NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SPECIAL OPERATIONS MISSION PLANNING AND ANALYSIS SUPPORT SYSTEM

by

Keith A. Hattes

June 1999

Thesis Advisor:	Gordon H. Bradley
Second Reader:	Arnold H. Buss

Approved for public release; distribution is unlimited.

19991129 016

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 for 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	3. REPORT TYPE AND DATES COVERED
	June 1999	Master's Thesis

4. TITLE AND SUBTITLE
SPECIAL OPERATIONS MISSION PLANNING AND ANALYSIS SUPPORT SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Hattes, Keith A.

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)
United States Special Operations Command, Attn: SORR-SC
7701 Tampa Point Blvd.
MacDill AFB, FL 33621

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 U.S. Government.

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

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Current mission preparation and analysis methods place an undue burden of effort on conventional and special operations forces to effectively synchronize and execute their increasingly complex operational responsibilities in a rapidly changing global environment. This thesis developed a tool for the United States Special Operations Command (USSOCOM) in support of their Mission Planning, Analysis, Rehearsal, and Execution (MPARE) initiative to allow special operations forces commanders and staffs to conduct mission planning and analysis in a distributed environment, and rapidly produce dynamic synchronization matrices and scheduling products. Operations research methods provide the foundation for the analysis. The system developed in this thesis is called the Special Operations Mission Planning and Analysis Support System (SOMPASS). SOMPASS is simple to learn and operate, provides dynamic changes with little effort, and is universal in application. This system has the capability to execute on any hardware platform, operate across any network connection, and expand easily to support additional users and requirements. This thesis provides not only a demonstration of capabilities through a special operations oriented illustrative scenario, but also a working product that can be adapted for use in mission planning and analysis by all units under USSOCOM.

14. SUBJECT TERMS
Mission Planning, Analysis, Synchronization, Critical Path Method, CPM, Special Operations, MPARE, Java, Loosely Coupled Components

15. NUMBER OF PAGES
152

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	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239.18

Approved for public release; distribution is unlimited.

SPECIAL OPERATIONS MISSION PLANNING AND ANALYSIS SUPPORT SYSTEM

Keith A. Hattes
Captain, United States Army
B.S., United States Military Academy, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 1999

Author: Keith A. Hattes

Keith A. Hattes

Approved by:

Gordon H. Bradley, Thesis Advisor

Arnold H. Buss, Second Reader

Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

Current mission preparation and analysis methods place an undue burden of effort on conventional and special operations forces to effectively synchronize and execute their increasingly complex operational responsibilities in a rapidly changing global environment. This thesis developed a tool for the United States Special Operations Command (USSOCOM) in support of their Mission Planning, Analysis, Rehearsal, and Execution (MPARE) initiative to allow special operations forces commanders and staffs to conduct mission planning and analysis in a distributed environment, and rapidly produce dynamic synchronization matrices and scheduling products. Operations research methods provide the foundation for the analysis. The system developed in this thesis is called the Special Operations Mission Planning and Analysis Support System (SOMPASS). SOMPASS is simple to learn and operate, provides dynamic changes with little effort, and is universal in application. This system has the capability to execute on any hardware platform, operate across any network connection, and expand easily to support additional users and requirements. This thesis provides not only a demonstration of capabilities through a special operations oriented illustrative scenario, but also a working product that can be adapted for use in mission planning and analysis by all units under USSOCOM.

DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1
1. Military Operations 1
2. The Nature of Special Operations 3
3. Vision 4
4. Need 5
5. Current Systems 8
6. Mission Planning, Analysis, Rehearsal, and Execution (MPARE) 10
7. Operations Research Applications 12

B. PROBLEM 13

C. STATEMENT OF THESIS 15

II. THEORY AND FOUNDATIONS 17

A. NETWORK FLOWS AND GRAPHS 17

B. SOLUTION METHODS 18
1. Critical Path Method (CPM) 18
2. Program Evaluation and Review Technique (PERT) 19

C. TASK REPRESENTATIONS 20
1. Activity on Node (AON) 20
2. Activity on Arrow (AOA) 21

D. COMPONENTS 23
1. Characteristics 23
2. Loosely Coupled Component Architecture 24
3. König 26
4. Thistle 27

E. ALGORITHMS 29
1. Topological Sort 30
2. Longest Path 30
3. Critical Path Method 32

III. SYSTEM DESIGN 35

A. REQUIREMENTS 35

B. CAPABILITIES 36
1. Platform Independent 36
2. Dynamic Loading and Viewing 38
3. Distributed 39
4. Thin Client 40
5. Extensible 41

