedIndex( 0 );
                this.setEnabled( 0, true );
                this.setEnabled( 1, true );
                this.setEnabled( 2, true );
        this.depenPrimaryTextField.setText("");
        this.depenOutputTextField.setText("");
        this.depenSecondaryTextField.setText("");
            }

    /**
     * Before leave the dependency panel, save all the depency objects in
depenHashtable
     */
            public void
depenCancelButton_actionPerformed(java.awt.event.ActionEvent event)
            {
        try{
            FileOutputStream depFileOut = new FileOutputStream(
this.pathName+"\\dependency.cfg" );
            ObjectOutputStream dependencyOut = new ObjectOutputStream(
depFileOut );
```

```
            if( this.depenHashtable.size() > 0 ){
        if( dependencyOut != null ){
                dependencyOut.writeObject( this.depenHashtable );
            }
            dependencyOut.flush();
            dependencyOut.close();
            depFileOut.close();
            }
    }
    catch(IOException e ){
      debug("IOException_DepCancel: "+e);
    }

          this.setEnabled( 3, false);

          this.configManagTabbedPane.setSelectedIndex( 0 );

          this.setEnabled( 0, true );
          this.setEnabled( 1, true );
          this.setEnabled( 2, true );

    this.depenPrimaryTextField.setText("");
    this.depenOutputTextField.setText("");
    this.depenSecondaryTextField.setText("");
        }

  /**
    * Check all the changes had saved or not before exit
ProjectSchemaFrame
    */
        public void
doneButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    for( int i=0; i<saveTab.length; i++ ){
      if( !saveTab[i] ){
            Object[] options = { "Yes", "No" };
```

```
            int j = JOptionPane.showOptionDialog(this, "Would you like to
save?",
            "Warning", JOptionPane.DEFAULT_OPTION,
JOptionPane.WARNING_MESSAGE, null, options, options[0]);
            if( j==0 ){
              stepSaveButton_actionPerformed(event);
              compSaveButton_actionPerformed(event);
              EHLDoneButton_actionPerformed(event);
              depenOKButton_actionPerformed(event);

            }
        }
    }
          setVisible( false );
          dispose();
        }

  /**
    * Set enable a tab panel
    *
    * @param tabIndex : an index of panel
    * @param flag : true or false
    */
        public void setEnabled( int tabIndex, boolean flag ){
          this.configManagTabbedPane.setEnabledAt( tabIndex, flag );
        }

  /**
    * Short cut to print the output
    * @param string : the output string
    */
  public void debug( String string ){
    System.out.println(string);
  }

  class SymItem implements java.awt.event.ItemListener
        {
```

```
public void itemStateChanged(java.awt.event.ItemEvent
event)
{
Object object = event.getSource();
if (object == depenStepComboBox)

depenStepComboBox_itemStateChanged(event);
else if (object == depenEHLComboBox)

depenEHLComboBox_itemStateChanged(event);
else if (object == existedStepComboBox)

existedStepComboBox_itemStateChanged(event);
else if (object == existedCompComboBox)

existedCompComboBox_itemStateChanged(event);
else if (object == existedEHLComboBox)

existedEHLComboBox_itemStateChanged(event);
}
}

/**
 * Refresh stepHashtable and existedStepComboBox, and add new items
 *
 * @param stepVector : contains the current step type objects
 */
public void setStepComboBox( Vector stepVector ){
  this.existedStepComboBox.removeAllItems();
  this.existedStepComboBox.addItem(STEP_TYPE_TITLE);

  this.stepHashtable = new Hashtable();

  for( int i=0; i< stepVector.size(); i++ ){
    StepType st = (StepType)stepVector.elementAt( i );
    String stID = st.getStepID();

    //set step Hashtable
    this.stepHashtable.put( stID, st );

    this.existedStepComboBox.addItem(stID);
  }
}

/**
 * Refresh compHashtable and  existedCompComboBox, and add new
items
 *
 * @param compVector : contains the current component type objects
 */
public void setCompComboBox( Vector compVector ){
  this.existedCompComboBox.removeAllItems();
  this.existedCompComboBox.addItem(COMPONENT_TYPE_TITLE);

  this.compHashtable = new Hashtable();

  for( int i=0; i< compVector.size(); i++ ){
    ComponentType ct = (ComponentType)compVector.elementAt( i );
    String ctID = ct.getComponentID();

    //set step Hashtable
    this.compHashtable.put( ctID, ct );

    this.existedCompComboBox.addItem(ctID);
  }
}

/**
 * Refresh EHLHashtable and existedEHLComboBox, and add new
items
 *
 * @param EHLVector : contains the current EHL objects
 */
public void setEHLComboBox( Vector EHLVector ){
```

```
    this.existedEHLComboBox.removeAllItems();
    this.existedEHLComboBox.addItem(PROCESS_TITLE);

    this.EHLHashtable = new Hashtable();

    for( int i=0; i< EHLVector.size(); i++ ){
        EHL ehl = (EHL)EHLVector.elementAt( i );
        String EHLName = ehl.getEHLName();

        //set step Hashtable
        this.EHLHashtable.put( EHLName, ehl );

        this.existedEHLComboBox.addItem(EHLName);
    }
}

/**
 * Refresh depenEHLComboBox and add new items
 *
 * @param EHLVector : contains the current EHL objects
 */
public void setDepenEHLComboBox( Vector EHLVector ){
    this.depenEHLComboBox.removeAllItems();

    for( int j=0; j< EHLVector.size(); j++ ){
        EHL ehl = (EHL)EHLVector.elementAt(j);
        this.depenEHLComboBox.addItem(ehl.getEHLName());
    }
}

/**
 * Refresh depenStepComboBox, add new items,
 * and create new directory for the selected step
 *
 * @param stepVector : contains the current step names
 */
public void setDependStepComboBox( Vector stepVector ){
    String loopName =
(String)this.depenEHLComboBox.getSelectedItem();

    this.depenStepComboBox.removeAllItems();
    this.depenStepComboBox.addItem(STEP_TYPE_TITLE);
    for( int i=0; i< stepVector.size(); i++ ){
        String depStep = ((String)stepVector.elementAt( i )).trim();
        this.depenStepComboBox.addItem(depStep);
        File directory = new File( this.pathName, depStep );
        if( !directory.isDirectory() ){
            directory.mkdir();
        }
    }
}

/**
 * View the selected step type object
 *
 * @param st : selected step type object
 */
public void setStepInfo( StepType st ){
    stepIDTextField.setText( st.getStepID() );
    stepNameTextField.setText( st.getStepName() );
    stepDescriptionTextArea.setText( st.getStepDescription() );
}

/**
 * View the selected component type object
 *
 * @param ct : selected component type object
 */
public void setCompInfo( ComponentType ct ){
    compIDTextField.setText( ct.getComponentID() );
    compNameTextField.setText( ct.getComponentName() );
    compDescriptionTextArea.setText( ct.getComponentDescription() );
}
```

```
/**
 * View the selected EHL object
 *
 * @param ehl : selected EHL object
 */
public void setEHLInfo( EHL ehl ){
   EHLNameTextField.setText( ehl.getEHLName() );
   EHLPathTextField.setText( ehl.getEHLPath() );
}

/**
 * View the selected dependency object
 *
 * @param selectedDepStep : selected dependency object name
 */
public void setDepInfo( String selectedDepStep ){
   if( !selectedDepStep.equals(STEP_TYPE_TITLE) ){
      StringTokenizer st = new StringTokenizer( selectedDepStep,"-");
            String before_ = st.nextToken("-");
            String _after = st.nextToken("-");
      this.depenPrimaryTextField.setText( _after );
      this.depenOutputTextField.setText( _after);

            if( this.depenHashtable.containsKey( selectedDepStep ) ){
         Dependency dep = (Dependency)this.depenHashtable.get(
selectedDepStep );
         String secondInput = dep.getSecondaryInput();
         if( !secondInput.equals("") ){
            this.depenSecondaryTextField.setText(secondInput);
                }
      }
   }
}

/**
 * Read step.cfg, component.cfg, and loop.cfg files and insert their
contents
 * into stepVector, compVector, and EHLVector respectively
 *
 * @param pathName : the path of current project
 */
public void readInputFiles( String pathName ){
   File stepFile = new File(this.pathName, "step.cfg");
   if( stepFile.exists() ){
      try{
      FileInputStream fileInput = new FileInputStream( stepFile );
      ObjectInputStream stepIn = new ObjectInputStream( fileInput );
         if( stepIn != null ){
            this.stepVector = (Vector)stepIn.readObject();
            if( this.stepVector.size() > 0 ){
               this.setStepComboBox( this.stepVector );
               this.setListModel(this.stepVector);
            }
         }
         stepIn.close();
         fileInput.close();
      }
      catch( IOException e ){
         debug("IOException_StepInitial: "+e);
         if( this.stepVector.size() > 0 ){
            this.setStepComboBox( this.stepVector );
            this.setListModel(this.stepVector);
         }
      }
      catch( ClassNotFoundException ex ){
         debug("ClassNotFoundException_SteppInitial: "+ex);
         if( this.stepVector.size() > 0 ){
            this.setStepComboBox( this.stepVector );
            this.setListModel(this.stepVector);
         }
      }
   }

   File componentFile = new File(this.pathName, "component.cfg");
```

```
if( componentFile.exists() ){
  try{
    FileInputStream fileInput = new FileInputStream( componentFile)
;
    ObjectInputStream compIn = new ObjectInputStream( fileInput );
    if( compIn != null ){
      this.compVector = (Vector)compIn.readObject();
      if( this.compVector.size() > 0 ){
        this.setCompComboBox( this.compVector );
      }
    }
    compIn.close();
    fileInput.close();
  }
  catch( IOException e ){
    debug("IOException_CompInitial: "+e);
    if( this.compVector.size() > 0 ){
      this.setCompComboBox( this.compVector );
    }
  }
  catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_CompInitial: "+ex);
    if( this.compVector.size() > 0 ){
      this.setCompComboBox( this.compVector );
    }
  }
}

File loopFile = new File(this.pathName, "loop.cfg");
if( loopFile.exists() ){
  try{
    FileInputStream fileInput = new FileInputStream( loopFile );
    ObjectInputStream EHLIn = new ObjectInputStream( fileInput );
    if( EHLIn != null ){
      this.EHLVector = (Vector)EHLIn.readObject();
      if( this.EHLVector.size() > 0 ){
        this.setEHLComboBox( this.EHLVector );
      }
    }
    EHLIn.close();
    fileInput.close();
  }
  catch( IOException e ){
    debug("IOException_EHLInInitial: "+e);
    if( this.EHLVector.size() > 0 ){
      this.setEHLComboBox( this.EHLVector );
    }
  }
  catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_EHLInInitial: "+ex);
    if( this.EHLVector.size() > 0 ){
      this.setEHLComboBox( this.EHLVector );
    }
  }
 }
}

/**
 * Read dependency.cfg file and insert its content into depenHashtable
 */
public void readDepFile(){

  File dependencyFile = new File(this.pathName, "dependency.cfg");
  if( dependencyFile.exists() ){
    try{
      FileInputStream fileInput = new FileInputStream( dependencyFile
);
      ObjectInputStream dependencyIn = new ObjectInputStream(
fileInput );
      if( dependencyIn != null ){
        this.depenHashtable = (Hashtable)dependencyIn.readObject();
      }
      dependencyIn.close();
      fileInput.close();
```

```
        }
        catch( IOException e ){
           debug("IOException_readDepIn: "+e);
        }
        catch( ClassNotFoundException ex ){
           debug("ClassNotFoundException_readDepIn: "+ex);
        }
      }
    }
    /**
     * List all existing step type objects in the project
     * and allow a user to view or edit them
     */
          void
existedStepComboBox_itemStateChanged(java.awt.event.ItemEvent event)
          {
      if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedStepID = (String)event.getItem();
        this.selectedStepIndex =
this.existedStepComboBox.getSelectedIndex();
        if( this.selectedStepIndex == 0 ){
           //Clean up the frame
                this.stepClearButton_actionPerformed( null );
              }
        else if( this.selectedStepIndex > 0 ){
           if( this.stepHashtable.containsKey( selectedStepID ) ){
              this.setStepInfo( (StepType)this.stepHashtable.get(
selectedStepID ) );
           }
        }
      }
          }

    /**
     * List all existing component type objects in the project
     * and allow a user to view or edit them
     */
```

```
          void
existedCompComboBox_itemStateChanged(java.awt.event.ItemEvent
event)
          {
      if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedCompID = (String)event.getItem();
        this.selectedCompIndex =
this.existedCompComboBox.getSelectedIndex();
        if( this.selectedCompIndex == 0 ){
           //Clean up the frame
                this.compClearButton_actionPerformed( null );
              }
        if( this.compHashtable.containsKey( selectedCompID ) ){
           this.setCompInfo( (ComponentType)this.compHashtable.get(
selectedCompID ) );
        }
      }
    }

    /**
     * List all existing EHL objects in the project
     * and allow a user to view or edit them
     */
          void
existedEHLComboBox_itemStateChanged(java.awt.event.ItemEvent event)
          {
      if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedEHLName = (String)event.getItem();
        this.selectedEHLIndex =
this.existedEHLComboBox.getSelectedIndex();
        if( this.selectedEHLIndex == 0 ){
           //Clean up the frame
                this.EHLClearButton_actionPerformed( null );
              }
        if( this.EHLHashtable.containsKey( selectedEHLName ) ){
           this.setEHLInfo( (EHL)this.EHLHashtable.get(
selectedEHLName ) );
```

```
        }
      }
          }
  /**
   * List all existing step type objects in a selected project
   * and allow a user to view, edit, or set secondary input
   */
  void depenStepComboBox_itemStateChanged(java.awt.event.ItemEvent
event)
          {
      Dependency dep = null;
      String selectedDepStep = null;
      if( event != null ){
        selectedDepStep = ((String)event.getItem()).trim();
      }
      this.testIndex++;
      String loopName =
(String)this.depenEHLComboBox.getSelectedItem();
      String stepName =
(String)this.depenStepComboBox.getItemAt(this.selectedDepenStepIndex);
      String output = (String)this.depenOutputTextField.getText();
      String primary = (String)this.depenPrimaryTextField.getText();
      String secondary = (String)this.depenSecondaryTextField.getText();

      if( this.testIndex < 2 ){
        this.selectedDepenStepIndex =
this.depenStepComboBox.getSelectedIndex();
        if( (secondary != null) && (this.selectedDepenStepIndex > 0) ){
          if( this.depenHashtable.containsKey( stepName ) ){
            dep = (Dependency) this.depenHashtable.get( stepName );
            if( !secondary.equals("") ){
              dep.setSecondaryInput( secondary );
            }
          }
          else{
            dep = new Dependency( loopName, stepName, output,
primary, secondary);
            this.depenHashtable.put( stepName, dep );
          }
          this.depenSecondaryTextField.setText("");
        }
      }
      else {
        if( (stepName != null) &&
(this.depenHashtable.containsKey(stepName )) ){
          dep = (Dependency) this.depenHashtable.get( stepName );
          if( !secondary.equals("") ){
            dep.setSecondaryInput( secondary );
          }
        }
        this.testIndex = 0;
      }
      if( selectedDepStep.equals(STEP_TYPE_TITLE) ){
        this.depenOutputTextField.setText("");
        this.depenPrimaryTextField.setText("");
        this.depenSecondaryTextField.setText("");
      }
      else{
        this.setDepInfo( selectedDepStep );
      }
          }
  /**
   * List all existing EHL name in the project
   * and allow a user to see its all step type objects
   */
        void
depenEHLComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
          if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedItem = (String)event.getItem();
        int selectedIndex = this.depenEHLComboBox.getSelectedIndex();

        if( this.EHLHashtable.containsKey( selectedItem ) ){
          EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
```

```
            StringTokenizer st = new StringTokenizer(
(String)ehl.getEHLPath(), "," );
                    Vector tokenizeVector = new Vector();
                    while( st.hasMoreTokens() ){
                       tokenizeVector.addElement( st.nextToken() );
                    }
            this.setDependStepComboBox( tokenizeVector );
        }
    }
        }
  /**
   * Set step type object list in EHL panel
   *
   * @param stepVector : all existing step type objects
   */
        public void setListModel( Vector stepVector ){
           this.listModel.removeAllElements();
           for( int i=0; i< stepVector.size(); i++ ){
              StepType st = (StepType) stepVector.elementAt( i );
              this.listModel.addElement( st.getStepID() );
           }
        }

        class SymMouse extends java.awt.event.MouseAdapter
        {
                public void mouseClicked(java.awt.event.MouseEvent
event)
                {
                        Object object = event.getSource();
                        if (object == stepTypesList)
                                stepTypesList_mouseClicked(event);
                }
}
  /**
   * Monitor mouse click action on stepTypesList
   */
        void stepTypesList_mouseClicked(java.awt.event.MouseEvent
event)
        {
           String s =
((String)this.stepTypesList.getSelectedValue()).trim();
           String EHLStep = (String) this.EHLPathTextField.getText();
           Vector v = new Vector();
           if( !EHLStep.equals("") ){
        StringTokenizer st = new StringTokenizer(EHLStep, ",");
        while( st.hasMoreTokens() ){
           String s1 = ((String) st.nextToken()).trim();
           v.addElement(s1);
        }
        if( !v.contains(s) ){
                this.EHLPathTextField.setText(EHLStep+", "+s);
                v.addElement(s);
        }
              else{
           JOptionPane.showMessageDialog(this, s+" is already in this
process!",
                        "Error", JOptionPane.ERROR_MESSAGE);
              }
           }
           else{
              this.EHLPathTextField.setText(s);
           }
}
  /**
   * Save the last selected dependency object with
   * its content before quit the dependency panel
   */
        void saveLastDep(){
     Dependency dep = null;
     String loopName =
(String)this.depenEHLComboBox.getSelectedItem();
     String stepName =
(String)this.depenStepComboBox.getItemAt(this.selectedDepenStepIndex);
```

```
        String output = (String)this.depenOutputTextField.getText();
        String primary = (String)this.depenPrimaryTextField.getText();
        String secondary = (String)this.depenSecondaryTextField.getText();

        this.selectedDepenStepIndex =
this.depenStepComboBox.getSelectedIndex();
        if( (secondary != null) && (this.selectedDepenStepIndex > 0) ){
          if( this.depenHashtable.containsKey( stepName ) ){
            dep = (Dependency) this.depenHashtable.get( stepName );
            if( !secondary.equals("") ){
              dep.setSecondaryInput( secondary );
            }
          }
          else{
            dep = new Dependency( loopName, stepName, output, primary,
secondary);
            this.depenHashtable.put( stepName, dep );
          }
        }
}
  /**
   * Launch ListDialog where lists all existing component type names
   */
          void
secondaryButton_actionPerformed(java.awt.event.ActionEvent event)
          {
        (new ListDialog(this, "Component Types List",
this.compVector)).setVisible(true);
}
  /**
   * Set all selected component type names from ListDialog
   * into secondary input text field in dependency panel
   *
   * @param objs : current selected items from ListDialog
   */
          public void setItemList(Object[] objs){
            if( objs.length >= 1 ){
              String s = (String)objs[0];
              for( int i=1; i<objs.length; i++ ){
                s = s + ", "+ objs[i];
              }
              this.depenSecondaryTextField.setText(s);
            }
          }
}
```

```
package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * ReviewComponentContentDialog : View all links in a component
content of selectted step
 *
 * Implement CasesTitle where stores all global variables of Cases package
 */
```

```
//////////////////////////////////////////////////////////////////////////////
public class ReviewComponentContentDialog extends
com.sun.java.swing.JDialog implements CasesTitle
{
  /**
   * listModel : list all links of a selected link type
   */
  DefaultListModel listModel = new DefaultListModel();

  /**
   * storedVector : contains links of all available link types in the
   * component content of selected step
   */
  public Vector[] storedVector = {new Vector(), new Vector(), new
Vector(), new Vector(), new Vector(), new Vector()};

        /**
   * Build ReviewComponentContentDialog
   */
        public ReviewComponentContentDialog(Frame parent)
        {
                super(parent);

                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Review Component Content");
                setModal(true);
                getContentPane().setLayout(null);
                setSize(500,450);
                setVisible(false);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                JLabel1.setText("Available Links");
                getContentPane().add(JLabel1);
                JLabel1.setForeground(java.awt.Color.black);
                JLabel1.setBounds(15,162,470,24);
                getContentPane().add(connectionComboBox);
                connectionComboBox.setBounds(115,60,270,24);
                itemScrollPane.setOpaque(true);
                getContentPane().add(itemScrollPane);
                itemScrollPane.setBounds(15,190,470,200);
                itemScrollPane.getViewport().add(itemList);
                itemList.setBounds(0,0,467,197);
                exitButton.setText("Exit");
                exitButton.setActionCommand("Cancel");
                getContentPane().add(exitButton);
                exitButton.setBounds(212,400,75,24);
                connectButton.setText("Connect");
                connectButton.setActionCommand("jbutton");
                getContentPane().add(connectButton);
                connectButton.setBounds(400,120,85,24);

        titleLabel.setHorizontalAlignment(com.sun.java.swing.SwingCons
tants.CENTER);
                titleLabel.setText("jlabel");
                getContentPane().add(titleLabel);
                titleLabel.setForeground(java.awt.Color.black);
                titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
                titleLabel.setBounds(25,10,450,30);
                selectedTextField.setEditable(false);
                getContentPane().add(selectedTextField);
                selectedTextField.setBackground(java.awt.Color.white);
                selectedTextField.setBounds(16,120,384,24);
                //}}
                        //{{INIT_MENUS
```

```
//}}

//{{REGISTER_LISTENERS
SymItem lSymItem = new SymItem();
connectionComboBox.addItemListener(lSymItem);
SymAction lSymAction = new SymAction();
connectButton.addActionListener(lSymAction);
exitButton.addActionListener(lSymAction);
SymMouse aSymMouse = new SymMouse();
itemList.addMouseListener(aSymMouse);
//}}

itemList.setModel(listModel);
connectionComboBox.addItem(LINKS_TITLE);
for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
connectionComboBox.addItem(LINK_FILE_NAMES[i]);
}
}

public ReviewComponentContentDialog()
{
this((Frame)null);
}

/**
* After select a step and press OK button from ListDialog, it will be launched
*/
public ReviewComponentContentDialog(String title, Vector[] storedVector){
this();
this.titleLabel.setText(title);
this.storedVector = storedVector;
}

public ReviewComponentContentDialog(String sTitle)
{
this();
setTitle(sTitle);
}

public void setVisible(boolean b)
{
if (b)
setLocation(50, 50);
super.setVisible(b);
}

static public void main(String args[])
{
(new ReviewComponentContentDialog()).setVisible(true);
}

public void addNotify()
{
// Record the size of the window prior to calling parents addNotify.
Dimension size = getSize();

super.addNotify();

if (frameSizeAdjusted)
return;
frameSizeAdjusted = true;

// Adjust size of frame according to the insets
Insets insets = getInsets();
setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
}

// Used by addNotify
boolean frameSizeAdjusted = false;
```

```
        //{{DECLARE_CONTROLS
        com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox connectionComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JScrollPane itemScrollPane = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JList itemList = new
com.sun.java.swing.JList();
        com.sun.java.swing.JButton exitButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton connectButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel titleLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField selectedTextField = new
com.sun.java.swing.JTextField();
        //}}

        class SymItem implements java.awt.event.ItemListener
        {
                public void itemStateChanged(java.awt.event.ItemEvent
event)
                {
                        Object object = event.getSource();
                        if (object == connectionComboBox)

        connectionComboBox_itemStateChanged(event);
                }
        }
  /**
   * Select a link type
   */
        void
connectionComboBox_itemStateChanged(java.awt.event.ItemEvent event)
        {
            if( event.getStateChange() == event.SELECTED ){
            selectedTextField.setText("");
            listModel.removeAllElements();
              String selectedItem = (String)event.getItem();
              for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
          if( selectedItem.equals(LINK_FILE_NAMES[i]) ){
            for( int j=0; j<storedVector[i].size(); j++ ){
              listModel.addElement(storedVector[i].elementAt(j));
            }
            i= LINK_FILE_NAMES.length;
          }
        }
        }
        }

        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == connectButton)
                                connectButton_actionPerformed(event);
                        else if (object == exitButton)
                                exitButton_actionPerformed(event);
                }
        }
  /**
   * Connect to an application which matchs with selected item from
connectionComboBox
   * and view the file which the name is the content of selectedTextField
link
   */
        void connectButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
        String s = " " + selectedTextField.getText();
```

```
        if( !s.equals("") ){
          if( connectionComboBox.getSelectedIndex()>0 ){
            int index = connectionComboBox.getSelectedIndex()-1;
            if( index == 3 ){
                  (new PersonnelFrame(s.trim(), "View")).setVisible(true);
            }
            else{
              try {
                Runtime runtime = Runtime.getRuntime();
                runtime.exec(EXECUTIONS[index]+s);
              }
              catch( Exception e ) {
                throw new RuntimeException(e.toString());
              }
            }
          }
        }
          }
    /**
     * Exit ReviewComponentContentDialog
     */
          void exitButton_actionPerformed(java.awt.event.ActionEvent
event)
          {
        setVisible( false );
        dispose();
          }

          class SymMouse extends java.awt.event.MouseAdapter
          {
                    public void mouseClicked(java.awt.event.MouseEvent
event)
                    {
                              Object object = event.getSource();
                              if (object == itemList)
                                        itemList_mouseClicked(event);
                    }
          }
    /**
     * Set the selectedTextField when a user selects an item from this list
     */
          void itemList_mouseClicked(java.awt.event.MouseEvent event)
          {
            if( event.getClickCount() > 0) {
              String s = (String)this.itemList.getSelectedValue();
              this.selectedTextField.setText(s);
            }
    }
}
```

```
package Cases;

import java.awt.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import java.awt.event.*;
import java.util.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * SkillTableFrame : a table of skill IDs, skill names, and skill levels
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Cases
 */
//////////////////////////////////////////////////////////////////////////////
```

```
public class SkillTableFrame extends com.sun.java.swing.JFrame
implements CasesTitle
{
  /**
   * scf : parent frame to launch this frame
   */
  StepContentFrame scf = null;

  /**
   * pf : parent frame to launch this frame
   */
  PersonnelFrame pf = null;

  /**
   * skillVector : selected skill to return to parent frame
   */
  private Vector skillVector = new Vector();

  /**
   * skillModel : monitor skill table
   */
  protected DefaultTableModel skillModel = null;

  /**
   * Build SkillTableFrame
   */
        public SkillTableFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Skill List");
                getContentPane().setLayout(null);
                setSize(405,305);
                setVisible(false);
                tableScrollPane.setOpaque(true);
                getContentPane().add(tableScrollPane);
                tableScrollPane.setBounds(33,54,324,169);
                tableScrollPane.getViewport().add(skillTable);
                skillTable.setBounds(0,0,321,166);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                JLabel1.setText("Skill List");
                getContentPane().add(JLabel1);
                JLabel1.setForeground(java.awt.Color.black);
                JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
                JLabel1.setBounds(15,6,375,47);
                OKButton.setText("OK");
                OKButton.setActionCommand("OK");
                getContentPane().add(OKButton);
                OKButton.setBounds(121,252,73,24);
                cancelButton.setText("Cancel");
                cancelButton.setActionCommand("Cancel");
                getContentPane().add(cancelButton);
                cancelButton.setBounds(211,252,73,24);
                //}}

                //{{INIT_MENUS
                //}}

                //{{REGISTER_LISTENERS
                SymAction lSymAction = new SymAction();
                OKButton.addActionListener(lSymAction);
                cancelButton.addActionListener(lSymAction);
                //}}
```

```
  /**
   * create the new skill table
   */
                createSkillTable();
        }

/**
 * StepContentFrame launch this frame when skill button is slected
 */
public SkillTableFrame(StepContentFrame scf){
   this();
   this.scf = scf;
}
/**
 * PersonnelFrame launch this frame  when skill button is slected
 */

public SkillTableFrame(PersonnelFrame pf){
   this();
   this.pf = pf;
}

        public SkillTableFrame(String sTitle)
        {
                this();
                setTitle(sTitle);
        }

        public void setVisible(boolean b)
        {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
        }

        static public void main(String args[])
        {
                (new SkillTableFrame()).setVisible(true);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets and menu
bar
                Insets insets = getInsets();
                com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
                int menuBarHeight = 0;
                if (menuBar != null)
                        menuBarHeight =
menuBar.getPreferredSize().height;
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JScrollPane tableScrollPane = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JTable skillTable = new
com.sun.java.swing.JTable();
```

```
        com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JButton OKButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton cancelButton = new
com.sun.java.swing.JButton();
        //}}

        //{{DECLARE_MENUS
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == OKButton)
                                OKButton_actionPerformed(event);
                        else if (object == cancelButton)
                                cancelButton_actionPerformed(event);
                }
        }

  /**
   * Return all selected skills to a parent frame and exit SkillTableFrame
   */
        void OKButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
          int[] selectedRows = skillTable.getSelectedRows();
          for( int i=0; i<selectedRows.length; i++ ){
            int index = selectedRows[i];
       String s = skillModel.getValueAt(index,0)+" :
"+skillModel.getValueAt(index,1)+" : "+skillModel.getValueAt(index,2);
            skillVector.addElement(s);
          }
          if( this.scf != null ){
            this.scf.setSkillComboBox(skillVector);
          }
          else if( this.pf != null ){
            this.pf.setSkillComboBox(skillVector);
          }
          setVisible(false);
          dispose();
        }

  /**
   * Exit this frame without return anything
   */
        void cancelButton_actionPerformed(java.awt.event.ActionEvent
event)
        {
          setVisible(false);
          dispose();
        }

  /**
   * Create a skill table with 3 columns and 21 rows
   */
        void createSkillTable(){
                String[] skillColumnNames = {"Skill ID", "Skill Name",
"Skill Level" };
                final Object[][] skillData = new Object[20][3];

                for( int i = 1; i <SKILL_ID; ++i ){
                        skillData[i-1][0] = ""+i;
                }

                for( int j=0; j<SKILL_LIST.length; j++ ){
                  skillData[j][1] = SKILL_LIST[j];
                  skillData[j][2]=""+0;
                }
```

```
		skillModel = new DefaultTableModel(skillData,
skillColumnNames);
			JComboBox skillLevelComboBox = new JComboBox();
			for( int k= 0; k <SKILL_LEVEL; ++k ){
				skillLevelComboBox.addItem(""+k);
			}
		skillTable.setModel(skillModel);

			TableColumn skillLevelColumn =
skillTable.getColumn(skillColumnNames[2]);

			// Use the combo box as the editor in the "Skill Level"
column.
			skillLevelColumn.setCellEditor(new
DefaultCellEditor(skillLevelComboBox));
		}
}
```



```
package Cases;

/**
 * Step Content Object which is used to save in step.cnt file
 */

import java.io.Serializable;
import java.util.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * StepContent : Create a step content object and save it in step.cnt file
 */
//////////////////////////////////////////////////////////////////////////////
public class StepContent implements Serializable{

/**
 * stepName : step version
 */
private String stepName = null;

/**
 * status : status of the step, eg. approved, scheduled,complete,...
 */
private String status = null;

/**
 * skill : a vector of skills
 */
private Vector skill = new Vector();

/**
 * securityLevel : security level of the step
 */
private int securityLevel = 0;

/**
 * organizer : a person's ID to organize the step
 */
private String organizer = null;

/**
 * predecessors : a vector of atomics
 */
private Vector predecessors = new Vector();

/**
 * priority : priority of the step
 */
private int priority = 0;

/**
```

```
 * estimateDuration : estimate how long the job will finish
 */
private int estimatedDuration = 0;

/**
 * deadline : deadline of the job
 */
private String deadline = null;

/**
 * earliestStartTime : the time to start in the plan
 */
private String earliestStartTime = null;

/**
 * finishTime : when the job is finished
 */
private String finishTime = null;

/**
 * realStartTime : the actual day to start the job
 */
private String realStartTime = null;

/**
 * manager : a person's ID to manage the job
 */
private String manager = null;

/**
 * evaluation : description of an evaluation
 */
private String evaluation = null;

/**
 * evaluator : a person's ID to evaluate the job
 */
private String evaluator = null;

/**
 * This StepContent constructor is used to create a step content object
 */
public StepContent( String stepName, String status, Vector skill,
            int securityLevel, String evaluation, String evaluator,
            String organizer,Vector predecessors, int priority,
            int estimatedDuration, String deadline, String startTime,
            String finishTime,  String manager){
  this.stepName = stepName;
  this.status = status;
  this.skill = skill;
  this.securityLevel = securityLevel;
  this.evaluation = evaluation;
  this.evaluator = evaluator;
  this.organizer = organizer;
  this.predecessors = predecessors;
  this.priority = priority;
  this.estimatedDuration = estimatedDuration;
  this.deadline = deadline;
  this.earliestStartTime = startTime;
  this.finishTime = finishTime;
  this.manager = manager;
}

/**
 * Empty StepContent constructor
 */
public StepContent(){
}

/**
 * Set a name for the step
 *
 * @param s : name of the step
 */
```

```
public void setStepName(String s){
   this.stepName = s;
}

/**
 * Get the step's name
 *
 * @return stepName : the step's name
 */
public String getStepName(){
   return this.stepName;
}

/**
 * Set status for the step
 *
 * @param s : status of the step
 */
public void setStatus(String s){
   this.status = s;
}

/**
 * Status of the step
 *
 * @return status : status of the step
 */
public String getStatus(){
   return this.status;
}

/**
 * Set skills for the step
 *
 * @param v : vector of skills
 */
public void setSkill(Vector v){
   this.skill = v;
}

/**
 * Skills of the step
 *
 * @return skill : vector of skills of the step
 */
public Vector getSkill(){
   return skill;
}

/**
 * Set security level for the step
 *
 * @param i: security level
 */
public void setSecurityLevel(int i){
   this.securityLevel = i;
}

/**
 * Security level of the step
 *
 * @return securityLevel : security level of the step
 */
public int getSecurityLevel(){
   return this.securityLevel;
}

/**
 * Set evaluation of the step
 *
 * @param s : a string of evaluation
 */
public void setEvaluation(String s){
   this.evaluation = s;
```

```
}

/**
 * Evaluation of the step
 *
 * @return evaluation : evaluation of the step
 */
public String getEvaluation(){
   return this.evaluation;
}

/**
 * Set evaluator for the step
 *
 * @param s : evaluator's name
 */
public void setEvaluator(String s){
   this.evaluator = s;
}

/**
 * Evaluator of the step
 *
 * @return evaluator : evaluator of the step
 */
public String getEvaluator(){
   return this.evaluator;
}

/**
 * Set organizer for the step
 *
 * @param s : the name of organizer
 */
public void setOrganizer(String s){
   this.organizer = s;
}

/**
 * Organizer of the step
 *
 * @return organizer : organizer of the step
 */
public String getOrganizer(){
   return this.organizer;
}

/**
 * Set predecessors for the step
 *
 * @param v : vector of atomics
 */
public void setPredecessors(Vector v){
   this.predecessors = v;
}

/**
 * Predecessors of the step
 *
 * @return predecessors : a vector of atomics
 */
public Vector getPredecessors(){
   return this.predecessors;
}

/**
 * Set priority for the step
 *
 * @param i : level of priority
 */
public void setPriority(int i){
   this.priority = i;
}
```

```
/**
 * Priority of the step
 *
 * @return priority : an integer of priority level
 */
public int getPriority(){
   return this.priority;
}

/**
 * Set duration for the step
 *
 * @param i : prediction of how long it will take to finish the job
 */
public void setDuration(int i){
   this.estimatedDuration = i;
}

/**
 * Estimate duration to finish the job
 *
 * @return estimatedDuration : time to finish the job
 */
public int getDuration(){
   return this.estimatedDuration;
}

/**
 * Set deadline of the job
 *
 * @param d : exactly day to be done this job
 */
public void setDeadline(String d){
   this.deadline = d;
}

/**
 * Deadline of the job
 *
 * @return deadline : a string to express the deadline
 */
public String getDeadline(){
   return this.deadline;
}

/**
 * Set earliest start time for the job
 *
 * @param d : a string to express the earliest start time
 */
public void setStartTime(String d){
   this.earliestStartTime = d;
}

/**
 * The day to plan starting the job
 *
 * @return earliestStartTime :  plan to start on this day
 */
public String getStartTime(){
   return this.earliestStartTime;
}

/**
 * Set the finish time for the job
 *
 * @param d : a string to express the finish day
 */
public void setFinishTime(String d){
   this.finishTime = d;
}

/**
 * The finish day
```

```
 *
 * @return finishTime : a string to express the finish day
 */
public String getFinishTime(){
   return this.finishTime;
}

/**
 * Set the exactly day to start the job
 *
 * @param realStartTime : a string to express the day to start the job
 */
public void setRealStartTime(String d){
   this.realStartTime = d;
}

/**
 * The day to start the job
 *
 * @return realStartTime : a string to express the day to start the job
 */
public String getRealStartTime(){
   return this.realStartTime;
}

/**
 * Set manager's ID for the step
 *
 * @param s : the manager's ID
 */
public void setManager(String s){
   this.manager = s;
}

/**
 * The manager's ID to manage the job
 *
 * @return manager : the manager's ID
 */
public String getManager(){
   return this.manager;
}
}
```

```
package Cases;

import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;
import com.symantec.itools.awt.MaskedTextField;

////////////////////////////////////////////////////////////////////////////////
/**
 * StepContentFrame : Use to create/delete/view step content of the
selected step
 *
```

```
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_StepContent
 */
//////////////////////////////////////////////////////////////////////////////
public class StepContentFrame extends com.sun.java.swing.JFrame
implements CasesTitle, I_StepContent
{
  /**
   * atomicsVector : contains all atomics of the selected step
   */
  public Vector atomicsVector = new Vector();

  /**
   * selectedSkill : contains all selected skills from SkillTableFrame
   */
  public Vector selectedSkill = null;

  /**
   * selectedPred : contains all selected predecessors from ListDialog
   */
  public Vector selectedPred = null;

  /**
   * pathName : the complete path of current step
   */
  public String pathName = null;

  /**
   * Build StepContentFrame
   */
        public StepContentFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("SPIDER-Step Content");
                getContentPane().setLayout(null);
                setSize(700,550);
                setVisible(false);
                getContentPane().add(saveButton);
                saveButton.setBounds(0,0,0,0);
                getContentPane().add(deleteButton);
                deleteButton.setBounds(0,0,0,0);
                exitButton.setText("Exit");
                exitButton.setActionCommand("Save");
                getContentPane().add(exitButton);
                exitButton.setBounds(590,470,75,24);

        titleLable.setHorizontalAlignment(com.sun.java.swing.SwingCons
tants.RIGHT);
                titleLable.setText("Step Content:");
                getContentPane().add(titleLable);
                titleLable.setForeground(java.awt.Color.black);
                titleLable.setFont(new Font("Dialog", Font.BOLD, 18));
                titleLable.setBounds(82,30,130,30);  //y=6
                /**stepVersionTF**/
                stepVersionTextField.setEditable(false);
                getContentPane().add(stepVersionTextField);

        stepVersionTextField.setBackground(java.awt.Color.white);

        stepVersionTextField.setBounds(102,105,200,24);//246,55,300,24)
;
                /**/
                /**stepVersionLabel**/
```

```
JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel1.setText("Step Version");
        getContentPane().add(JLabel1);
        JLabel1.setForeground(java.awt.Color.black);
        JLabel1.setBounds(1,105,100,24);//154,55,90,24);
        /**/
        getContentPane().add(managerComboBox);
        managerComboBox.setBounds(472,405,200,24);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel2.setText("Manager");
        getContentPane().add(JLabel2);
        JLabel2.setForeground(java.awt.Color.black);
        JLabel2.setBounds(371,405,100,24);
        /**Status label**/

JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel3.setText("Status");
        getContentPane().add(JLabel3);
        JLabel3.setForeground(java.awt.Color.black);
        JLabel3.setBounds(1,155,100,24);//1,105,100,24);
        getContentPane().add(statusComboBox);

statusComboBox.setBounds(102,155,200,24);//102,105,200,24);

JLabel5.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel5.setText("Evaluation");
        getContentPane().add(JLabel5);
        JLabel5.setForeground(java.awt.Color.black);
        JLabel5.setBounds(1,305,100,24);