C. MODEL 42
1. Network Representation 42
2. Tasks and Properties 43

D. ARCHITECTURE 45
1. Loosely Coupled Components 45
2. Graphical User Interface (GUI) 46
3. Model-View-Controller (MVC) 47

E. IMPLEMENTATION 48
1. Description 48
2. Operation 49
3. Products 55

IV. ILLUSTRATIVE SCENARIO 59

A. SITUATION 59

B. TASK ORGANIZATION 59

C. MISSION DEVELOPMENT 60
1. Situation 60
2. Mission 61
3. Concept of the Operation 62

D. MISSION TASKS, EVENTS, AND DEPENDENCE 62

E. IMPLEMENTATION 63

V. ANALYSIS 65

A. RESULTS 65

B. SENSITIVITY ANALYSIS 67

C. APPLICABILITY 70

VI. THE NEXT LEVEL 71

A. RESOURCE MANAGEMENT 71

B. PRECENDENCE NETWORKS – MULTIPLE DEPENDENCY 71

C. AUTOMATED DATA CALCULATION AND SELECTION 72

D. SIMULATION 72

VII. CONCLUSION 73

APPENDIX A. CRITCAL PATH SOLVER 77

APPENDIX B. ILLUSTRATIVE SCENARIO ACTIVITY AND EVENT LIST 87

APPENDIX C. SYSTEM DEMONSTRATION 91

LIST OF REFERENCES 123

BIBLIOGRAPHY 127

INITIAL DISTRIBUTION LIST 131

LIST OF FIGURES

1. AON Activity 21
2. AON Network 21
3. AOA Activity with Events 22
4. AOA Network 23
5. Loosely Coupled Components Architecture 25
6. König Dijkstra Shortest Path Algorithm Output 27
7. Flora Map Display 28
8. Longest Path Linear Program 29
9. Topological Sort Algorithm 31
10. Longest Path Algorithm 33
11. Model-View-Controller Architecture 47
12. System Architecture 48
13. Critical Path Solver Control Panel 50
14. Blank Flora Map Display for Project Design 51
15. Detachable Graph Editing Tool Bar 51
16. Node Property Editor 52
17. Graph Property Editor 52
18. Simple Network Example 53
19. Flora Symbol Info Display 54
20. Cycle Error Message 54
21. Simple Network Example with Critical Path Solution 55
22. Example Synchronization Matrix 56
23. Example Execution Checklist 57
24. JSOTF Location and Composition 60
25. Mission Area of Operations 61
26. Scheme of Maneuver: Actions on the Objective 63
27. Mission Network Representation 64
28. Zoom in to Actions on the Objective Tasks 64
29. Solved Mission Network 65
30. Mission Synchronization Matrix 66
31. Mission Execution Checklists 67
32. Initial Critical Path and Properties 68
33. Modification of a Task Property 69
34. Updated Critical Path and Properties 69
35. System Startup 91
36. Draw Nodes 93
37. Add Arcs 95
38. Load Graph 97
39. Solve Critical Path 99
40. Highlighted Critical Path with Node Symbol Info Display 101
41. Edit Graph Window 103

42. Loading the Illustrative Scenario 105
43. Display Zoom 107
44. Create Synchronization Matrix and Execution Checklists 109
45. Illustrative Scenario Critical Path 111
46. Illustrative Scenario Synchronization Matrix 113
47. Illustrative Scenario Execution Checklists 115
48. Additional Node Properties 117
49. Edit Node Properties 119
50. Updated Critical Path and Properties 121