JLabel6.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel6.setText("Evaluator");
        getContentPane().add(JLabel6);
        JLabel6.setForeground(java.awt.Color.black);
        JLabel6.setBounds(1,355,100,24);
        getContentPane().add(evaluatorComboBox);
        evaluatorComboBox.setBounds(102,355,200,24);

JLabel7.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel7.setText("Security Level");
        getContentPane().add(JLabel7);
        JLabel7.setForeground(java.awt.Color.black);
        JLabel7.setBounds(1,255,100,24);
        getContentPane().add(securityLevelComboBox);
        securityLevelComboBox.setBounds(102,255,200,24);

JLabel9.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
        JLabel9.setText("Organizer");
        getContentPane().add(JLabel9);
        JLabel9.setForeground(java.awt.Color.black);
        JLabel9.setBounds(1,405,100,24);
        getContentPane().add(organizerComboBox);
        organizerComboBox.setBounds(101,405,200,24);
        getContentPane().add(predecessorsComboBox);
        predecessorsComboBox.setBounds(472,105,200,24);

JLabel12.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
        JLabel12.setText("Priority");
        getContentPane().add(JLabel12);
        JLabel12.setForeground(java.awt.Color.black);
        JLabel12.setBounds(371,155,100,24);
        getContentPane().add(priorityComboBox);
```

```
priorityComboBox.setBounds(472,155,200,24);
//**SkillComboBox
getContentPane().add(skillComboBox);

skillComboBox.setBounds(102,205,200,24);//102,155,200,24);
//**/

JLabel16.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.RIGHT);
JLabel16.setText("Estimated Duration");
getContentPane().add(JLabel16);
JLabel16.setForeground(java.awt.Color.black);
JLabel16.setBounds(361,205,110,24);
getContentPane().add(evaluationTextField);
evaluationTextField.setBounds(101,305,200,24);
deadlineTextField.setEditable(false);
getContentPane().add(deadlineTextField);
deadlineTextField.setBackground(java.awt.Color.white);
deadlineTextField.setBounds(472,255,200,24);
earliestSTTextField.setEditable(false);
getContentPane().add(earliestSTTextField);

earliestSTTextField.setBackground(java.awt.Color.white);
earliestSTTextField.setBounds(472,305,200,24);
finishTimeTextField.setEditable(false);
getContentPane().add(finishTimeTextField);

finishTimeTextField.setBackground(java.awt.Color.white);
finishTimeTextField.setBounds(472,355,200,24);
selectedLabel.setText("jlabel");
getContentPane().add(selectedLabel);
selectedLabel.setForeground(java.awt.Color.black);
selectedLabel.setFont(new Font("Dialog", Font.BOLD,
14));
selectedLabel.setBounds(217,30,400,30); //y=6
predecessorsButton.setText("Predecessors");
predecessorsButton.setActionCommand("jbutton");
getContentPane().add(predecessorsButton);
predecessorsButton.setBounds(351,105,120,24);
//**SkillButton
skillButton.setText("Skill");
skillButton.setActionCommand("Skill");
getContentPane().add(skillButton);
skillButton.setBounds(21,205,80,24);//21,155,80,24);
//**/
deadlineButton.setText("Deadline");
deadlineButton.setActionCommand("Deadline");
getContentPane().add(deadlineButton);
deadlineButton.setBounds(371,255,100,24);
startTimeButton.setText("Earliest Start Time");
startTimeButton.setActionCommand("Earliest Start
Time");

getContentPane().add(startTimeButton);
startTimeButton.setBounds(332,305,139,24);
finishTimeButton.setText("Finish Time");
finishTimeButton.setActionCommand("Finish Time");
getContentPane().add(finishTimeButton);
finishTimeButton.setBounds(371,355,100,24);
getContentPane().add(estDurationTextField);
estDurationTextField.setBounds(472,205,200,24);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
saveButton.addActionListener(lSymAction);
deleteButton.addActionListener(lSymAction);
exitButton.addActionListener(lSymAction);
predecessorsButton.addActionListener(lSymAction);
skillButton.addActionListener(lSymAction);
deadlineButton.addActionListener(lSymAction);
startTimeButton.addActionListener(lSymAction);
```

```
                finishTimeButton.addActionListener(lSymAction);
                //}}

        //Set status combobox
        statusComboBox.addItem("Proposed");
        statusComboBox.addItem("Approved");
        statusComboBox.addItem("Scheduled");
        statusComboBox.addItem("Assigned");
        statusComboBox.addItem("Decomposed");
        statusComboBox.addItem("Abandoned");
        statusComboBox.addItem("Completed");

        //Set security level from 0..5
        for( int i=0; i<SECURITY_LEVEL; i++){
           securityLevelComboBox.addItem(i+"");
        }

        //Set priority combobox from 0..5
        for( int j=0; j<PRIORITY_LEVEL; j++){
           priorityComboBox.addItem(j+"");
        }
            }

    /**
     * CasesFrame launch this frame when Step Content menu item is
selected or
     * Step Content button from Trace Frame is pressed
     *
     * @param traceFrame :  = null if launched from Step Content menu
item
     *                  != null if launched from Step Content button
     * @param pathName : the path of current step
     * @param atomics : vector of the current step's atomics
     */
    public StepContentFrame(TraceFrame traceFrame, String stepName,
String pathName, Vector atomics)
            {
        this();
                this.atomicsVector = atomics;
                this.selectedLabel.setText( pathName );
                this.stepVersionTextField.setText( stepName );
                this.pathName = pathName;
                setStakeHolders();
            symantec.itools.awt.util.Calendar theCalendar = new
symantec.itools.awt.util.Calendar();
                try{
                   File f = new File(pathName+"\\step.cnt");
                   if( f.exists() ){
                      FileInputStream fileInput = new FileInputStream(f);
                      ObjectInputStream oi = new ObjectInputStream(
fileInput );
                      if( oi != null ){
                         setInitial((StepContent)oi.readObject());
                      }
                      else{
                         try{
                         DateFormat df = DateFormat.getDateInstance();
                         Date d = df.parse(theCalendar.getDate());
            deadlineTextField.setText(d.toString());
            earliestSTTextField.setText(d.toString());
            finishTimeTextField.setText(d.toString());
            }
            catch(Exception e){System.out.println(e);}
                      }
                      oi.close();
                      fileInput.close();
        }
                }
                catch( IOException io ){
                   debug("IOException: "+io);
                   try{
                      DateFormat df = DateFormat.getDateInstance();
                      Date d = df.parse(theCalendar.getDate());
        deadlineTextField.setText(d.toString());
```

```
  earliestSTTextField.setText(d.toString());
  finishTimeTextField.setText(d.toString());
}
catch( Exception e){System.out.println(e);}

        }
        catch( ClassNotFoundException c ){
          debug("ClassNotFoundException: "+c);
        }
        if( traceFrame != null ){
          setReadOnly();
        }
        else{
          saveButton.setText("Save");
          saveButton.setActionCommand("Save");
          getContentPane().add(saveButton);
          saveButton.setBounds(440,470,75,24);
          deleteButton.setText("Delete");
          deleteButton.setActionCommand("Save");
          getContentPane().add(deleteButton);
          deleteButton.setBounds(515,470,75,24);
        }
  }

  public StepContentFrame(String sTitle)
  {
        this();
        setTitle(sTitle);
  }

  public void setVisible(boolean b)
  {
        if (b)
                setLocation(50, 50);
        super.setVisible(b);
  }

        static public void main(String args[])
        {
                (new StepContentFrame()).setVisible(true);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets and menu
bar
                Insets insets = getInsets();
                com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
                int menuBarHeight = 0;
                if (menuBar != null)
                        menuBarHeight =
menuBar.getPreferredSize().height;
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JButton saveButton = new
com.sun.java.swing.JButton();
```

```
        com.sun.java.swing.JButton deleteButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton exitButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel titleLable = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField stepVersionTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox managerComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox statusComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel4 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel5 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel6 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox evaluatorComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel7 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox securityLevelComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox organizerComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JComboBox predecessorsComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel12 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JComboBox priorityComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JComboBox skillComboBox = new
com.sun.java.swing.JComboBox();
        com.sun.java.swing.JLabel JLabel16 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField evaluationTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField deadlineTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField earliestSTTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField finishTimeTextField = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel selectedLabel = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JButton predecessorsButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton skillButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton deadlineButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton startTimeButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton finishTimeButton = new
com.sun.java.swing.JButton();
        //com.sun.java.swing.JTextField estDurationTextField = new
com.sun.java.swing.JTextField();
        MyTextField estDurationTextField = new MyTextField();
        //}}

        //{{DECLARE_MENUS
        //}}

        class SymAction implements java.awt.event.ActionListener
```

```
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == saveButton)
                                saveButton_actionPerformed(event);
                        else if (object == deleteButton)
                                deleteButton_actionPerformed(event);
                        else if (object == exitButton)
                                exitButton_actionPerformed(event);
                        else if (object == predecessorsButton)

        predecessorsButton_actionPerformed(event);
                        else if (object == skillButton)
                                skillButton_actionPerformed(event);
                        else if (object == deadlineButton)

        deadlineButton_actionPerformed(event);
                        else if (object == startTimeButton)

        startTimeButton_actionPerformed(event);
                        else if (object == finishTimeButton)

        finishTimeButton_actionPerformed(event);
                }
        }

  /**
   * Create a step content object and save it in step.cnt file
   * under the current step path.  Then exit this frame
   */
        public void
saveButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    try{
        FileOutputStream fileOutput = new
FileOutputStream(this.pathName+"\\step.cnt");
        ObjectOutputStream oo = new ObjectOutputStream(fileOutput);
        if( oo != null ){
          oo.writeObject((StepContent)getStepContent());
        }
        oo.flush();
        oo.close();
        fileOutput.close();
      }
      catch(IOException io){
        debug("IOException: "+io);
      }
      exitButton_actionPerformed(event);
          }

  /**
   * Confirm message will ask a user before agree to delete step.cnt file
   */
        public void
deleteButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          File f = new File(this.pathName, "step.cnt");
          if( f.exists() ){
            int result = JOptionPane.showConfirmDialog( this, "step.cnt
file will be deleted.  Would you like to continue?",
                        "Confirm
Message",JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
        //result = 0 ==> yes
        //result = 1 ==> no
        if( result == 0 ){
          f.delete();
          exitButton_actionPerformed(event);
            }
          }
          else{
```

```
            JOptionPane.showMessageDialog(this, "step.cnt file does not
exist!",
                        "Warning
Message",JOptionPane.INFORMATION_MESSAGE);
            }
        }

  /**
   * Exit this frame
   */
        public void
exitButton_actionPerformed(java.awt.event.ActionEvent event)
        {
          setVisible( false );
          dispose();
        }

  /**
   * Set initial of this frame if the selected step already created step.cnt file
   *
   * @param stepContent : step content object of this current step
   */
        public void setInitial( StepContent stepContent ){
          Vector temp = new Vector();


this.stepVersionTextField.setText((String)stepContent.getStepName());

this.statusComboBox.setSelectedItem((String)stepContent.getStatus());

          //Set up skill comboBox
          temp = (Vector)stepContent.getSkill();
    if( temp != null ){
            this.selectedSkill = new Vector();
          this.skillComboBox.addItem(SKILL_TITLE);
            for( int i=0; i<temp.size(); i++ ){
              this.skillComboBox.addItem(temp.elementAt(i));
              this.selectedSkill.addElement(temp.elementAt(i));
            }
          }

this.securityLevelComboBox.setSelectedItem(""+stepContent.getSecurityL
evel());

this.evaluationTextField.setText((String)stepContent.getEvaluation());

this.evaluatorComboBox.setSelectedItem((String)stepContent.getEvaluator(
));

this.organizerComboBox.setSelectedItem((String)stepContent.getOrganizer
());

          //Set up predecessors comboBox
          temp = new Vector();
          temp = (Vector)stepContent.getPredecessors();
    if( temp != null ){
            this.selectedPred = new Vector();
          this.predecessorsComboBox.addItem("Current Selected List");
            for( int i=0; i<temp.size(); i++ ){
              this.predecessorsComboBox.addItem(temp.elementAt(i));
              this.selectedPred.addElement(temp.elementAt(i));
            }
          }

this.priorityComboBox.setSelectedItem(""+stepContent.getPriority());

this.estDurationTextField.setText(""+stepContent.getDuration());

this.deadlineTextField.setText((String)stepContent.getDeadline());

this.earliestSTTextField.setText((String)stepContent.getStartTime());

this.finishTimeTextField.setText((String)stepContent.getFinishTime());
```

```
this.managerComboBox.setSelectedItem((String)stepContent.getManager())
;
        }

  /**
   * Create a step content object before save it
   *
   * @return stepContent object with all the information from this frame
   */
        public StepContent getStepContent(){
            int security = (new
Integer((String)this.securityLevelComboBox.getSelectedItem())).intValue()
;
            int priority = (new
Integer((String)this.priorityComboBox.getSelectedItem())).intValue();
            String estDur = (String)this.estDurationTextField.getText();
            int duration = 0;
            if( !estDur.equals("") ){
                duration = (new Integer(estDur)).intValue();
            }
            StepContent stepContent = new StepContent();


stepContent.setStepName((String)this.stepVersionTextField.getText());

stepContent.setStatus((String)this.statusComboBox.getSelectedItem());
            stepContent.setSkill((Vector)this.selectedSkill);
            stepContent.setSecurityLevel(security);

stepContent.setEvaluation((String)this.evaluationTextField.getText());

stepContent.setEvaluator((String)this.evaluatorComboBox.getSelectedItem(
));

stepContent.setOrganizer((String)this.organizerComboBox.getSelectedItem
());
```



```
            stepContent.setPredecessors((Vector)this.selectedPred);
            stepContent.setPriority(priority);
            stepContent.setDuration(duration);

stepContent.setDeadline((String)this.deadlineTextField.getText());

stepContent.setStartTime((String)this.earliestSTTextField.getText());

stepContent.setFinishTime((String)this.finishTimeTextField.getText());

stepContent.setManager((String)this.managerComboBox.getSelectedItem())
;

            return stepContent;
        }

  /**
   * Get all personnel objects in stakeholder directory,
   * and add them in evaluator, organizer, and manager combo boxes
   */
        public void setStakeHolders(){
            String[] list = (String[])STAKEHOLDER.list();
            if( list.length > 0 ){
                for( int i=0; i<list.length; i++ ){
          this.evaluatorComboBox.addItem(list[i]);
          this.organizerComboBox.addItem(list[i]);
          this.managerComboBox.addItem(list[i]);
                }
            }
        }

  /**
   * This function is called when SkillTableFrame's OK button is pressed,
   * it will return a vector of selected skills.  Use this vector to set
   * skillComboBox
   *
   * @param v : vector of skills from SkillTableFrame
```

```
 */
public void setSkillComboBox(Vector v){
   this.selectedSkill = new Vector();
   this.skillComboBox.removeAllItems();
   if( v.size() > 0 ){
         this.skillComboBox.addItem(SKILL_TITLE);
            for( int i=0; i<v.size(); i++ ){
               this.skillComboBox.addItem(v.elementAt(i));
      this.selectedSkill.addElement(v.elementAt(i));
            }
   }
      }

/**
 * Launch ListDialog to allows a user to select more than one atomics
 */
      public void
predecessorsButton_actionPerformed(java.awt.event.ActionEvent event)
      {
         Vector v = new Vector();
         String parentPath = CASESDIRECTORY.getAbsolutePath();
         int length = parentPath.length();
         debug("Path length = "+length);
         for( int i=0; i<this.atomicsVector.size(); i++ ){
            File f = (File) this.atomicsVector.elementAt(i);
            String s = f.getAbsolutePath();
            String sub1 = s.substring(length);
            int index = sub1.indexOf("\\");
            String sub2 = sub1.substring(index+1);

            debug("Sub1 = "+sub1);
            debug("Sub2 = "+sub2);
            StringTokenizer st = new StringTokenizer(sub2, "\\");
            String theAtomic = null;
            int j=0;
            while( st.hasMoreTokens()){
               String theString = st.nextToken();
               if( j==0 ){
                  theAtomic = theString;
               }
               else if( j==1){
                  theAtomic = theAtomic + theString;
               }
               else if( j==2){
                  theAtomic = theAtomic + "-" + theString;
               }
               else if( j>2){
                  theAtomic = theAtomic + "." + theString;
               }
               j++;
            }
            v.addElement(theAtomic);
         }
   (new ListDialog(this, "Predecessor List", v)).setVisible(true);
      }

/**
 * Launch SkillTableFrame to allows a user to select more than one
skills
 */
      public void
skillButton_actionPerformed(java.awt.event.ActionEvent event)
      {
   (new SkillTableFrame(this)).setVisible(true);
      }

/**
 * It is only called when Step Content button in TraceFrame is pressed.
 * The purpose is to view the content of this step, and the user won't
allow
 * to create/edit/delete/save this step content at all.
 */
      public void setReadOnly(){
            managerComboBox.setEnabled( false );
```

```
                statusComboBox.setEnabled( false );
                evaluatorComboBox.setEnabled( false );
                securityLevelComboBox.setEnabled( false );
                priorityComboBox.setEnabled( false );
                organizerComboBox.setEnabled( false );
                predecessorsButton.setEnabled( false );
                skillButton.setEnabled( false );
        evaluationTextField.setEditable( false );
        estDurationTextField.setEditable( false );
        deadlineTextField.setEditable( false );
        earliestSTTextField.setEditable( false );
        finishTimeTextField.setEditable( false );

            evaluationTextField.setBackground(java.awt.Color.white);

            estDurationTextField.setBackground(java.awt.Color.white);
                deadlineTextField.setBackground(java.awt.Color.white);

            earliestSTTextField.setBackground(java.awt.Color.white);

            finishTimeTextField.setBackground(java.awt.Color.white);
              startTimeButton.setEnabled( false );
              deadlineButton.setEnabled( false );
              finishTimeButton.setEnabled( false );
            }

    /**
     * Launch CalendarDialog
     */
            public void
startTimeButton_actionPerformed(java.awt.event.ActionEvent event)
            {
              String s = earliestSTTextField.getText();
              if( !s.equals("") ){
                try{
                  (new CalendarDialog(this,s,0)).setVisible(true);
                }
                catch(Exception e){System.out.println(e);}
              }
              else{
                (new CalendarDialog(this,s,0)).setVisible(true);
              }
            }

    /**
     * Launch CalendarDialog
     */
            public void
deadlineButton_actionPerformed(java.awt.event.ActionEvent event)
            {
              String s = deadlineTextField.getText();
              if( !s.equals("") ){
                try{
                  (new CalendarDialog(this,s,1)).setVisible(true);
                }
                catch(Exception e){System.out.println(e);}
              }
              else{
                (new CalendarDialog(this,s,1)).setVisible(true);
              }
            }

    /**
     * Launch CalendarDialog
     */
            public void
finishTimeButton_actionPerformed(java.awt.event.ActionEvent event)
            {
              String s = finishTimeTextField.getText();
              if( !s.equals("") ){
                try{
                  (new CalendarDialog(this,s,2)).setVisible(true);
                }
                catch(Exception e){System.out.println(e);}
```

```
    }
    else{
            (new CalendarDialog(this,s,2)).setVisible(true);
    }
        }

  /**
   * Short cut to print the output
   * @param string : the output string
   */
        void debug(String s){
          System.out.println(s);
        }

        /**
         * Custom JTextfield.  It is used to create estDurationTextField
since
         * it only allows a user to input integer. And, JTextField doesn't
have
         * that capability.
         */
  public class MyTextField extends JTextField
  {
   protected void processKeyEvent( KeyEvent e )
   {
    if ( e.getModifiers() == 0 )
    {
      int keyChar = e.getKeyChar();
      if ( keyChar >= '0' && keyChar <= '9' ||
          keyChar == '\n' ){
            super.processKeyEvent( e );
      }
    }
   }
  }

  /**
```

```
   * Get an array of selected atomics from ListDialog and set
predecessorsComboBox
   *
   * @param oa : an array of selected atomics, and elements of this array
are File objects
   */
  public void setPredecessorComboBox(Object[] oa){
            this.predecessorsComboBox.removeAllItems();
            this.selectedPred = new Vector();
     if( oa.length > 0 ){
            this.predecessorsComboBox.addItem("Current Selected List");
       for( int i=0; i<oa.length; i++ ){
                this.predecessorsComboBox.addItem(oa[i]);
                this.selectedPred.addElement(oa[i]);
              }
     }
   }
}
```

```
package Cases;

/**
 * Step Type Object which is used to save in step.cfg file
 */

import java.io.Serializable;

//////////////////////////////////////////////////////////////////////////////
/**
 * StepType : Create a step type object and save it in step.cfg file
 */
```

```
//////////////////////////////////////////////////////////////////////////////
public class StepType implements Serializable{

  /**
   * stepID : step type ID
   */
  private String stepID = null;

  /**
   * stepName : name of the step type
   */
  private String stepName = null;

  /**
   * stepDescription : description of the step
   */
  private String stepDescription = null;

  /**
   * This StepType constructor is used to create step type object
   */
  public StepType( String stepID, String stepName, String stepDescription
){
    this.stepID = stepID;
    this.stepName = stepName;
    this.stepDescription = stepDescription;
  }

  public StepType() {}

  /**
   * Step type ID
   *
   * @param stepID : step type ID
   */
  public String getStepID(){
    return this.stepID;
  }

  /**
   * Step type name
   *
   * @param stepName : name of the step type
   */
  public String getStepName(){
    return this.stepName;
  }

  /**
   * Step type description
   *
   * @param stepDescription : the description of step type
   */
  public String getStepDescription(){
    return this.stepDescription;
  }
}
```

```
package Cases;

import java.awt.*;
import java.util.*;
import java.io.*;
import com.sun.java.swing.*;

//////////////////////////////////////////////////////////////////////////////
/**
 * TraceFrame : the main purpose of this frame is to view and trace the step
and
 * substeps
```

```
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Trace
 */
//////////////////////////////////////////////////////////////////////////////
public class TraceFrame extends com.sun.java.swing.JFrame implements
CasesTitle, I_Trace
{
  /**
   * casesFrame : parent frame which launch this frame
   */
  CasesFrame casesFrame = null;

  /**
   * storedVector : stores contents of link files
   */
  public Vector[] storedVector = {new Vector(), new Vector(), new
Vector(), new Vector(), new Vector(), new Vector()};

  /**
   * currentIndex : monitor the position of current step in the history vector
   */
  public int currentIndex = 0;

  /**
   * currentLocation : make sure the current step is inserted at the right
position in the history vector
   */
  public int currentLocation = 0;

  /**
   * pathName : the path of the current project
   */
  public String pathName = null;

  /**
   * history : a vector of all selected steps in this frame
   */
  public Vector history = new Vector();

  /**
   * fnList : a vector of steps, strings, in the current project
   */
  public Vector fnList = new Vector();

  /**
   * currentPath : a string to keep the current step when tracing around
   */
  public String currentPath = null;

  /**
   * output : holds the output of the current step
   */
  private String output = "";

  /**
   * stepVersion : holds the selected step version
   */
  private String stepVersion = "";

  /**
   * Build TraceFrame
   */
        public TraceFrame()
        {
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
```

```
//{{INIT_CONTROLS
setTitle("SPIDER-Trace");
getContentPane().setLayout(null);
setSize(670,320);
setVisible(false);
forwardButton.setText("Forward");
forwardButton.setActionCommand("Forward");
getContentPane().add(forwardButton);
forwardButton.setBounds(68,1,81,24);
backwardButton.setText("Backward");
backwardButton.setActionCommand("Backward");
getContentPane().add(backwardButton);
backwardButton.setBounds(150,1,92,24);
homeButton.setText("Home");
homeButton.setActionCommand("Home");
getContentPane().add(homeButton);
homeButton.setBounds(1,1,67,24);
stepVersionTextField.setEditable(false);
getContentPane().add(stepVersionTextField);

stepVersionTextField.setBackground(java.awt.Color.white);
stepVersionTextField.setBounds(242,55,300,24);
outputTextField.setEditable(false);
getContentPane().add(outputTextField);
outputTextField.setBackground(java.awt.Color.white);
outputTextField.setBounds(242,100,300,24);

JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel1.setText("Step Version");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setBounds(55,55,180,24);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel2.setText("Primary Input Component(s)");
getContentPane().add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(55,145,180,24);

JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel3.setText("Secondary Input Component(s)");
getContentPane().add(JLabel3);
JLabel3.setForeground(java.awt.Color.black);
JLabel3.setBounds(55,190,180,24);

JLabel4.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.RIGHT);
JLabel4.setText("Output Component");
getContentPane().add(JLabel4);
JLabel4.setForeground(java.awt.Color.black);
JLabel4.setBounds(55,100,180,24);
closeButton.setText("Close");
closeButton.setActionCommand("OK");
getContentPane().add(closeButton);
closeButton.setBounds(302,252,75,24);
stepContentButton.setText("Step Content");
stepContentButton.setActionCommand("Step Content");
getContentPane().add(stepContentButton);
stepContentButton.setBounds(560,1,107,24);
primaryTextField.setEditable(false);
getContentPane().add(primaryTextField);
primaryTextField.setBackground(java.awt.Color.white);
primaryTextField.setBounds(242,145,300,24);
secondaryTextField.setEditable(false);
getContentPane().add(secondaryTextField);

secondaryTextField.setBackground(java.awt.Color.white);
secondaryTextField.setBounds(242,190,300,24);
traceButton.setText("Trace");
traceButton.setActionCommand("Trace");
getContentPane().add(traceButton);
```

```
traceButton.setBounds(242,1,67,24);
decomposeButton.setText("Decompose");
decomposeButton.setActionCommand("Decompose");
getContentPane().add(decomposeButton);
decomposeButton.setBounds(310,1,102,24);
componentButton.setText("Component Content");
componentButton.setActionCommand("Component
Content");
getContentPane().add(componentButton);
componentButton.setBounds(413,1,146,24);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
homeButton.addActionListener(lSymAction);
forwardButton.addActionListener(lSymAction);
backwardButton.addActionListener(lSymAction);
closeButton.addActionListener(lSymAction);
stepContentButton.addActionListener(lSymAction);
traceButton.addActionListener(lSymAction);
decomposeButton.addActionListener(lSymAction);
componentButton.addActionListener(lSymAction);
//}}

//Grey out when the history vector is empty
forwardButton.setEnabled( false );
backwardButton.setEnabled( false );
}

/**
 * CasesFrame launch this frame when Trace menu item is selected
 */
public TraceFrame( CasesFrame casesFrame, String pathName, Vector
componentsVector, Vector fnList ){
this();
this.casesFrame = casesFrame;
this.pathName = pathName;
this.fnList = fnList;
setInitial(componentsVector);
}

public TraceFrame(String sTitle)
{
this();
setTitle(sTitle);
}

public void setVisible(boolean b)
{
if (b)
setLocation(50, 50);
super.setVisible(b);
}

static public void main(String args[])
{
(new TraceFrame()).setVisible(true);
}

public void addNotify()
{
// Record the size of the window prior to calling parents
addNotify.
Dimension size = getSize();

super.addNotify();

if (frameSizeAdjusted)
return;
frameSizeAdjusted = true;
```

```
		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JButton forwardButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton backwardButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton homeButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JTextField stepVersionTextField = new
com.sun.java.swing.JTextField();
	com.sun.java.swing.JTextField outputTextField = new
com.sun.java.swing.JTextField();
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JLabel JLabel4 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JButton closeButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton stepContentButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JTextField primaryTextField = new
com.sun.java.swing.JTextField();
	com.sun.java.swing.JTextField secondaryTextField = new
com.sun.java.swing.JTextField();
	com.sun.java.swing.JButton traceButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton decomposeButton = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton componentButton = new
com.sun.java.swing.JButton();
	//}}

	//{{DECLARE_MENUS
	//}}

  class SymAction implements java.awt.event.ActionListener
	{
		public void actionPerformed(java.awt.event.ActionEvent
event)
		{
			Object object = event.getSource();
			if (object == homeButton)
				homeButton_actionPerformed(event);
			else if (object == closeButton)
				closeButton_actionPerformed(event);
			else if (object == forwardButton)
				forwardButton_actionPerformed(event);
			else if (object == backwardButton)

	backwardButton_actionPerformed(event);
			else if (object == stepContentButton)

	stepContentButton_actionPerformed(event);
			else if (object == traceButton)
				traceButton_actionPerformed(event);
```

```
                    else if (object == decomposeButton)

        decomposeButton_actionPerformed(event);
                    else if (object == componentButton)

        componentButton_actionPerformed(event);
                }
        }

    /**
     * View the first step in the history vector
     */
        public void
homeButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    this.currentIndex = 0;
    String s = (String)this.history.elementAt(this.currentIndex);
    this.currentPath = (String)convertToThePath(s);
    searchPath(s);
        }

    /**
     * View the step at the currentIndex+1 postion in the history vector
     */
        public void
forwardButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    if( (++this.currentIndex >= 0 ) && (this.currentIndex <
this.history.size()) ){
        String s = (String)this.history.elementAt(this.currentIndex);
        this.currentPath = (String)convertToThePath(s);
        searchPath(s);
    }
        }

    /**
     * View the step at the currentIndex-1 postion in the history vector
     */
        public void
backwardButton_actionPerformed(java.awt.event.ActionEvent event)
        {
    if ( --this.currentIndex >= 0 ){
        String s = (String)this.history.elementAt(this.currentIndex);
        this.currentPath = (String)convertToThePath(s);
        searchPath(s);
    }
        }

    /**
     * Launch StepContentFrame to view the content of this current step
     */
        public void
stepContentButton_actionPerformed(java.awt.event.ActionEvent event)
        {
        String s = (String)this.history.elementAt(this.currentIndex);
        this.currentPath = (String)convertToThePath(s);
    this.casesFrame.setStepContent(this, "s-"+s, this.currentPath );
    }

    /**
     * Exit TraceFrame
     */
        public void
closeButton_actionPerformed(java.awt.event.ActionEvent event)
        {
        setVisible(false);
        dispose();
        }

    /**
     * Short cut to print the output
     * @param string : the output string
     */
        public void debug(String s){
```

```
            System.out.println(s);
        }

    /**
     * Set intitial of this frame with the selected step from JFileChooser
     *
     * @param componentsVector : a vector of strings holds all component
names of the selected step
     */
    public void setInitial(Vector componentsVector){
        String s = (String)componentsVector.elementAt(0);
        setHistory(s);
        this.currentPath = (String)convertToThePath(s);
        if( this.currentPath != null ){
            searchPath(s);
        }
    }

    /**
     * Hold all steps which have been viewed and traced
     *
     * @param s : the current step name
     */
    public void setHistory( String s){
        this.history.insertElementAt(s,currentLocation++);
    }

    /**
     * Convert from a selected step into the complete file path of this step
and
     * make sure this step is valid or not
     *
     * @param s : selected step name
     */
    public String convertToThePath(String s){
        String thePath = null;
        output = s;
        s= "s-"+s;
        stepVersion = s;

        //To convert the string of selected item into the file path
        for( int i=0; i<this.fnList.size(); i++ ){
            String fn = (String)this.fnList.elementAt(i);
            String sub = s.substring(0,fn.length());

            if( sub.equals(fn)){
                i = this.fnList.size();
                thePath= this.pathName+"\\"+sub+"\\";
                String sub2 = s.substring(fn.length());

                char[] ca = (char[])sub2.toCharArray();
                int result = 0;
                for( int j=0; j<ca.length; j++ ){
                    String s3 = ca[j]+"";
                    result = s3.compareTo("-");
                    if( result == 0 ){
                        j=ca.length;
                        StringTokenizer st = new StringTokenizer(sub2,"-");
                            String before_ = st.nextToken("-");
                            thePath = thePath+before_;
                            String _after = st.nextToken("-");
                            st = new StringTokenizer(_after,".");
                            while(st.hasMoreTokens()){
                                thePath = thePath+"\\"+st.nextToken();
                            }
                    }
                    else if( (result > 0) && (j==ca.length - 1) ){
                        thePath = thePath+sub2;
                    }
                }
            }
        }
        if( thePath == null ){
            JOptionPane.showMessageDialog(this, s+" is invalid name.",
```

```
                "Error Message",JOptionPane.ERROR_MESSAGE);
        return null;
      }
      return thePath;
    }

    /**
     * Search the path of this step
     *
     * @param s : the selected step name
     */
    public void searchPath( String s){
      searchIndex();
      if( this.currentPath != null ){
        searchInputFiles( this.currentPath );
      }
    }

    /**
     * Search input.p and input.s files of the current step, and get theirs
     * contents to insert in the primary and secondary textfields directly
     *
     * @param thePath : the complete path of the selected step
     */
    public void searchInputFiles( String thePath ){
      clearTextFields();
      try{
                File f = new File(thePath+"\\input.p");
                if( f.exists() ){
                  FileInputStream fileInputP = new
FileInputStream(f);
                  DataInputStream primIn = new DataInputStream(
fileInputP );
                  if( primIn != null ){
                    String s = (String)primIn.readLine();
                    if( (s != null) && (!s.equals("")) ){
                      this.primaryTextField.setText( s );
                    }
                  }
                  primIn.close();
                  fileInputP.close();
                }
                f = new File(thePath+"\\input.s");
                if( f.exists() ){
                  FileInputStream fileInputS = new FileInputStream(f);
                  DataInputStream secondIn = new DataInputStream(
fileInputS);
                  if( secondIn != null ){
                    String s = (String)secondIn.readLine();
                    if( (s != null) && (!s.equals("")) ){
                      this.secondaryTextField.setText( s );
                    }
                  }
                  secondIn.close();
                  fileInputS.close();
                }
                this.outputTextField.setText(output);
                this.stepVersionTextField.setText(stepVersion);
                }
                catch( IOException io ){
              debug("IOException: "+io);
                }
    }

    /**
     * To find which button (home, forward, or backward) should be set
enabled
     * and it must match with the currentIndex in history vector
     */
    public void searchIndex(){
      if( this.currentIndex < this.history.size() ){
        if( this.currentIndex == (this.history.size()-1) ){
          this.forwardButton.setEnabled( false );
        }
```

```
        else{
            this.forwardButton.setEnabled( true );
        }
    }
    if( this.currentIndex > 0 ){
        this.backwardButton.setEnabled( true );
    }
    else{
        this.backwardButton.setEnabled( false );
    }
}

/**
 * Get selected item from ListDialog and DecomposeListDialog
 *
 * @param currentPath : reverse from a selected item into the path of
this component
 * @param selectedItem : a selected component names
 */
public void setSelectedItem(String currentPath, String selectedItem){
    this.currentPath = currentPath;
    setHistory(selectedItem);
    this.currentIndex = this.history.size() - 1;
    searchPath(currentPath);
}

/**
 * Searching aFile and get its content to insert into one of elements of
storedVector
 *
 * @param aFile : link files
 * @param fileType : type of link file, eg. txt.link, word.link, ...
 */
        public void searchFiles(File aFile, String fileType){
    File f = new File( aFile, fileType);
    try{
        BufferedReader br = null;
```

```
String item = null;
if( fileType.equals(LINK_FILE_NAMES[0]) ){
    br = new BufferedReader( new FileReader(f));
    if( br != null ){
        while( (item = br.readLine()) != null ){
            this.storedVector[0].addElement(item);
        }
    }
}
else if( fileType.equals(LINK_FILE_NAMES[1]) ){
    br = new BufferedReader( new FileReader(f));
    if( br != null ){
        while( (item = br.readLine()) != null ){
            this.storedVector[1].addElement(item);
        }
    }
}
else if( fileType.equals(LINK_FILE_NAMES[2]) ){
    br = new BufferedReader( new FileReader(f));
    if( br != null ){
        while( (item = br.readLine()) != null ){
            this.storedVector[2].addElement(item);
        }
    }
}
else if( fileType.equals(LINK_FILE_NAMES[3]) ){
    br = new BufferedReader( new FileReader(f));
    if( br != null ){
        while( (item = br.readLine()) != null ){
            this.storedVector[3].addElement(item);
        }
    }
}
else if( fileType.equals(LINK_FILE_NAMES[4]) ){
    br = new BufferedReader( new FileReader(f));
    if( br != null ){
        while( (item = br.readLine()) != null ){
```

```
                this.storedVector[4].addElement(item);
            }
        }
    }
    else if( fileType.equals(LINK_FILE_NAMES[5]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.storedVector[5].addElement(item);
            }
        }
    }
}
catch( IOException io ){
    debug("IOException: "+io);
}
    }

/**
 * Create a file with a specific string s if s is valid
 *
 * @param s : a selected string
 * @return f : a file with the name is s
 */
public File searchFilePath( String s){
    String thePath = null;

    s = "s-"+s;
    //To convert the string of selected item into the file path
    for( int i=0; i<this.fnList.size(); i++ ){
        String fn = (String)this.fnList.elementAt(i);
        String sub = s.substring(0,fn.length());

        if( sub.equals(fn)){
            i = this.fnList.size();
            thePath= sub+"\\";
            String sub2 = s.substring(fn.length());
            char[] ca = (char[])sub2.toCharArray();
            int result = 0;
            for( int j=0; j<ca.length; j++ ){
                String s3 = ca[j]+"";
                result = s3.compareTo("-");
                if( result == 0 ){
                    j=ca.length;
                    StringTokenizer st = new StringTokenizer(sub2,"-");
                        String before_ = st.nextToken("-");
                        thePath = thePath+before_;
                        String _after = st.nextToken("-");
                        st = new StringTokenizer(_after,".");
                        while(st.hasMoreTokens()){
                            thePath = thePath+"\\"+st.nextToken();
                        }
                }
                else if( (result > 0) && (j==ca.length - 1) ){
                    thePath = thePath+sub2;
                }
            }
        }
    }
    File f = new File(this.pathName,thePath);
    return f;
}

/**
 * Check a selected string is valid or invalid file name in the current
project
 *
 * @param selectedItem : a selected component name
 * @return a component name which matchs in the current project
 */
public String checkSelection(String selectedItem ){
    if( selectedItem != null ){
        int j = selectedItem.indexOf("-");
```

```
        if( j>0){
                String sub2 = (selectedItem.substring(j+1)).trim();
                char c = (char)sub2.charAt(0);
                boolean isLetter = (new Character(c)).isLetter(c);
                if( isLetter ){
            return sub2;
          }
          else{
            return selectedItem;
          }
        }
        else{
          return selectedItem;
        }
      }
      return selectedItem;
        }

    /**
     * Tokenize a string with the delimit is ","
     *
     * @param v : a vector of words without ","
     * @param s : a string is tokenized
     */
    public void tokenizer( Vector v, String s){
            StringTokenizer st = new StringTokenizer(s,",");
            while(st.hasMoreTokens()){
              String theString = (st.nextToken()).trim();
              v.addElement(theString);
            }
        }

    /**
     * View the content of available steps in the ListDialog
     */
        public void
traceButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            Vector v = new Vector();
            tokenizer(v, primaryTextField.getText());
            tokenizer(v, secondaryTextField.getText());
            (new ListDialog(this, "Trace Component", v,
0)).setVisible(true);
        }

    /**
     * View the content of available decomposed steps of the current step
     */
        public void
decomposeButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            Vector v = new Vector();
            v.addElement(outputTextField.getText());
            tokenizer(v, primaryTextField.getText());
            tokenizer(v, secondaryTextField.getText());
            (new DecomposeListDialog(this, "Decompose",
v)).setVisible(true);
        }

    /**
     * View all links and connect these links to their applications
     * of available steps in ListDialog
     */
        public void
componentButton_actionPerformed(java.awt.event.ActionEvent event)
        {
            Vector v = new Vector();
            v.addElement(outputTextField.getText());
            tokenizer(v, primaryTextField.getText());
            tokenizer(v, secondaryTextField.getText());
            (new ListDialog(this, "Component Content", v,
1)).setVisible(true);
        }
```

```
/**
 * Refresh all textfiels in this frame
 */
        public void clearTextFields(){
            this.stepVersionTextField.setText("");
            this.outputTextField.setText("");
            this.primaryTextField.setText("");
            this.secondaryTextField.setText("");
        }