LIST OF TABLES

1. Java Supported Platforms and Operating Systems 37
2. Mission Task List 87

LIST OF SYMBOLS, ACRONYMS AND/OR ABBREVIATIONS

AAF	Army Airfield
AAR	After Action Review
AO	Area of Operations
AOA	Activity on Arrow
AON	Activity on Node
API	Application Programmer Interface
BDA	Battle Damage Assessment
BOS	Battlefield Operating Systems
C4I	Command, Control, Communications, Computers, and Intelligence
C4IFTW	C4I For The Warrior
CINC	Commander in Chief
COA	Course(s) of Action
COE	Common Operating Environment
CONOPS	Concept of Operations
COP	Common Operational Picture
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
CPM	Critical Path Method
CPU	Central Processing Unit
CRD	Capstone Requirements Document
DA	Direct Action
DII	Defense Information Infrastructure
DISA	Defense Information Systems Agency
DISN	Defense Information System Network
DMS	Defense Message System
DoD	Department of Defense
EDRE	Emergency Deployment Readiness Exercise
EET	Earliest Event Time
EFT	Earliest Finishing Time
EST	Earliest Starting Time
FM	Field Manual
FOB	Forward Operational Base
FOL	Forward Operating Location
GCCS	Global Command and Control System
GCSS	Global Combat Support System
GUI	Graphical User Interface
HLA	High Level Architecture
HQ	Headquarters
HLZ	Helicopter Landing Zone, see also LZ
JSOTF	Joint Special Operations Task Force
JTA	Joint Technical Architecture

JVM	Java Virtual Machine
LAN	Local Area Network
LCC	Loosely Coupled Components
LET	Latest Event Time
LFT	Latest Finishing Time
LST	Latest Starting Time
LZ	Landing Zone, see also HLZ
MNS	Mission Needs Statement
MPARE	Mission Planning, Analysis, Rehearsal, and Execution
MVC	Model-View-Controller
NGO	Non-Governmental Organization
ODA	(Special Forces) Operational Detachment Alpha, see also SFOD A
ODC	(Special Forces) Operational Detachment Charlie, see also SFOD C
OOP	Object-Oriented Programming
OOTW	Operations Other Than War
OPORD	Operations Order
PERT	Program Evaluation and Review Technique
PZ	Pickup Zone
RGR	Ranger
RMI	Remote Method Invocation
SBF	Support by Fire
SF	Special Forces
SFG	Special Forces Group
SFOD A	Special Forces Operational Detachment Alpha, see also ODA
SFOD C	Special Forces Operational Detachment Charlie, see also ODC
SIPRNET	Secret Internet Protocol Router Network
SO	Special Operations
SOAD	Special Operations Aviation Detachment
SOAR	Special Operations Aviation Regiment
SOC	Special Operations Command
SOCCE	Special Operations Command and Control Element
SOF	Special Operations Forces
SOMPASS	Special Operations Mission Planning and Analysis Support System
SOP	Standing Operating Procedures
SOS	Special Operations Squadron
SOW	Special Operations Wing
SR	Special Reconnaissance
TA	Target Analysis
TACSOP	Tactical Standing Operating Procedures
TF	Task Force
TM	Team
TRAC	TRADOC Analysis Center
TRADOC	United States Army Training and Doctrine Command
USSOCOM	United States Special Operations Command

EXECUTIVE SUMMARY

This thesis developed a tool for the United States Special Operations Command (USSOCOM) in support of their Mission Planning, Analysis, Rehearsal, and Execution (MPARE) initiative to allow special operations forces commanders and staffs to conduct mission planning and analysis in a distributed environment. Operations research methods provide the foundation for the analysis. The system developed in this thesis is called the Special Operations Mission Planning and Analysis Support System (SOMPASS). This system has the capability to execute on any hardware platform, operate across any network connection, and expand easily to support additional users and requirements.

Many current military planning and support systems offer tools that employ operations research techniques, but the rapid increase in complexity of military operations since the end of the Cold War, along with advances in technology, have rendered these systems either obsolete or insufficient. Commanders and staffs will benefit from advances made using operations research techniques in all aspects of military operations, including mission planning, decision support, and logistics management.