/**
 * Launch ReviewComponentContentDialog after select one component
from ListDialog
 *
 * @param selectedItem : a selected component name from ListDialog
 * @param f : file with the name is selectedItem
 */
        public void setComponentContent(String selectedItem, File f){
    String[] list = f.list();
    for( int j=0; j<list.length; j++ ){
        String s = (String)list[j];
        File aFile = null;
        if( s.equals(COMPONENT_CONTENT_DIR) ){
            aFile = new File(f, s);
            if( aFile.isDirectory() ){
                j = list.length;
                list = aFile.list();
                for( int k=0; k<storedVector.length; k++ ){
                    storedVector[k] = new Vector();
                }
                for( int i=0; i<list.length; i++ ){
                    searchFiles(aFile, (String)list[i]);
                }
                (new ReviewComponentContentDialog(selectedItem,
this.storedVector)).setVisible(true);
            }
        }
    }
        }
}


package Cases;

/**
 * Version Control Object which is used to save in the
 * current.vsn file
 */

import java.io.Serializable;

////////////////////////////////////////////////////////////////////////////////
/**
 * VersionControl : Create a version control object and save it in
current.vsn file
 */
////////////////////////////////////////////////////////////////////////////////
public class VersionControl implements Serializable{
    /**
     * currentLoop : current process of the project
     */
    private String currentLoop = null;

    /**
     * currentStep : current step of the current process
     */
    private String currentStep = null;

    /**
     * current variant : current variant of the current step
     */
    private String currentVariant = null;
```

```
/**
 * currentVersion : current version of the current step
 */
private String currentVersion = null;

/**
 * currentStatus : current status of the current step, eg, Completed,
Approved, ...
 */
private String currentStatus = "";

/**
 * VersionControl constructor with 5 elements
 */
public VersionControl( String currentLoop, String currentStep, String
currentVariant,
                String currentVersion, String currentStatus )
{
   this.currentLoop = currentLoop;
   this.currentStep = currentStep;
   this.currentVariant = currentVariant;
   this.currentVersion = currentVersion;
   this.currentStatus = currentStatus;
}

/**
 * VersionControl constructor with 4 elements
 */
public VersionControl( String currentLoop, String currentStep, String
currentVersion,
                String currentStatus )
{
   this.currentLoop = currentLoop;
   this.currentStep = currentStep;
   this.currentVersion = currentVersion;
   this.currentStatus = currentStatus;
}
```



```
/**
 * VersionControl constructor with 3 elements
 */
public VersionControl( String currentLoop, String currentStep, String
currentVersion )
{
   this.currentLoop = currentLoop;
   this.currentStep = currentStep;
   this.currentVersion = currentVersion;
}

public VersionControl() { }

/**
 * @return currentLoop : current process of this version control object
 */
public String getCurrentLoop(){
   return this.currentLoop;
}

/**
 * @return currentStep : current step of this version control object
 */
public String getCurrentStep(){
   return this.currentStep;
}

/**
 * @return currentVersion : current version of this version control object
 */
public String getCurrentVersion(){
   return this.currentVersion;
}

/**
 * @return currentVariant : current variant of this version control object
```

```
    */
   public String getCurrentVariant(){
      return this.currentVariant;
   }

   /**
    * @return currentStatus : current status of this version control object
    */
   public String getCurrentStatus(){
      return this.currentStatus;
   }
}
```



```
package JobSchedule;

/*
                A basic implementation of the JDialog class.
*/

import java.awt.*;
import com.sun.java.swing.*;

public class JAboutDialog extends com.sun.java.swing.JDialog
{
        public JAboutDialog(Frame parentFrame)
        {
                super(parentFrame);
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("JFC Application - About");
                setModal(true);
                getContentPane().setLayout(new GridBagLayout());
                setSize(248,94);
                setVisible(false);
                okButton.setText("OK");
                okButton.setActionCommand("OK");
                okButton.setOpaque(false);
                okButton.setMnemonic((int)'O');
                getContentPane().add(okButton, new
com.symantec.itools.awt.GridBagConstraintsD(2,1,1,1,0.0,0.0,java.awt.Gri
dBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
Insets(0,0,10,0),0,0));
                okButton.setBounds(98,59,51,25);

        aboutLabel.setHorizontalAlignment(com.sun.java.swing.SwingCo
nstants.CENTER);
                aboutLabel.setText("A JFC Application");
                getContentPane().add(aboutLabel, new
com.symantec.itools.awt.GridBagConstraintsD(0,0,3,1,1.0,1.0,java.awt.Gri
dBagConstraints.CENTER,java.awt.GridBagConstraints.BOTH,new
Insets(0,0,0,0),0,0));
                aboutLabel.setBounds(0,0,248,59);
                //}}

                //{{REGISTER_LISTENERS
```

```
            SymWindow aSymWindow = new SymWindow();
            this.addWindowListener(aSymWindow);
            SymAction lSymAction = new SymAction();
            okButton.addActionListener(lSymAction);
            //}}
        }

        public void setVisible(boolean b)
        {
          if (b)
          {
            Rectangle bounds = (getParent()).getBounds();
            Dimension size = getSize();
            setLocation(bounds.x + (bounds.width - size.width)/2,
                        bounds.y + (bounds.height - size.height)/2);
          }

            super.setVisible(b);
        }

        public void addNotify()
        {
            // Record the size of the window prior to calling parents
addNotify.
            Dimension d = getSize();

            super.addNotify();

            if (fComponentsAdjusted)
                return;
            // Adjust components according to the insets
            Insets insets = getInsets();
            setSize(insets.left + insets.right + d.width, insets.top +
insets.bottom + d.height);
            Component components[] =
getContentPane().getComponents();
            for (int i = 0; i < components.length; i++)
            {
                Point p = components[i].getLocation();
                p.translate(insets.left, insets.top);
                components[i].setLocation(p);
            }
            fComponentsAdjusted = true;
        }

        // Used for addNotify check.
        boolean fComponentsAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JButton okButton = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel aboutLabel = new
com.sun.java.swing.JLabel();
        //}}

        class SymWindow extends java.awt.event.WindowAdapter
        {
            public void windowClosing(java.awt.event.WindowEvent
event)
            {
                Object object = event.getSource();
                if (object == JAboutDialog.this)
                    jAboutDialog_windowClosing(event);
            }
        }

        void jAboutDialog_windowClosing(java.awt.event.WindowEvent
event)
        {
            // to do: code goes here.

            jAboutDialog_windowClosing_Interaction1(event);
        }
```

```
void
jAboutDialog_windowClosing_Interaction1(java.awt.event.WindowEvent
event) {
        try {
                // JAboutDialog Hide the JAboutDialog
                this.setVisible(false);
        } catch (Exception e) {System.out.println(e);
        }
}

class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent
event)
        {
                Object object = event.getSource();
                if (object == okButton)
                        okButton_actionPerformed(event);
        }
}

void okButton_actionPerformed(java.awt.event.ActionEvent
event)
{
        // to do: code goes here.

        okButton_actionPerformed_Interaction1(event);
}

void
okButton_actionPerformed_Interaction1(java.awt.event.ActionEvent event)
{
        try {
                // JAboutDialog Hide the JAboutDialog
                this.setVisible(false);
        } catch (Exception e) {System.out.println(e);
        }
}
```

```
package JobSchedule;

import java.awt.*;
import com.sun.java.swing.*;
import java.util.*;


public class JDialog_jobskill extends com.sun.java.swing.JDialog
{
  Vector jobskill;

        public JDialog_jobskill(Frame parent)
        {
                super(parent);
```

```
// This code is automatically generated by Visual Cafe
when you add
// components to the visual environment. It instantiates
and initializes
// the components. To modify the code, only use code
syntax that matches
// what Visual Cafe can generate, or Visual Cafe may be
unable to back
// parse your Java file into its visual environment.
//{{INIT_CONTROLS
setTitle("Required Skills");
getContentPane().setLayout(null);
setSize(388,250);
setVisible(false);
JButton_delete_deleteperson.setText("Exit");

JButton_delete_deleteperson.setActionCommand("Delete");
getContentPane().add(JButton_delete_deleteperson);
JButton_delete_deleteperson.setBounds(146,204,96,24);

JLabel2.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
JLabel2.setText("Required Skills");
getContentPane().add(JLabel2);
JLabel2.setFont(new Font("Dialog", Font.BOLD, 15));
JLabel2.setBounds(110,24,168,24);
getContentPane().add(JLabel3);
JLabel3.setBounds(74,48,240,24);
JScrollPane1.setOpaque(true);
getContentPane().add(JScrollPane1);
JScrollPane1.setBounds(62,72,264,108);
JScrollPane1.getViewport().add(JTextArea1);
JTextArea1.setBounds(0,0,261,105);
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
JButton_delete_deleteperson.addActionListener(lSymAction);
//}}
}

public JDialog_jobskill()
{
this((Frame)null);
}
public JDialog_jobskill(Vector v)
{
this((Frame)null);
jobskill=new Vector();
jobskill=v;
int n=jobskill.size();
this.JTextArea1.setText("");
this.JTextArea1.append("Skill ID"+"\t"+"Skill
Name"+"\t"+"Skill Level"+"\n");
this.JTextArea1.append("\n");
for(int i=0; i<jobskill.size(); i++){
String s=(String) jobskill.elementAt(i);

StringTokenizer st = new StringTokenizer(s,":");
st.hasMoreTokens();
String number=new String(st.nextToken());
String name=st.nextToken();
String level=st.nextToken();

this.JTextArea1.append(number+"\t"+name+"\t"+level+"\n");

System.out.println("job is: "+number+name+level);
}

}

public JDialog_jobskill(String sTitle)
```

```
{
        this();
        setTitle(sTitle);
}

public void setVisible(boolean b)
{
        if (b)
                setLocation(50, 50);
        super.setVisible(b);
}

static public void main(String args[])
{
        (new JDialog_jobskill()).setVisible(true);
}

public void addNotify()
{
        // Record the size of the window prior to calling parents
addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
                return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
}

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
com.sun.java.swing.JButton JButton_delete_deleteperson = new
com.sun.java.swing.JButton();
com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
com.sun.java.swing.JScrollPane JScrollPane1 = new
com.sun.java.swing.JScrollPane();
com.sun.java.swing.JTextArea JTextArea1 = new
com.sun.java.swing.JTextArea();
//}}


class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent
event)
        {
                Object object = event.getSource();
                if (object == JButton_delete_deleteperson)

JButtonDeleteDeleteperson_actionPerformed(event);

        }
}

void
JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent
event)
{
        // to do: code goes here.
        this.setVisible(false);

}//end performence
}
```

```
package JobSchedule;

import java.awt.*;
import com.sun.java.swing.*;
import java.util.*;


public class JDialog_message extends com.sun.java.swing.JDialog
{

  public double w;
  public Weight weight;
  JFrame_manage p;
  int n;
        public JDialog_message(Frame parent)
        {
                super(parent);
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Error");
                getContentPane().setLayout(null);
                setSize(311,122);
                setVisible(false);
                JButton_delete_deleteperson.setText("Exit");

        JButton_delete_deleteperson.setActionCommand("Delete");
                getContentPane().add(JButton_delete_deleteperson);
                JButton_delete_deleteperson.setBounds(111,84,88,24);

        JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                JLabel3.setText("No job scheduled");
                getContentPane().add(JLabel3);
                JLabel3.setBounds(72,24,167,24);
                //}}

                //{{REGISTER_LISTENERS
                SymAction lSymAction = new SymAction();

        JButton_delete_deleteperson.addActionListener(lSymAction);
                //}}
        }

        public JDialog_message()
        {
                this((Frame)null);
        }


        public JDialog_message(String sTitle)
        {
                this();
                setTitle(sTitle);
        }

        public void setVisible(boolean b)
        {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
```

```
        }

        static public void main(String args[])
        {
                (new JDialog_message()).setVisible(true);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets
                Insets insets = getInsets();
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JButton JButton_delete_deleteperson = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == JButton_delete_deleteperson)

        JButtonDeleteDeleteperson_actionPerformed(event);

                }
        }

        void
JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent
event)
        {
                // to do: code goes here.
                this.setVisible(false);
    this.dispose();


        }
}
```

```
package JobSchedule;

import java.awt.*;
import com.sun.java.swing.*;
import java.util.*;

public class JDialog_message1 extends com.sun.java.swing.JDialog
{
```

```
public double w;
public Weight weight;
JFrame_manage p;
int n;
        public JDialog_message1(Frame parent)
        {
                super(parent);
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Error");
                getContentPane().setLayout(null);
                setSize(311,122);
                setVisible(false);
                JButton_delete_deleteperson.setText("Exit");

        JButton_delete_deleteperson.setActionCommand("Delete");
                getContentPane().add(JButton_delete_deleteperson);
                JButton_delete_deleteperson.setBounds(111,84,88,24);

        JLabel3.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                JLabel3.setText("No job or stakeholder assigned");
                getContentPane().add(JLabel3);
                JLabel3.setBounds(36,24,239,36);
                //}}

                //{{REGISTER_LISTENERS
                SymAction lSymAction = new SymAction();
        JButton_delete_deleteperson.addActionListener(lSymAction);
                //}}
        }

        public JDialog_message1()
        {
                this((Frame)null);
        }


        public JDialog_message1(String sTitle)
        {
                this();
                setTitle(sTitle);
        }

        public void setVisible(boolean b)
        {
                if (b)
                        setLocation(50, 50);
                super.setVisible(b);
        }

        static public void main(String args[])
        {
                (new JDialog_message1()).setVisible(true);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();
```

```
		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets
		Insets insets = getInsets();
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JButton JButton_delete_deleteperson = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
	//}}


	class SymAction implements java.awt.event.ActionListener
	{
		public void actionPerformed(java.awt.event.ActionEvent
event)
		{
			Object object = event.getSource();
			if (object == JButton_delete_deleteperson)

	JButtonDeleteDeleteperson_actionPerformed(event);

		}
	}

	void
JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent
event)
	{
		// to do: code goes here.
		this.setVisible(false);
	this.dispose();


	}
}
```

```
package JobSchedule;

import java.awt.*;
import com.sun.java.swing.*;


public class JDialog_weight extends com.sun.java.swing.JDialog
{
	public JDialog_weight(Frame parentFrame)
	{
		super(parentFrame);
		// This code is automatically generated by Visual Cafe
when you add
		// components to the visual environment. It instantiates
and initializes
```

```
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("JFC Application - About");
                setModal(true);
                getContentPane().setLayout(new GridBagLayout());
                setSize(416,162);
                setVisible(false);
                //}}

                //{{REGISTER_LISTENERS
                SymWindow aSymWindow = new SymWindow();
                this.addWindowListener(aSymWindow);
                SymAction lSymAction = new SymAction();
                //}}
        }

        public void setVisible(boolean b)
        {
          if (b)
          {
                Rectangle bounds = (getParent()).getBounds();
                Dimension size = getSize();
                setLocation(bounds.x + (bounds.width - size.width)/2,
                        bounds.y + (bounds.height - size.height)/2);
          }

                super.setVisible(b);
        }

        public void addNotify()
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension d = getSize();

                super.addNotify();

                if (fComponentsAdjusted)
                        return;
                // Adjust components according to the insets
                Insets insets = getInsets();
                setSize(insets.left + insets.right + d.width, insets.top +
insets.bottom + d.height);
                Component components[] =
getContentPane().getComponents();
                for (int i = 0; i < components.length; i++)
                {
                        Point p = components[i].getLocation();
                        p.translate(insets.left, insets.top);
                        components[i].setLocation(p);
                }
                fComponentsAdjusted = true;
        }

        // Used for addNotify check.
        boolean fComponentsAdjusted = false;

        //{{DECLARE_CONTROLS
        //}}

        class SymWindow extends java.awt.event.WindowAdapter
        {
                public void windowClosing(java.awt.event.WindowEvent
event)
                {
                        Object object = event.getSource();
                        if (object == JDialog_weight.this)
                                jAboutDialog_windowClosing(event);
                }
        }
```

```
        void jAboutDialog_windowClosing(java.awt.event.WindowEvent
event)
        {
                // to do: code goes here.

                jAboutDialog_windowClosing_Interaction1(event);
        }

        void
jAboutDialog_windowClosing_Interaction1(java.awt.event.WindowEvent
event) {
                try {
                        // JAboutDialog Hide the JAboutDialog
                        this.setVisible(false);
                } catch (Exception e) {System.out.println(e);
                }
        }

        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                }
        }
}
```

```
package JobSchedule;

import java.awt.*;
import com.sun.java.swing.*;
import java.util.*;

public class JDialog_weit extends com.sun.java.swing.JDialog
{

   public double w;
   public Weight weight;
   JFrame_manage p;
   int n;
        public JDialog_weit(Frame parent)
        {
                super(parent);
```

```
// This code is automatically generated by Visual Cafe
when you add
// components to the visual environment. It instantiates
and initializes
// the components. To modify the code, only use code
syntax that matches
// what Visual Cafe can generate, or Visual Cafe may be
unable to back
// parse your Java file into its visual environment.
//{{INIT_CONTROLS
setTitle("Weight");
getContentPane().setLayout(null);
setSize(388,171);
setVisible(false);
JButton_delete_deleteperson.setText("Done");

JButton_delete_deleteperson.setActionCommand("Delete");
getContentPane().add(JButton_delete_deleteperson);
JButton_delete_deleteperson.setBounds(156,132,76,24);

JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
JLabel1.setText("Weight: ");
getContentPane().add(JLabel1);
JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
JLabel1.setBounds(36,24,132,24);
JLabel2.setText("(from 0.0 to 1.0)");
getContentPane().add(JLabel2);
JLabel2.setBounds(144,24,168,24);
getContentPane().add(JTextField1);
JTextField1.setBounds(96,84,192,24);
getContentPane().add(JLabel3);
JLabel3.setBounds(96,48,192,24);
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
JButton_delete_deleteperson.addActionListener(lSymAction);
//}}
}

public JDialog_weit()
{
this((Frame)null);
}

public JDialog_weit(JFrame_manage p1, Weight weight1, int n1)
{
this((Frame)null);
weight=weight1;
p=p1;
n=n1;
}

public JDialog_weit(String sTitle)
{
this();
setTitle(sTitle);
}

public void setVisible(boolean b)
{
if (b)
setLocation(50, 50);
super.setVisible(b);
}

static public void main(String args[])
{
(new JDialog_weit()).setVisible(true);
}

public void addNotify()
```

```
        {
                // Record the size of the window prior to calling parents
addNotify.
                Dimension size = getSize();

                super.addNotify();

                if (frameSizeAdjusted)
                        return;
                frameSizeAdjusted = true;

                // Adjust size of frame according to the insets
                Insets insets = getInsets();
                setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height);
        }

        // Used by addNotify
        boolean frameSizeAdjusted = false;

        //{{DECLARE_CONTROLS
        com.sun.java.swing.JButton JButton_delete_deleteperson = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField JTextField1 = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == JButton_delete_deleteperson)

        JButtonDeleteDeleteperson_actionPerformed(event);

                }
        }

        void
JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent
event)
        {
                // to do: code goes here.
                Double d=new
Double(((String)this.JTextField1.getText()).trim());
                w=d.doubleValue();
                //int
i=Integer.parseInt(((String)this.JTextField1.getText()).trim());
                if(0.0<=w&&w<=1.0){
                  weight.put_w(w);
                  if(n==1){
                    p.sortbyD_E();
                    p.datavalu();
                  }
                  else{
                     p.sortbyD_S();
                     p.datavalu();
                  }
                  this.setVisible(false);
        dispose();
        this.dispose();
                }
                else{
                  this.JLabel3.setText("Invalid Value");
```

```
            }

        }
}
```



```
package JobSchedule;

import Cases.*;
import java.awt.*;
import com.sun.java.swing.*;
import java.util.*;

public class JFrame_assignjob extends com.sun.java.swing.JFrame
{
   Vector  person_queue,job_queue,work_queue;
        Vector   personskill_queue,jobskill_queue;
        Vector  result;
        int index, grade, step, n, year, month, date;
        String jobname;

        StepContent job;
        Vector p_q, pk_q, w_q, job_pool;
        CasesFrame parent;

        public JFrame_assignjob()
        {
                result=new Vector();
                step=1;
                n=0;
                // This code is automatically generated by Visual Cafe
when you add
                // components to the visual environment. It instantiates
and initializes
                // the components. To modify the code, only use code
syntax that matches
                // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                // parse your Java file into its visual environment.
                //{{INIT_CONTROLS
                setTitle("Job Assignment");
                getContentPane().setLayout(null);
                setSize(492,532);
                setVisible(false);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                JLabel1.setText("Job Assignment");
                getContentPane().add(JLabel1);
                JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
                JLabel1.setBounds(150,12,192,24);

        JButton1.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.LEFT);
                JButton1.setText("        1. Filter by Security Level");
                JButton1.setActionCommand("by priority");
                getContentPane().add(JButton1);
                JButton1.setBounds(168,264,276,24);

        JButton2.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.LEFT);
```

```
JButton2.setText("          2. Filter by Required Skills");
JButton2.setActionCommand("by skills");
getContentPane().add(JButton2);
JButton2.setBounds(168,300,276,24);

JButton3.setHorizontalAlignment(com.sun.java.swing.SwingConst
ants.LEFT);
JButton3.setText("          3. Assign this Job");
JButton3.setActionCommand("assign the job");
getContentPane().add(JButton3);
JButton3.setBounds(168,336,276,24);
JButton4.setText("Exit");
JButton4.setActionCommand("Save and Exit");
getContentPane().add(JButton4);
JButton4.setBounds(72,492,372,24);
JScrollPane1.setOpaque(true);
getContentPane().add(JScrollPane1);
JScrollPane1.setBounds(72,372,372,108);
JScrollPane1.getViewport().add(JTextArea2);
JTextArea2.setBounds(0,0,369,105);
JLabel2.setText("Security Level");
getContentPane().add(JLabel2);
JLabel2.setBounds(24,108,132,24);
JLabel7.setText("Job ID");
getContentPane().add(JLabel7);
JLabel7.setBounds(24,72,84,28);
JLabel8.setText("Deadline");
getContentPane().add(JLabel8);
JLabel8.setBounds(24,144,108,28);
JLabel9.setText("Estimated Duration");
getContentPane().add(JLabel9);
JLabel9.setBounds(24,180,120,24);
getContentPane().add(JTextField1);
JTextField1.setBounds(168,180,276,28);
getContentPane().add(JTextField2);
JTextField2.setBounds(168,144,276,28);
getContentPane().add(JTextField3);
JTextField3.setBounds(168,108,276,28);
getContentPane().add(JTextField4);
JTextField4.setBounds(168,72,276,28);
JLabel3.setText("Required Skills");
getContentPane().add(JLabel3);
JLabel3.setBounds(24,216,120,24);
JButton5.setText("Required Skills");
JButton5.setActionCommand("Job skill");
getContentPane().add(JButton5);
JButton5.setBounds(168,216,276,24);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
JButton4.addActionListener(lSymAction);
JButton1.addActionListener(lSymAction);
JButton2.addActionListener(lSymAction);
JButton3.addActionListener(lSymAction);
JButton5.addActionListener(lSymAction);
//}}

//{{REGISTER_LISTENERS
//SymAction lSymAction = new SymAction();
//JButton3.addActionListener(lSymAction);
//}}

}

public JFrame_assignjob(CasesFrame parent1,Vector
person_queue1,Vector job_queue1, Vector job_pool1)
{
this();
```

```
		parent=parent1;
		person_queue=person_queue1; job_queue=job_queue1;
		job_pool=job_pool1;
		this.job=(StepContent) job_queue.firstElement();
		String name=job.getStepName();
		this.jobname=name;


		String sk1=new String();
		this.JTextField4.setText(job.getStepName());

	this.JTextField3.setText(sk1.valueOf(job.getSecurityLevel()));
		this.JTextField2.setText(job.getDeadline());
		this.JTextField1.setText(sk1.valueOf(job.getDuration()));
		//System.out.println("in assignment constructor person
size:"+person_queue.size());
	//cleanperson_job();
	}

	public JFrame_assignjob(String sTitle)
	{
		this();
		setTitle(sTitle);

	}

	public void setVisible(boolean b)
	{
		if (b)
			setLocation(50, 50);
		super.setVisible(b);
	}

	static public void main(String args[])
	{
		(new JFrame_assignjob()).setVisible(true);
	}

	public void addNotify()
	{
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JButton JButton1 = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JButton JButton2 = new
com.sun.java.swing.JButton();
```

```
        com.sun.java.swing.JButton JButton3 = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JButton JButton4 = new
com.sun.java.swing.JButton();
        com.sun.java.swing.JScrollPane JScrollPane1 = new
com.sun.java.swing.JScrollPane();
        com.sun.java.swing.JTextArea JTextArea2 = new
com.sun.java.swing.JTextArea();
        com.sun.java.swing.JLabel JLabel2 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel7 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel8 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JLabel JLabel9 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JTextField JTextField1 = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField JTextField2 = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField JTextField3 = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JTextField JTextField4 = new
com.sun.java.swing.JTextField();
        com.sun.java.swing.JLabel JLabel3 = new
com.sun.java.swing.JLabel();
        com.sun.java.swing.JButton JButton5 = new
com.sun.java.swing.JButton();
        //}}

        //{{DECLARE_MENUS
        //}}


        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent
event)
                {
                        Object object = event.getSource();
                        if (object == JButton4)
                                JButton4_actionPerformed(event);
                        else if (object == JButton1)
                                JButton1_actionPerformed(event);
                        else if (object == JButton2)
                                JButton2_actionPerformed(event);
                        else if (object == JButton3)
                                JButton3_actionPerformed(event);
                        else if (object == JButton5)
                                JButton5_actionPerformed(event);
                }
        }

        void JButton4_actionPerformed(java.awt.event.ActionEvent event)
        {
                // to do: code goes here.
                this.setVisible(false);

        }

        void Filterbysecurity(){

          // p_q=(Vector) person_queue.clone();
           p_q=new Vector();
           int security=job.getSecurityLevel();
           System.out.println("job security is "+security);
           for(int i=0; i<person_queue.size(); i++){
              Personnel person=(Personnel) person_queue.elementAt(i);
              System.out.println("the person security is
"+person.getSecurityLevel());
              if(security<=person.getSecurityLevel()){
                 System.out.println("the security is
"+person.getSecurityLevel()
```

```
        +"persion id is "+person.getID());

        p_q.addElement(person);

      }
    }//end for
    System.out.println("in filterbysecurity: p_q size is "+p_q.size());
  }//end Filter


  int getmonthbyint(String month){
    if(month.equals("January")){
      return 1;
    }
    else{
      if(month.equals("February")){
        return 2;
      }
      else{
        if(month.equals("March")){
          return 3;
        }
        else{
          if(month.equals("April")){
            return 4;
          }
          else{
            if(month.equals("May")){
              return 5;
            }
            else{
              if(month.equals("June")){
                return 6;
              }
              else{
                if(month.equals("July")){
                  return 7;
                }
                else{
                  if(month.equals("August")){
                    return 8;
                  }
                  else{
                    if(month.equals("September")){
                      return 9;
                    }
                    else{
                      if(month.equals("October")){
                        return 10;
                      }
                      else{
                        if(month.equals("November")){
                          return 11;
                        }
                        else{
                          if(month.equals("December")){
                            return 12;
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }//end if-else
    return 0;
  }

  String setmonth(int month){
    if(month==1){
```

```
    String s="January";
    return s;
  }
  else{
    if(month==2){
      String s="February";
      return s;
    }
    else{
      if(month==3){
        String s="March";
        return s;
      }
      else{
        if(month==4){
          String s="April";
          return s;
        }
        else{
          if(month==5){
            String s="May";
            return s;
          }
          else{
            if(month==6){
              String s="June";
              return s;
            }
            else{
              if(month==7){
                String s="July";
                return s;
              }
              else{
                if(month==8){
                  String s="August";
                  return s;
                }
                else{
                  if(month==9){
                    String s="September";
                    return s;
                  }
                  else{
                    if(month==10){
                      String s="October";
                      return s;
                    }
                    else{
                      if(month==11){
                        String s="November";
                        return s;
                      }
                      else{
                        if(month==12){
                          String s="December";
                          return s;
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  return null;

}//end

void getdate(String date){
```

```
        this.year=0; this.month=0; this.date=0;
        System.out.println("in getdate");
        StringTokenizer st = new StringTokenizer(date,",");
        st.hasMoreTokens();
            String s=new String(st.nextToken());
            String s2=new String(st.nextToken());

            StringTokenizer st1 = new StringTokenizer(s," ");
            st1.hasMoreTokens();
            String s1=st1.nextToken();
            this.month=getmonthbyint(s1);
            try{
              this.date=Integer.parseInt(st1.nextToken());
              //this.year=Integer.parseInt(s);
    }
    catch( Exception e){}

      }

      void Filterbyskills(){
        try{
          for(int i=0; i<this.p_q.size(); i++){
            this.grade=0;
            Personnel p=(Personnel) p_q.elementAt(i);
            Vector v=(Vector) p.getSkill();

            for(int j=0; j<v.size(); j++){
              String s1=(String)v.elementAt(j);

              StringTokenizer st = new StringTokenizer(s1,":");
              String s2=new String(st.nextToken());
                int number1=Integer.parseInt(s2.trim());
                String name1=((String)st.nextToken()).trim();
                int
level1=Integer.parseInt(((String)st.nextToken()).trim());
                System.out.println("take each the person's skill");
                System.out.println("the skill number, name,
level:");
                System.out.println(number1+name1+level1);
                for(int k=0; k<this.job.getSkill().size(); k++){
                  String
s3=(String)this.job.getSkill().elementAt(k);
                  System.out.println("the job skill is "+s3);
                  StringTokenizer st2 = new
StringTokenizer(s3,":");
                  String s4=((String)(st2.nextToken())).trim();
                  int number2=Integer.parseInt(s4.trim());
                  String name2=((String)st2.nextToken()).trim();
                  int
level2=Integer.parseInt(((String)st2.nextToken()).trim());
                  System.out.println("the job's number, name,
level:");
                  System.out.println(number2+" "+name2+"
"+level2);
                  if(number1==number2&&level1>=level2){
                    grade++;
                    System.out.println("number1,
level1"+number1+level1);
                    System.out.println("number2,
level2"+number2+level2);
                  }
                }//end for(k)

            }//end for(j)
            int id=Integer.parseInt(p.getID());
            Result r=new Result(id,grade);
            result.addElement(r);

          }//end for(i)
          sortresult();
    }
    catch( Exception e){}
      }
```

```
    void sortresult(){

      Result min, tmp;
      for(int i=0; i<result.size(); i++){
        min=(Result) result.elementAt(i);
        for(int j=i+1; j<result.size(); j++){
          Result r1=(Result) result.elementAt(j);

          if(min.get_grade()<r1.get_grade()){
            tmp=min;
            min=r1;
            result.setElementAt(tmp, j);
          }//end if()
        }//end for(j)

        result.setElementAt(min, i);
      }//end for(//i)


    }//end sort

    Personnel seachbyid(int id){
      try{
        for(int j=0; j<p_q.size(); j++){
          Personnel p=(Personnel) p_q.elementAt(j);
          int id1=Integer.parseInt(p.getID());
          if(id==id1){
            return p;
          }
        }
}
catch( Exception e){}
      return null;
    }//end seach

    void display1(){
```

```
        this.JTextArea2.setText("");
        System.out.println("p_q siz is "+p_q.size());
        this.JTextArea2.append("Stakeholder ID"+"\t"+"Security
Level"+"\n");
            String sk=new String();
            for(int i=0; i<p_q.size(); i++){
              Personnel p=(Personnel) p_q.elementAt(i);
          this.JTextArea2.append(sk.valueOf(p.getID()+"\t"));

this.JTextArea2.append(sk.valueOf(p.getSecurityLevel()+"\n"));

        }

      }

      void display2(){

        this.JTextArea2.setText("");
        this.JTextArea2.append("Stakeholder ID"+"\t"+"Match Number
of Required Skills"+"\n");
        String s=new String();

       for(int i=0; i<result.size(); i++){
              Result r=(Result) result.elementAt(i);
          this.JTextArea2.append(s.valueOf(r.get_id()+"\t"));
          this.JTextArea2.append(s.valueOf(r.get_grade()+"\n"));

        }

      }

      void display3(int pid){
        Personnel p=seachbyid(pid);
        Vector v=p.getMajorJobs();

        StepContent step1=(StepContent) v.elementAt(0);
```

```
            StepContent step2=null;
            if(v.size()==2){
              step2=(StepContent) v.elementAt(1);
            }

            this.JTextArea2.setText("");
            this.JTextArea2.append("Stakeholder ID"+"\t"+"Job
ID"+"\t"+"Earliest Start Time"
                                +"\t"+"Estimated Duration"+"\n");
            this.JTextArea2.append("");
            this.JTextArea2.append(p.getID()+"\t"+
step1.getStepName()+"\t"+step1.getStartTime()+"\t"+step1.getDuration()
+"\n");
            if(step2!=null){

this.JTextArea2.append(p.getID()+"\t"+step2.getStepName()+"\t"+step2.get
StartTime()+"\t"+step2.getDuration()+"\n");
            }

        }


        void JButton1_actionPerformed(java.awt.event.ActionEvent event)
        {
                // to do: code goes here.
                if(step==1){
                  step++;
                  Filterbysecurity();
                  display1();
                }


        }

        void JButton2_actionPerformed(java.awt.event.ActionEvent event)
        {
                // to do: code goes here.
                if(step==2){
                  step++;
                  Filterbyskills();
                  display2();
                }

            symantec.itools.awt.util.Calendar c=new
symantec.itools.awt.util.Calendar();
            String s=c.getDate();
            System.out.println("today is "+s);
            getdate(s);
            System.out.println("year, month, date are:");
            System.out.println(year+" "+ month+" "+ date);
        }

        void JButton3_actionPerformed(java.awt.event.ActionEvent event)
        {
                // to do: code goes here.
                if(step==3||step==4){
                  step++;
                  int  num=0;
                  if(n==0){

                    num=assing_Performed();
                    if(num==1){
                      System.out.println("job jobname is "+jobname);
                      System.out.println("befor save job_pool size=
"+job_pool.size());
                      for(int i=0; i<job_pool.size(); i++){
                        StepContent
step=(StepContent)job_pool.elementAt(i);
                    System.out.println("before save the step status and
name  "+step.getStatus()+"  "+step.getStepName());
                  }
                      parent.savePersonnelVector(person_queue);
                      parent.saveStepContentVector(job_pool);
                    }
```

```
                n++;
              }
            }
        }//end

        int assing_Performed(){

          symantec.itools.awt.util.Calendar c=new
symantec.itools.awt.util.Calendar();
          int duration, personid;
          String jobname;
          Result r;

          duration=this.job.getDuration();
          jobname=this.job.getStepName();

          for(int i=0; i<result.size(); i++){

            personid=((Result) result.elementAt(i)).get_id();
            Personnel p=seachbyid(personid);
            Vector v1=p.getMajorJobs();
            Vector v2=p.getMinorJobs();

            String s=c.getDate();
            System.out.println("clear person");

            if(v1.size()<2){
             //set the job's status
             job.setStatus("Assigned");
             //set the job's starttime
             job.setStartTime(s);
             //save job
             setjob_pool(jobname, this.job);
             //set the person's morja job
             ((StepContent)this.job).setRealStartTime(c.getDate());
             v1.addElement(this.job);
             p.setMajorJobs(v1);
             //save the person
             setperson_queue(personid, p);

             //remove the job
             job_queue.removeElement(this.job);

             display3(personid);

             return 1;
            }//end if
            else{

             //set the person's minor job
             ((StepContent)this.job).setRealStartTime(c.getDate());
             v2.addElement(this.job);
             //set Mina job
             p.setMinorJobs(v2);
             //save the person
             setperson_queue(personid, p);
             return 1;
            }

          }//end for

          return 0;

        }//end assing

        void setperson_queue(int personid, Personnel p1){
          try{
            for(int i=0; i<person_queue.size(); i++){
              Personnel p=(Personnel) person_queue.elementAt(i);
              if(personid==Integer.parseInt(p.getID())){
                person_queue.setElementAt(p1, i);
              }
            }//end for_loop
          }
```

```
catch(Exception e){}
    }
    void setjob_pool(String jobname, StepContent step1){
      for(int i=0; i<job_pool.size(); i++ ){
        StepContent step=(StepContent) job_pool.elementAt(i);
        if(jobname.equals(step.getStepName())){
          job_pool.setElementAt(step1, i);
        }
      }
    }
    void cleanperson_job(){
      try{
        for(int i=0; i<person_queue.size(); i++){
          Personnel p=(Personnel) person_queue.elementAt(i);
          Vector v1=p.getMajorJobs();
          Vector v2=p.getMinorJobs();


          v1.removeAllElements();
          p.setMajorJobs(v1);
          v2.removeAllElements();
          p.setMinorJobs(v2);
          int personid=Integer.parseInt(p.getID());
          setperson_queue(personid, p);
        }
        parent.savePersonnelVector(person_queue);
}
catch( Exception e){}
    }
    void JButton5_actionPerformed(java.awt.event.ActionEvent event)
    {
            // to do: code goes here.
            Vector v=(Vector) job.getSkill();
            (new JDialog_jobskill(v)).setVisible(true);


    }
}

package JobSchedule;

/*
        A basic implementation of the JFrame class.
*/
import Cases.*;
import java.util.*;
import java.util.Date;
import java.awt.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.DefaultTableModel;
import java.lang.Double;
import com.symantec.itools.swing.JButtonGroupPanel;

public class JFrame_manage extends com.sun.java.swing.JFrame
{
  Vector job_pool, job_queue1,job_queue2, job_queue;
  double w;
```

```
Weight weight;
//TableModel mytable;
int year=0, month=0, date=0;

        public JFrame_manage()
        {
           weight=new Weight();
           job_queue1=new Vector();
           job_queue2=new Vector();

           //initai weight value
           w=0.5;
           System.out.println("in manage2");
                 // This code is automatically generated by Visual Cafe
when you add
                 // components to the visual environment. It instantiates
and initializes
                 // the components. To modify the code, only use code
syntax that matches
                 // what Visual Cafe can generate, or Visual Cafe may be
unable to back
                 // parse your Java file into its visual environment.
                 //{{INIT_CONTROLS
                 setTitle("Job Scheduling");
                 getContentPane().setLayout(null);
                 setSize(596,415);
                 setVisible(false);

        JLabel1.setHorizontalAlignment(com.sun.java.swing.SwingConsta
nts.CENTER);
                 JLabel1.setText("Job Scheduling Policy");
                 getContentPane().add(JLabel1);
                 JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
                 JLabel1.setBounds(190,24,216,36);
                 JButton5.setText("Exit");
                 JButton5.setActionCommand("Save and Exit");
                 getContentPane().add(JButton5);
                 JButton5.setBounds(256,372,92,24);
                 getContentPane().add(JLabel6);
                 JLabel6.setBounds(24,72,108,24);
                 JScrollPane2.setOpaque(true);
                 getContentPane().add(JScrollPane2);
                 JScrollPane2.setBounds(60,132,480,216);
                 JScrollPane2.getViewport().add(JTable1);
                 JTable1.setBounds(0,0,477,213);
                 getContentPane().add(JComboBox1);
                 JComboBox1.setBounds(144,72,396,36);
                 //}}

                 //{{INIT_MENUS
                 //}}


                 //{{REGISTER_LISTENERS
                 SymAction lSymAction = new SymAction();
                 JButton5.addActionListener(lSymAction);
                 SymFocus aSymFocus = new SymFocus();
                 SymItem lSymItem = new SymItem();
                 JComboBox1.addItemListener(lSymItem);
                 JScrollPane2.addFocusListener(aSymFocus);
                 JTable1.addFocusListener(aSymFocus);
                 //}}
                 JComboBox1.addItem("Select a Policy");
                 JComboBox1.addItem("High Priority First");
                 JComboBox1.addItem("Minimum Deadline First
(Min_D)");
                 JComboBox1.addItem("Minimum Estimated Duration
First (Min_E)");
                 JComboBox1.addItem("Minimum Earliest Start Time
First (Min_S)");
                 JComboBox1.addItem("Minimum Laxity First
(Min_L)");
                 JComboBox1.addItem("Min_D*w + Min_E*(1-w)");
```