Paramount among the choices for development of future technological systems is a solution to address shortcomings in mission planning and analysis. Current military mission preparation and analysis, for both conventional and special operations forces, requires tremendous effort for detailed planning, resourcing, analysis, rehearsal, and synchronized execution to produce the desired success. In an extensive effort to address critical Special Operations mission challenges, USSOCOM began the MPARE initiative

in 1997. One of the critical components to the success of such a system is the ability to synchronize these missions. Special Operations, as well as conventional military operations, have unique requirements that demand special capabilities that are neither provided by commercially available systems, nor available on the current generation of limited functionality military planning systems.

SOMPASS focuses on leveraging the power of a highly flexible component architecture to support the rapid development and construction of military planning and analysis tools and systems that will operate seamlessly over extensible networks on heterogeneous computing hardware and software systems. This system combines several newly developed components with previously developed components. A graphical user interface component was built to tie together the critical path solver algorithm with the graph design tools and present the system specific outputs: the mission synchronization matrix and execution checklists. Together, these components provide a useful and powerful mission planning and analysis support system that is dynamic, flexible, and component based. The special operations scenario presented in this thesis is designed to demonstrate the capabilities and applicability of SOMPASS.

SOMPASS has been designed to address the needs of USSOCOM by providing a mission planning and analysis tool for Special Operations Forces commanders and staffs to help reduce traditional mission planning preparation time and manual effort required to produce operational support documents, including the mission synchronization matrix and unit execution checklists, as well as enhance the capabilities for conducting analysis.

ACKNOWLEDGMENT

I would like to thank Professor Gordon H. Bradley for his outstanding guidance, assistance, and support throughout my work on this difficult project. I would like to thank Professor Arnold H. Buss for always bringing up new ideas to explore that sometimes seemed to make things more complicated, but were for the better. I would also like to thank LTC Charles Shaw for arranging my experience tour with USSOCOM and providing insight into Special Operations, Army Operations Analysis, and the plethora of acronyms that exist out there. I would like to thank the Loosely Coupled Components group for providing a supportive environment for me to showcase my research, and aid me in eliminating the numerous bugs in my system, as well as enhancing other components to suit my needs; in particular, MAJ Jack Jackson of TRAC-Monterey and CPT Norbert Schrepf of The German Army.

I also want to thank the officers and staff of USSOCOM, SORR; namely COL Nolen Bivens and LTC Joel Parker who provided for and sponsored me on my experience tour. LTC Parker was especially helpful in providing focus on what was important, and continued his mentorship after moving here to support the school directly as the new department Army LNO/Instructor. I can only hope that this research will be of some benefit to them to assist our outstanding soldiers.

I. INTRODUCTION

> Today, America's Armed Forces are the world standard for military excellence and joint warfighting. We will further strengthen our military capabilities by taking advantage of improved technology and the vitality and innovation of our people to prepare our forces for the 21st century. [Ref. 1: p. 34]
>
> General John M. Shalikashvili
> Former Chairman of the Joint Chiefs of Staff

A. BACKGROUND

1. Military Operations

Just as the world, its nations, and society evolved throughout history, the nature of warfare has also evolved. Continuing advances in tactics, doctrine, weapons, and technology compose a continuing cycle that progresses with each new step in its component elements. Each of these advances, in turn, causes the nature of warfare and the conduct of military operations to become more complex. As complexity increases, the forces conducting these operations must be better prepared, trained, and equipped than their predecessors in order to succeed. In his preeminent work, *On War*, Carl von Clausewitz summarized the complex nature of military operations: "The conduct of war resembles the working of an intricate machine with tremendous friction, so that combinations which are easily planned on paper can be executed only with great effort." [Ref. 2: p. I-2]

Since the end of the Cold War, the United States faces an increasingly complex and diverse set of challenges due to our role as the world's only remaining superpower with worldwide presence and global power projection responsibilities. In this modern era, technological advances in communications and the increased pace of world events

can produce crises that rapidly expand in unpredictable ways, thereby reducing the time available to prepare and employ forces to defuse these critical situations. Thus, some of the same technological advances that improve our military capabilities strain our forces by increasing the difficulty and complexity of military operations and confound our ability to react to and defuse these crises. [Ref. 2] In a military that operates by force-projection, such as we do today, synchronization of operations is paramount [Ref. 3].