```
		JComboBox1.addItem("Min_D*w + Min_S*(1-w)");

	}

	public JFrame_manage(Vector job_queue1)
	{
		this();
		System.out.println("this is in manage");
		job_queue=job_queue1;
		System.out.println("job_queue="+job_queue.size());
		datavalu();
		//display();
	}

	public JFrame_manage(String sTitle)
	{
		this();
		setTitle(sTitle);
	}


	public void setVisible(boolean b)
	{
		if (b)
			setLocation(50, 50);
		super.setVisible(b);
	}

	static public void main(String args[])
	{
		(new JFrame_manage()).setVisible(true);
	}

	public void addNotify()
	{
		// Record the size of the window prior to calling parents
addNotify.
		Dimension size = getSize();

		super.addNotify();

		if (frameSizeAdjusted)
			return;
		frameSizeAdjusted = true;

		// Adjust size of frame according to the insets and menu
bar
		Insets insets = getInsets();
		com.sun.java.swing.JMenuBar menuBar =
getRootPane().getJMenuBar();
		int menuBarHeight = 0;
		if (menuBar != null)
			menuBarHeight =
menuBar.getPreferredSize().height;
		setSize(insets.left + insets.right + size.width, insets.top +
insets.bottom + size.height + menuBarHeight);
	}

	// Used by addNotify
	boolean frameSizeAdjusted = false;

	//{{DECLARE_CONTROLS
	com.sun.java.swing.JLabel JLabel1 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JButton JButton5 = new
com.sun.java.swing.JButton();
	com.sun.java.swing.JLabel JLabel6 = new
com.sun.java.swing.JLabel();
	com.sun.java.swing.JScrollPane JScrollPane2 = new
com.sun.java.swing.JScrollPane();
	com.sun.java.swing.JTable JTable1 = new
com.sun.java.swing.JTable();
	com.sun.java.swing.JComboBox JComboBox1 = new
com.sun.java.swing.JComboBox();
```

```
//}}

//{{DECLARE_MENUS
//}}

//creat Jtable
void datavalu(){

  DefaultTableModel mymodel;
  int size=job_queue.size();
  String datavalue[][]=new String[size][6];
String header[]=new String[6];

  header[0]="Job ID";
  header[1]="Priority";
  header[2]="Deadline";
  header[3]="Estimated Duration";
  header[4]="Earliest Starttime";
  header[5]="Laxity";

  for(int i=0; i<size; i++){
    String s=new String();
      StepContent step=(StepContent)job_queue.elementAt(i);
      datavalue[i][0]=step.getStepName();
      datavalue[i][1]=s.valueOf(step.getPriority());
      datavalue[i][2]=step.getDeadline();
      datavalue[i][3]=s.valueOf(step.getDuration());
      datavalue[i][4]=step.getStartTime();

      //caculation laxity
      int laxity, y1,y2,m1,m2,d1,d2;

      String deadline=step.getDeadline();
      getdate(deadline);
      y1=0;  m1=0;  d1=0;
      y1=this.year;  m1=this.month;  d1=this.date;

      String starttime=step.getStartTime();
      getdate(starttime);
      y2=0;  m2=0;  d2=0;
      y2=this.year;  m2=this.month;  d2=this.date;
      laxity=((y1-y2)*360+(m1-m2)*30+(d1-d2))-
step.getDuration();

      datavalue[i][5]=s.valueOf(laxity);


  }
  mymodel=new DefaultTableModel(datavalue, header);
  JTable1.setModel(mymodel);
  System.out.println("after creat JTable");

}


int getmonthbystring(String month){
  if(month.equals("January")){
    return 1;
  }
  else{
    if(month.equals("February")){
      return 2;
    }
    else{
      if(month.equals("March")){
        return 3;
      }
      else{
        if(month.equals("April")){
          return 4;
        }
        else{
          if(month.equals("May")){
            return 5;
```

```
}
else{
  if(month.equals("June")){
    return 6;
  }
  else{
    if(month.equals("July")){
      return 7;
    }
    else{
      if(month.equals("August")){
        return 8;
      }
      else{
        if(month.equals("September")){
          return 9;
        }
        else{
          if(month.equals("October")){
            return 10;
          }
          else{
            if(month.equals("November")){
              return 11;
            }
            else{
              if(month.equals("December")){
                return 12;
              }
            }
          }
        }
      }
    }
  }
}
```

```
      }
    }
  }//end if-else
  return 0;
}

String getmonthbyint(int month){
  if(month==1){
    String s="January";
    return s;
  }
  else{
    if(month==2){
      String s="February";
      return s;
    }
    else{
      if(month==3){
        String s="March";
        return s;
      }
      else{
        if(month==4){
          String s="April";
          return s;
        }
        else{
          if(month==5){
            String s="May";
            return s;
          }
          else{
            if(month==6){
              String s="June";
              return s;
            }
            else{
```

```
            if(month==7){
              String s="July";
              return s;
            }
            else{
              if(month==8){
                String s="August";
                return s;
              }
              else{
                if(month==9){
                  String s="September";
                  return s;
                }
                else{
                  if(month==10){
                    String s="October";
                    return s;
                  }
                  else{
                    if(month==11){
                      String s="November";
                      return s;
                    }
                    else{
                      if(month==12){
                        String s="December";
                        return s;
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
```

```
        }
      }
    }
    return null;

  }//end

  void sortbypriority(){

    StepContent max, tmp;
    for(int i=0; i<job_queue.size(); i++){
      max=(StepContent) job_queue.elementAt(i);
      for(int j=i+1; j<job_queue.size(); j++){
        StepContent step1=(StepContent) job_queue.elementAt(j);
        if(max.getPriority()<step1.getPriority()){
          tmp=max;
          max=step1;
          job_queue.setElementAt(tmp, j);
        }//end if()
      }//end for(j)

      job_queue.setElementAt(max, i);
    }//end for(//i)

  }//end sort

  void sortbydeadline(){

  System.out.println("in sorbydeadline ");

    StepContent earlist, tmp;
    int y1=0, y2=0, m1=0, m2=0, d1=0, d2=0;
    for(int i=0; i<job_queue.size(); i++){
      earlist=(StepContent) job_queue.elementAt(i);

      String dead1=earlist.getDeadline();
```

```
      getdate(dead1);
      y1=0; m1=0; d1=0;
      y1=this.year;  m1=this.month;  d1=this.date;

      for(int j=i+1; j<job_queue.size(); j++){
         StepContent step1=(StepContent) job_queue.elementAt(j);
         String dead2=step1.getDeadline();
         getdate(dead2);
         y2=0; m2=0; d2=0;
         y2=this.year;  m2=this.month;  d2=this.date;

         int n=after(y1,m1,d1,y2,m2,d2);

         if(n==1){
            dead1=step1.getDeadline();
            getdate(dead1);
            y1=0; m1=0; d1=0;
            y1=this.year;  m1=this.month;  d1=this.date;
            tmp=earlist;
            earlist=step1;
            job_queue.setElementAt(tmp, j);
         }//end if()
      }//end for(j)

      job_queue.setElementAt(earlist, i);
   }//end for(//i)

}

int after(int y1, int m1, int d1, int y2, int m2, int d2){
   if(y1>y2){
      return 1;
   }
   else{
      if(y1==y2){
         if(m1>m2){
            return 1;
         }
         else{
            if(m1==m2){
               if(d1>d2){
                  return 1;
               }
            }
         }
      }
   }
   return 0;
}//end after

void getdate(String date){
   try{
      this.year=0; this.month=0; this.date=0;
      StringTokenizer st = new StringTokenizer(date," ");
      st.hasMoreTokens();
            String s=new String(st.nextToken());

            this.month=getmonthbystring(s);
            String d=st.nextToken();
            this.year=Integer.parseInt(st.nextToken());

            StringTokenizer st1 = new StringTokenizer(d,",");
      st1.hasMoreTokens();
            String d2=new String(st1.nextToken());
            this.date=Integer.parseInt(d2);
}
catch(Exception e){}
   }


   void sortbyduration(){

      StepContent min, tmp;
      int y1, y2, m1, m2, d1, d2;
```

```
    for(int i=0; i<job_queue.size(); i++){
      min=(StepContent) job_queue.elementAt(i);
      for(int j=i+1; j<job_queue.size(); j++){
        StepContent step1=(StepContent) job_queue.elementAt(j);
        if(min.getDuration()>step1.getDuration()){
          tmp=min;
          min=step1;
          job_queue.setElementAt(tmp, j);
        }//end if()
      }//end for(j)

      job_queue.setElementAt(min, i);
    }//end for(//i)

  }

  void sortbystarttime(){

    int y1, y2, m1, m2, d1, d2;
    StepContent ealist, tmp;
    for(int i=0; i<job_queue.size(); i++){
      ealist=(StepContent) job_queue.elementAt(i);
      String dead1=ealist.getStartTime();
      getdate(dead1);
      y1=this.year;  m1=this.month;  d1=this.date;

      for(int j=i+1; j<job_queue.size(); j++){
        StepContent step1=(StepContent) job_queue.elementAt(j);
        String dead2=step1.getStartTime();
        getdate(dead2);
         y2=0; m2=0; d2=0;
        y2=this.year;  m2=this.month;  d2=this.date;

        if(after(y1,m1,d1,y2,m2,d2)==1){
          dead1=step1.getStartTime();
          getdate(dead1);
          y1=this.year;  m1=this.month;  d1=this.date;
          tmp=ealist;
          ealist=step1;
          job_queue.setElementAt(tmp, j);
        }//end if()
      }//end for(j)

      job_queue.setElementAt(ealist, i);
    }//end for(//i)

  }//end sortbystarttime

  void sortbylaxity(){

    StepContent min, tmp;
    int laxity1,laxity2;
    int y1, y2, m1, m2, d1, d2;
    for(int i=0; i<job_queue.size(); i++){
      min=(StepContent) job_queue.elementAt(i);
      String deadline1=min.getDeadline();
      getdate(deadline1);
      y1=0;  m1=0;  d1=0;
      y1=this.year;  m1=this.month;  d1=this.date;

      String starttime1=min.getStartTime();
      getdate(starttime1);
      y2=0;  m2=0;  d2=0;
      y2=this.year;  m2=this.month;  d2=this.date;
      laxity1=((y1-y2)*360+(m1-m2)*30+(d1-d2))-
min.getDuration();

      for(int j=i+1; j<job_queue.size(); j++){
        StepContent step=(StepContent) job_queue.elementAt(j);
        String deadline2=step.getDeadline();
        getdate(deadline2);
        y1=0; m1=0; d1=0;
        y1=this.year;  m1=this.month;  d1=this.date;
        String starttime2=step.getStartTime();
```

```
                getdate(starttime2);
                y2=0; m2=0; d2=0;
                y2=this.year;  m2=this.month;  d2=this.date;
                laxity2=((y1-y2)*360+(m1-m2)*30+(d1-d2))-
step.getDuration();

                if(laxity1>laxity2){

                  tmp=min;
                  min=step;
                  job_queue.setElementAt(tmp, j);
                }//end if()
              }//end for(j)
              job_queue.setElementAt(min, i);
            }//end for(//i)

          }//end sortbylaxity()

          //sortbyMinD_MinE
          void sortbyD_E(){

            sortbydeadline();
            System.out.println("in D_E e="+w);
            w=weight.get_w();
            StepContent step=(StepContent) job_queue.lastElement();
            String s=step.getDeadline();
            getdate(s);
            int y=year;
            int m=month;
            int d=date;
            System.out.println("month= "+m+"  date= "+d);

            StepContent max, tmp;

            for(int i=0; i<job_queue.size(); i++){
              max=(StepContent) job_queue.elementAt(i);
              String deadline1=max.getDeadline();
              getdate(deadline1);
              int y_dead1=this.year;
              int m_dead1=this.month;
              int d_dead1=this.date;
              for(int j=i+1; j<job_queue.size(); j++){
                StepContent step1=(StepContent) job_queue.elementAt(j);
                String deadline2=step1.getDeadline();
                getdate(deadline2);
                int y_dead2=year;
                int m_dead2=month;
                int d_dead2=date;
                if((((y-y_dead1)*360+(m-m_dead1)*30+(d-d_dead1))*w
                  -max.getDuration()*(1-w))<
                  (((y-y_dead2)*360+(m-m_dead2)*30+(d-d_dead2))*w
                  -step1.getDuration()*(1-w)))
                {
                  tmp=max;
                  max=step1;
                  job_queue.setElementAt(tmp, j);

                }//end if()
              }//end for(j)


              job_queue.setElementAt(max, i);
            }//end for(//i)



          }

          void sortbyD_S(){

            sortbydeadline();
            System.out.println("in D_S e="+w);

            w=weight.get_w();
```

```
StepContent step=(StepContent) job_queue.lastElement();
//int y=job.get_deadline().getYear();
String s1=step.getDeadline();
getdate(s1);
int y_dead=year;
int m_dead=month;
int d_dead=date;

String s2=step.getStartTime();
getdate(s2);
int y_start=year;
int m_start=month;
int d_start=date;

StepContent max, tmp;
for(int i=0; i<job_queue.size(); i++){
   max=(StepContent) job_queue.elementAt(i);

   String s3=max.getDeadline();
   getdate(s3);
   int y_dead_max=year;
   int m_dead_max=month;
   int d_dead_max=date;

   String s4=max.getStartTime();
   getdate(s4);
   int y_start_max=year;
   int m_start_max=month;
   int d_start_max=date;

   for(int j=i+1; j<job_queue.size(); j++){
      StepContent step1=(StepContent) job_queue.elementAt(j);

      String s5=step1.getDeadline();
      getdate(s5);
      int y_dead_step=year;
      int m_dead_step=month;
      int d_dead_step=date;

      String s6=step1.getStartTime();
      getdate(s6);
      int y_start_step=year;
      int m_start_step=month;
      int d_start_step=date;

      if((((y_dead-y_dead_max)*365+(m_dead-
m_dead_max)*30+(d_dead-d_dead_max))*w
         +((y_start-y_start_max)*365+(m_start-
m_start_max)*30+(d_start-d_start_max))*(1-w))<
         (((y_dead-y_dead_step)*365+(m_dead-
m_dead_step)*30+(d_dead-d_dead_step))*w
         +((y_dead-y_start_step)*365+(m_dead-
m_start_step)*30+(d_dead-d_start_step))*(1-w)))
      {
         tmp=max;
         max=step1;
         job_queue.setElementAt(tmp, j);

      }//end if()
   }//end for(j)

   job_queue.setElementAt(max, i);
}//end for(//i)

}

class SymAction implements java.awt.event.ActionListener
{
      public void actionPerformed(java.awt.event.ActionEvent
event)
      {
            Object object = event.getSource();
            if (object == JButton5)
                  JButton5_actionPerformed(event);
```

```
        }
    }

    void JButton5_actionPerformed(java.awt.event.ActionEvent event)
    {
        // to do: code goes here.
        this.setVisible(false);

    }

    class SymFocus extends java.awt.event.FocusAdapter
    {
        public void focusGained(java.awt.event.FocusEvent
event)
        {
            Object object = event.getSource();
            if (object == JScrollPane2)
                JScrollPane2_focusGained(event);
            else if (object == JTable1)
                JTable1_focusGained(event);
        }
    }

    class SymItem implements java.awt.event.ItemListener
    {
        public void itemStateChanged(java.awt.event.ItemEvent
event)
        {
            Object object = event.getSource();
            if (object == JComboBox1)

JComboBox1_itemStateChanged(event);
```

```
            }
        }

        void JComboBox1_itemStateChanged(java.awt.event.ItemEvent
event)
        {
          System.out.println("in jombobox performence");

    if( event.getStateChange() == event.SELECTED ){
            // to do: code goes here.
            String str=(String) event.getItem();

            if(str.equals("High Priority First")){
              System.out.println("high priority first");
              //new Getdat();
              sortbypriority();
              datavalu();
            }
            else{
              if(str.equals("Minimum Deadline First (Min_D)")){
              System.out.println("deadline");
              sortbydeadline();
              datavalu();
              }
              else
                if(str.equals("Minimum Estimated Duration First
(Min_E)")){

                System.out.println("duration");
                sortbyduration();
                datavalu();
                }
                else{
                  if(str.equals("Minimum Earliest Start Time First
(Min_S)" )){

                  System.out.println("startime");
                  sortbystarttime();
                  datavalu();
```

```
                    }
                    else{
                      if(str.equals("Minimum Laxity First
(Min_L)")){
                      System.out.println("min_l");
                        sortbylaxity();
                        datavalu();
                        }
                        else{
                          if(str.equals("Min_D*w + Min_E*(1-
w)")){
                        System.out.println("D_E");
                        (new JDialog_weit(this, weight,
1)).setVisible(true);
                          //sortbyD_E();
                        }
                        else{
                          if(str.equals("Min_D*w + Min_S*(1-
w)")){
                          System.out.println("D_S");
                          (new JDialog_weit(this, weight,
2)).setVisible(true);
                          //sortbyD_S();

                          }
                        }
                          }
                    }
                  }
                }
            }

        }
        void JScrollPane2_focusGained(java.awt.event.FocusEvent event)
        {
            // to do: code goes here.
        }
        void JTable1_focusGained(java.awt.event.FocusEvent event)
        {
            // to do: code goes here.

        }
}
```

```
package JobSchedule;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import java.io.Serializable;


public class Predecessor implements Serializable{
   private Vector predec;
   public Predecessor(){predec=new Vector();}
   public Vector get_predec(){ return predec;}
}
```

```
package JobSchedule;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;
import com.sun.java.swing.*;

public class Result implements Serializable{
   private int personid;
   private int grade;
   public Result(int id, int grade1){
      personid=id;
      grade=grade1;
   }

   public Result() { }

   public int get_id(){ return personid; }
   public int get_grade(){ return grade; }
}
```

```
package JobSchedule;

import java.awt.*;
import java.io.Serializable;

public class Weight implements Serializable{
   private double w;
   private int i;
   public Weight(){ w=0.5; i=5;}
   public Weight(double w1){    w=w1;
   }
   public void put_w(double w1){w=w1;}
   public void put_i(int i1){i=i1;}
   public double get_w(){return w;}
   public int get_i(){return i;}
}
```

```
package JobSchedule;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import java.io.Serializable;


public class Work_schedul implements Serializable{
   private int personid;
   private Vector schedul;
   public Work_schedul(int id){
      personid=id;
      schedul=new Vector();
   }

   public Work_schedul() { }
   public int get_id(){ return personid;}
   public Vector get_schedul(){ return schedul;}
}
```

GLOSSARY

A

atomic component: An atomic component is a component that cannot be decomposed into refined components.

atomic evolution step: An atomic evolution step is a step that cannot be decomposed into refined steps. (See definition 15 in Chapter 3.)

atomic evolutionary hypergraph: An atomic evolutionary hypergraph is an evolutionary hypergraph that cannot be decomposed into refined hypergraphs. (See definition 17 in Chapter 3.)

atomic relational hypergraph net: An atomic relational hypergraph net is composed of a set of atomic SPIDERs. The atomic relational hypergraph net describes the relationship among each atomic step and its input and output nodes. (See definition 35 in Chapter 3.)

atomic SPIDER: It is an atomic step processed in different entrance relationships. (See definition 35 in Chapter 3.)

atomic step: An atomic step is a step that cannot be decomposed into refined steps.

C

C2: *CALL Dictionary and Thesaurus* defines C2 (Command and Control) as the exercise of authority and direction by a properly designated commander over assigned forces in accomplishing the mission.

C3: *CALL Dictionary and Thesaurus* defines C3 (Command, Control, and Communication) as the capabilities required by commanders to exercise command and control of their forces.

C3I: *CALL Dictionary and Thesaurus* defines intelligence as the collection of functions that generate knowledge of the enemy, weather, and geographical features required by a commander in planning and conducting combat operations.

C4I: *CALL Dictionary and Thesaurus* defines C4I as integrated systems of doctrine, procedures, organizational structures, personnel, equipment, facilities, and communications designed to support a commander's exercise of command and control through all phases of the operational continuum.

C4I users: C4I users could be a composite warfare commander, an officer in tactical command, a warfare area commander, a tactical action officer, a communication officer, etc.

CALL: CALL represents Center for Army Lessons Learned.

COMFORM (COnfiguration Management FORmalization for Maintenance): COMFORM is composed of several phases to provide the necessary guidance to maintain existing software systems.

Computer-Aided Prototyping System (CAPS): CAPS is an easy to use, visual and integrated tool that can be used to rapidly design real-time applications using its PSDL editor, reusable software database, program generator, real-time scheduler, and so on.

Computer-Aided Software Evolution System (CASES): CASES provides automated assistance throughout software evolution processes, using inferred dependencies to support the practical development of complex systems by physically distributed teams of developers.

chief information officer (CIO): The chief information officer (CIO) is a top level leader/ manager who monitors the entire software development process and manages the administration of project teams.

communication link: Communication links could be any digital communication system capable of transmitting and receiving digital messages.

component content link database: Because each output node of an atomic SPIDER has different types of content, we design component content links to connect different content in component content repositories. The component content links are stored in the component content link database.

component content repository: The component content repository includes the text component base, the software component base, and the personnel database.

component management: Component management is one of CASES functions. In this function stakeholders can enter, delete, retrieve, modify, and query the attributes of atomic component from the hypertext database or software library (including software base and design database).

component traceability: Component traceability is one of CASES functions. In this function an atomic component generated by its source atomic step can be traced not only by primary input which is the link between old version and new version atomic components, but also by a secondary input which is the link between source atomic step and components on which it depends, such as requirements and problem reports.

composite component: A composite component can be decomposed into refined components.

composite edge: A composite edge can be decomposed into refined edges in a hypergraph. (See definition 5 in Chapter 3.)

composite node: A composite node can be decomposed into refined nodes in a hypergraph. (See definition 5 in Chapter 3.)

composite step: A composite step can be decomposed into refined steps.

constraint management: Constraint management is one of CASES functions. In this function the project organizer sets constraints that affect the scheduling of steps, such as predecessors, priorities, deadlines, estimated duration, earliest start times, finish times, as well as constraints that affect personnel assignments, like security level and skill requirements for a step.

current component: A current component is a component a stakeholder is working on.

current step: A current step is a step a stakeholder is working on.

customer role: The roles of customers include system owners and end users.

D

dependency: The dependencies among software evolution objects are classified into four types: component-to-step, step-to-component, component-to-component, and step-to-step dependencies.

dependency action rule: The specific combination of dependencies can automatically support software evolution via the dependency action rules (stated by $\Rightarrow$).

dependency-computing model: The dependency-computing model integrates the fundamental software evolution model, such as the hypergraph model; the evolutionary hypergraph model; and the RH model, with the dependency rules that are driven by a lightweight inference engine.

dependency generation rule: According to data existence, the lightweight inference engine computes the dependencies among the software evolution objects via the dependency generation rules (stated by $\Leftrightarrow$).

dependency management: Dependency management is one of CASES functions. In this function the dependencies among atomic components to an atomic step can be identified and managed.

dependency rule: There are two kinds of dependency rules: *dependency generation rules* and d*ependency action rules*.

design database: The design database contains the PSDL prototype descriptions for each software development project using CAPS.

dynamic state model: The dynamic state model of evolution steps includes six states for a software evolution step: Proposed, Approved, Scheduled, Assigned, Completed, and Abandoned.

E

end user: The end user is a person who uses the software product and manipulate the software system.

Evolution Control System (ECS): The ECS provides automated assistance for the software evolution process in an uncertain environment where designer tasks and their properties are always changing. An ECS has two main functions. The first is to control and manage evolving software system components (version control and configuration management). The second is to control and coordinate evolution team interactions (planning and scheduling software evolution tasks, which they refer to as evolution steps).

evolution history merging: Creating a new component based on two primary input components is called software evolution history merging.

evolution history splitting: Creating a new component in a variant different from the original variant is called software evolution history splitting.

evolutionary hypergraph: An evolutionary hypergraph is a labeled, directed, and acyclic hypergraph together with component and step attributes. The evolutionary hypergraph is a multi-level structure due to the refinement of the hyperedge. (See definition 13 in Chapter 3.)

F

formal method: Formal method is an approach that uses mathematical and logical definitions to formalize real-word object behaviors and obtain mature engineering disciplines.

formal notation of relational hypergraph net: We use a formal notation, production formula, to represent a relational hypergraph net.

formal model: Formal model is a model that can be built by formal methods.

G

graph model (or graph data model): The graph model represents the evolution history as a directed acyclic graph $G = [C, S, I, O]$ which is a bipartite with respect to the edges I and O. To model the hierarchical structure of the evolution history, the graph model was modified to be a graph $G = [C, S, CE, SE, I, O]$.

H

heuristic scheduling algorithm: The heuristic scheduling algorithm can be used to improve scheduling time complexity problems.

hyperedge: The hyperedge is a multi-level structure of the evolution step.

hypergraph: The hypergraph is a dag (directed acyclic graph) with no looping paths. (See definition 1 in Chapter 3.)

hypergraph model: The hypergraph model is introduced to formalize the hierarchical structure of the evolution history in more detail.

hypergraph set: The hypergraph set is a set of hypergraph. (See definition 6 in Chapter 3.)

hyperpath: The hyperpath is defined to describe hypergraph traceability. (See definition 10 in Chapter 3.)

hyper-requirements: The approach builds on earlier work on hyper-programming, and is intended to support, by linking related objects, both the social context of requirements decisions and their traceability.

I

inference rule management: Inference rule management is one of CASES function. In this function the stakeholders can specify and adjust inference rules related to SPIDER formation, scheduling and assignment constraints, policies, special assignments, and so on, to help them resolve the design and management issues of the software development process.

input component: The input component to a current step is a set that combines a primary input component set and a secondary input component set.

input component search engine: The input component search engine can trace the dependencies among the software evolution components with the inference rules to find the input scope of the induced step.

issue analysis step: System analysts evaluate some issues from criticisms. (See definition 24 in Chapter 3.)

Issue-Based Information Systems (IBIS) model: IBIS model follows the principle that the design process for complex systems is fundamentally a conversation among the stakeholders to resolve design issues. This model was extended to encompass prototype demos, analysis, and design activities and applied to design a decision support mechanism for software requirements engineering.

J

job assignment: CASES assigns a job to system analysts or system designers.

job assignment engine: The function of the job assignment engine is to search a group of people who can achieve the software evolution activities in a specified atomic SPIDER.

job scheduling model: The job scheduling model is based on the heuristic mechanism that provides the features to rearrange/cancel requests in the step priority queue, and change step priorities dynamically.

L

lightweight inference engine: The lightweight inference engine is designed to compute the small scale and specific domain inference rules.

M

Missile Defense (MD) System: The MD system provides defense functions to a specified area or a nation so that it can be extended to a TMD (Theater Missile Defense) system and a NMD (National Missile Defense) system.

minimal hypergraph: A minimal hypergraph is a minimal unit of hypergraph whose edge set has only one edge. (See definition 7 in Chapter 3.)

module implementation step: System designers implement modules based on specifications. (See definition 27 in Chapter 3.)

myopic algorithm: The myopic algorithm is a job scheduling algorithm. Instead of using all of the remaining tasks to determine if a partial schedule is strong-feasible, Ramamritham limited the candidate tasks to check to some number k. Instead of looking at all the remaining tasks, we "myopically" examine the next k tasks.

N

navigation system: Navigation system is a system that provides a platform with its own positioning, course, velocity, and time data.

O

object-oriented method: The object-oriented method to building a system is based on the definition of a set of communicating entities called objects.

output component: A step can generate one unique output component.

P

path: A path in the hypergraph is an evolution history whose components, including nodes and hyperedges, can be traced. (See definition 2 in Chapter 3.)

personnel component retrieval engine: The personnel component retrieval engine can access the content of the personnel components, regarded as virtual teams or stakeholders, from the personnel database according to the version and variant number of a specified personnel component.

personnel management: Personnel management is one of CASES functions. In this function project managers control the current status of the project personnel such as skill, skill level, security level, on-hand jobs, and so forth.

platform sensor: Platform sensors could be any locally-mounted device capable of identifying azimuth, elevation, velocity, and/or heading if a contact or track is considered to be a platform.

primary-input-driven hypergraph: Each path in a primary-input-driven hypergraph is constructed by primary-input-driven path. (See definition 20 in Chapter 3.)

primary-input-driven path: If there exist an input node and an output node to an evolutionary hyperedge that are different versions of the same component then the path from the input node via the hyperedge to the output node is called a primary-input-driven path.

primary input component: If there exist an input component and an output component to a step that are different versions of the same component then the input component is called a primary input component.

primitive component: The primitive component that is a source component can not be produced by any step.

production formula: Production formula is a formal notation for representing a relational hypergraph net.

program integration step: System designers integrate software prototype programs from modules. (See definition 28 in Chapter 3.)

project evaluation: Project evaluation is one of CASES functions. In this function after project organizers propose an evolution step as a project, this project will be evaluated by project evaluators according to the possibility analysis of executing this software evolution step.

project evaluation team: The project evaluation teams include project team leaders/ managers and project evaluators.

project evaluator: The project evaluators take the responsibility of evaluating the software project including the following activities: (1) evaluate and modify software evolution processes under a specific software project. (2) evaluate and upgrade security levels, required skills and levels for stakeholders. (3) evaluate the formation of an atomic

SPIDER with the scheduling, skill, and security constraints proposed by project organizers or system designers. (4) make the risk assessment and the failure impact evaluation for a job.

project organization team: The project organization teams include project team leaders/ managers and project organizers.

project organizer: The project organizers take the responsibility of organizing a software project including the following activities: (1) create a project and define software evolution object types under a specific software project. (2) modify definitions of software evolution object types under a specific software project. (3) create or modify software evolution processes under a specific software project. (4) define or modify dependencies among software evolution objects, explore and manage required skills of projects. (5) manage the security level of a stakeholder. (6) organize an atomic SPIDER as a job and propose the job with scheduling, skill, and security constraints to a project evaluation team. (7) schedule and assign a job to a project analysis team or a project design team.

Project Scheduling Tool (PST): PST is based on scheduling algorithms of ECS which was developed by Salah Badr.

project team: In CASES, there are three kinds of project teams: the project organization team, the system analysis team, and the system design team.

project team leaders/managers: The project team leaders/managers lead the members of the project team: project organizers, project evaluators, system analysts and system designers, and manage the progress of evolution steps.

Prototype System Description Language (PSDL): PSDL is a specification language that is used in CAPS. PSDL provides graphical notation for dataflow diagrams enhanced with nonprocedural control timing constraints.

prototyping method: The prototyping process repeats a guess/check/modify cycle until the users agree that the demonstrated behavior is acceptable.

Q

QSS DOORS: QSS DOORS is a software system development tool currently selected by U.S. Treasury Inspector General for Tax Administration Corporate Management System Information Project

R

real-time embedded software prototyping: Real-time embedded software prototyping is a method to create a system that is able to react to new events within giving time constraints.

relational hypergraph: A relational hypergraph is an evolutionary hypergraph in which the dependency relationships between components and steps can have a hierarchy of specialized interpretations. (See definition 22 in Chapter 3.)

Relational Hypergraph Model (RH model): The RH model is a formal model for the software evolution which can help us develop tools to manage both the activities in a software development project and the products that those activities produce.

relational hypergraph net: The relational hypergraph net is a relational hypergraph which transfers a primary input hypergraph and secondary input hypergraphs into a top-level evolutionary hypergraph and an atomic evolutionary hypergraph. Therefore, a relational hypergraph net includes a top-level relational hypergraph net and an atomic level relational hypergraph net. (See definition 36 in Chapter 3.)

requirement analysis step: System analysts analyze some requirements from issues. (See definition 25 in Chapter 3.)

requirements management tool (RMT): The requirements management tool (RMT), such as QSS DOORS, can manage all the phases of the life cycle.

reusable software evolution component: The components can be reused in software evolution processes.

RH model base: RH model base is designed to store the relational hypergraph nets.

S

Schematic Model of the Analysis Process: Schematic Model of the Analysis Process is a formal process of software evolution developed by Ibrahim.

scheduling algorithm: The goal of the scheduling algorithm is to determine if a schedule for executing the tasks that satisfies the timing, precedence, and resource constraints exists, and to calculate such a schedule if it exists. The scheduling algorithm of a job is integrated by *JobSchedule* and *JobAssign* algorithms.

scheduling constraint: Scheduling constraints of a job include skills and skill levels, security level, predecessors, priority, deadline, estimated duration, earliest start time, and finish time.

scheduling policy heuristic: Scheduling policy heuristics are a set of scheduling policy rules that help CASES to schedule a job.

secondary-input-driven hypergraph: Each path in a secondary-input-driven hypergraph is constructed by secondary-input-driven path. (See definition 21 in Chapter 3.)

secondary-input-driven path: If there exist an input node and an output node to an evolutionary hyperedge that are different components then the path from the input node via the hyperedge to the output node is called a secondary-input-driven path.

secondary input component: If there exist an input component and an output component to a step that are different components, then the input component is called a secondary input component.

skill: The skill can be any related techniques of the software evolution.

software base: The software base contains PSDL descriptions and code for all available reusable software components.

software component: Software components include software code and the abstract code of specifications and designs.

software component retrieval engine: The software component retrieval engine can access the contents of the software components, such as specifications and programs, from the software component base according to the component content links of a specified software component.

software component reuse: The concept of software component reuse was applied not only to the reuse of software code but also to the reuse of abstract code of specifications and designs.

Software Development Life Cycle (SDLC): The SDLC model is called the waterfall model whose phases include requirements gathering, analysis, modeling or design, coding and testing.

software engineer role: The roles of software engineers include project team leaders/ managers, project organizers, project evaluators, system analysts, and system designers.

software evolution: We consider software evolution to include all the activities that change a software system, as well as the relationships among those activities.

software evolution component: Software evolution components include software and all of the components that are related to software evolution, such as *criticisms, issues, requirements, specifications, modules, programs, optimizations, test scenarios, and stakeholders,* within software evolution processes.

software evolution control: We use dependency rules to control software evolution activities.

software evolution description: We use graph model, hypergraph model, RH model to describe software evolution objects.

software evolution history (or evolution history): We use relational hypergraph to construct software evolution history.

software evolution management: We use dependency rules to manage software evolution objects.

software evolution object: Software evolution objects include software evolution steps and software evolution components.

software evolution process: Software evolution process includes software prototype evolution process and software product generation process. (See definition 33 in Chapter 3.)

software evolution search function: The software evolution search functions are designed as an interpreter to get the software evolution objects and their dependencies, to compute the number of objects in a net or step, and to evaluate properties in a relational hypergraph net.

software evolution step (or evolution step): Each software evolution step has an estimated task duration, deadline, priority, and a required skill level. Software evolution steps in software evolution process include: software prototype demo step, issue analysis step, requirement analysis step, specification design step, module implementation step, program integration step, software product demo step, and software product implementation step.

software evolution traceability: The issues of traceability in software evolution can be represented by paths of the hypergraph.

software product: Software product is a proposed software program that can be released as a commercial product.

software product demo step: System designers demo software product programs to obtain some optimizations. (See definition 29 in Chapter 3.)

software product generation process: The software production generation process repeats a cycle that optimizes and implements production codes from final results of the software prototype evolution process. (See definition 32 in Chapter 3.)

software product implementation step: System designers implement software product programs based on optimizations. (See definition 30 in Chapter 3.)

software project: A software project is a project that can be built by the RH model, organized by project organizers, evaluated by project evaluators, and completed by system analysts and system designers.

software prototype: A software prototype can be changed by customers. The final version of software prototype can be developed to a software product.

software prototype demo step: System designers demonstrate software prototype programs to obtain some criticisms. (See definition 23 in Chapter 3.)

software prototype evolution process: The software prototype evolution process repeats a guess/check/modify cycle until the users agree that the demonstrated behavior is acceptable. (See definition 31 in Chapter 3.)

specification design step: System designers design specifications based on requirements. (See definition 26 in Chapter 3.)

SPIDER: SPIDER denotes the Step Processed In Different Entrance Relationships.

SPIDER formation: SPIDER formation in preparation for executing a software evolution step is a component retrieval process via dependency rules and a lightweight inference.

stakeholder: According to Ibrahim's study, stakeholders include customers, designers, and implementers. We think the stakeholder has many roles in the software evolution. (See stakeholder role in glossary.)

stakeholder role: In the software evolution process, customers and software engineers are the primary stakeholders. The roles of customers include system owners and end users. The roles of software engineers include project team leaders/managers, project organizers, project evaluators, system analysts, and system designers.

step attribute retrieval engine: The step attribute retrieval engine can access the basic attributes of software evolution steps from the step database.

step database: a step database is designed to store the attributes of the software evolution steps.

step management: Step management is one of CASES functions. In this function the content of the top-level step can be automatically generated, refined, and queried. The content of the atomic step can also be automatically generated, combined, and queried.

step refinement: Step refinement is one of CASES functions. In this function, the software evolution top-level step can be refined into a set of atomic steps.

syntax-directed editor (SDE): SDE is an editor for editing PSDL grammar. No more SDE is provided for users in the new version of CAPS.

system analysis team: The system analysis teams include project team leaders/managers and system analysts.

system analyst: The system analysts take the responsibility of completing the analysis steps of software evolution, such as the criticism analysis step, the issue analysis step, and the requirements analysis step.

system design team: The system design teams include project team leaders/managers and system designers.

system designer: The system designers complete the design step of the software evolution, such as the specification design step, the module implementation step, the program integration step, the prototype demo step, the product implementation step, and the product demo step.

system developer: System analysts and system designers are also called system developers who carry out a assigned job by CASES.

system owner: The system owner is a sponsor who supports the software development project and owns the result of the developed software.

T

TAE Plus (Transportable Applications Environment Plus): TAE plus is developed to help user create the user interface. Source code for the user interface can then be generated directly from the code generator.

test scenario: A test scenario is created to test a software prototype program or a software product program.

text component retrieval engine: The text component retrieval engine can access the contents of text components, such as criticisms, issues, requirements, optimizations, and test scenarios, from the text component base according to the component content links of a specified text component.

top-level evolution step: The top-level evolution step is the root step of an evolutionary hypergraph. (See definition 14 in Chapter 3.)

top-level evolutionary hypergraph: The top-level evolution hypergraph is the root of an evolutionary hypergraph. (See definition 16 in Chapter 3.)

top-level relational hypergraph net: A top-level relational hypergraph net is composed of a set of top-level SPIDERs. The top-level relational hypergraph net describes the relationships not only among each top-level step and its input and output nodes but also among each composite node and its subnodes. (See definition 34 in Chapter 3.)

top-level SPIDER: This is a top-level step processed in different entrance relationships. (See definition 34 in Chapter 3.)

U

unknown dependency: The dependency unknown states that the dependency between two arbitrary components at the current time is unknown and needs to be inferred from the dependency rules.

V

variant: Variants represent alternative formulations of a software object with different objectives, such as running on different operating systems or serving different user communities.

version: A version of an object is one of the attributes of this object that can be represented as a string type containing the concatenation of an object identifier, a variant number, and a version number.

version control and configuration management: Version control and configuration management is one of CASES functions. In this function, a labeling function of CASES automatically determines the version and variation number of output components of a step. Software evolution process loops of CASES automatically construct the configuration management.

virtual team: A virtual team is formed to handle the input components and produce the output component to a specified step.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library 2
   Code 052
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. STB-ARL 1
   115 O'Keefe Building
   Georgia Institute of Technology
   Atlanta, GA 30332-0800

4. Dr. David Hislop 1
   Army Research Office
   4300 S. Miami Blvd.
   Research Triangle Park, NC 22709-2211

5. Chairman, Professor Dan Boger, Code CS/Bo 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

6. Software Engineering Program, Code CS 5
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

7. Professor Valdis Berzins, Code CS/Bz 2
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

8. Professor Luqi, Code CS/Lq 2
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

9. Professor Xiaoping Yun, Code EC/Yx 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943

10. Professor Craig Rasmussen, Code MA/Ra 1
Department of Mathematics
Naval Postgraduate School
Monterey, CA 93943

11. Hanh Le, Code D4223 1
Space and Naval Warefare Systems Center
53560 Hull Street
San Diego, CA 92152-5001

12. LTC Meng-Chyi Harn 10
National Defense Management College
P. O. Box 90046-15
Chung-Ho, Taipei 24500, Taiwan, R.O.C.