Successful planning can help offset these potential operational problems, though time limitations will always have an overarching impact. Tactical and operational planning must be a continuous process, frequently concurrent with ongoing operations, which further complicates its completion. Successful planning requires an understanding and appreciation of this simultaneous nature, and an ability to anticipate likely future events to counter shortfalls or exploit successes. Detailed synchronization during the planning and execution of the mission will promote successful achievement of the mission objectives. Synchronization requires a clear commander's intent to convey to the staff the idea for the sequence and flow of the operation that must then be developed in the plan with coordination of movement, fires, and supporting activities. This is an extremely complex process wherein coordination, collaboration, and rehearsal are keys to success in providing "...the ability to focus resources and activities in time and space to produce maximum relative combat power at the decisive point." [Ref. 3: p. Glossary-8]

A tool that assists the commander or staff in this synchronization process will therefore greatly enhance their capabilities and the ability of their units to conduct successful operations and achieve victory.

2. The Nature of Special Operations

As their name would indicate, Special Operations differ from conventional military operations. Some of the key differences include the degree of risk involved, both political and physical, as well as the means by which they achieve their objectives. Specifically, "Special Operations are operations conducted by specially organized, trained, and equipped military and paramilitary forces to achieve military, political, economic, or informational objectives by unconventional military means in hostile, denied, or politically sensitive areas." [Ref. 4: p. I-1] Due to their unique and strategic nature, they are often more directly affected by political-military considerations. Since Special Operations can encompass all aspects of military operations and may be conducted either independently or in conjunction with conventional operations, they are inherently more complex and require more detailed planning than conventional military operations. [Ref. 4]

Similarly, Special Operations Forces (SOF) fill a unique role in that they provide the ability to carry out operations where conventional forces are neither well suited nor even capable of success. SOF are specially trained, equipped, and organized units with specialized, highly focused capabilities [Ref. 4]. The unique qualifications and capabilities of SOF provide, to both national and theater level decision-makers, a greater range of options and flexibility in responses through their rapid adaptability and strategic advantage [Ref. 5].

Given the diversity and complexity of Special Operations, the need for tools to assist SOF in the preparation and conduct of their missions is even more pronounced than that of conventional military operations.

3. Vision

As the world environment continues to change and present more complex challenges, our forces must also adapt to this changing environment to preserve our ability to defend against current and emerging threats to our national security. This includes not only our weapons, doctrine, and training, but also the tools we use to assist in the accomplishment of our missions. *Joint Vision 2010* [Ref. 1] provides a template for the continuing development and advancement of our nation's warfighting capability into the future by leveraging advances in information-age technology through the development of four operational concepts: dominant maneuver, precision engagement, full dimensional protection, and focused logistics. Technological superiority has been crucial in our prior successes in combat, and will continue to be so in the foreseeable future. Therefore, continuing advances in information and systems integration technologies must be aggressively developed to provide decision-makers with accurate and timely information to gain "dominant battlespace awareness." [Ref. 1]

The United States Special Operations Command (USSOCOM) also has a vision for the future that expands on the concepts outlined in *Joint Vision 2010* and applies them to the nature of Special Operations and the role of Special Operations Forces. *SOF Vision 2020* [Ref. 6] "...provides a long-range strategy for SOF missions, force structure, equipment, and capabilities into and beyond 2020." [Ref. 6: p. 1]. It outlines defining

characteristics that focus on quality, well-trained personnel with a superior technological edge who provide military capabilities not available with conventional forces [Ref. 6]. Additionally, *Special Operations Forces: The Way Ahead* [Ref. 5] expands on the relevance of SOF and their unique abilities, as well as their need to "...examine every advantage our technological genius can supply...selectively exploit those few required for success...[and] leverage those critical technologies that give us a decided advantage." [Ref. 5: p. 7]

A common thread throughout all of these documents is the importance of information technologies and the critical role they play in the advancement and success of our forces in future conflicts. The sooner we begin the development of these needed systems, the sooner we will be able to leverage technology to our advantage. Although we currently have many advanced systems in our inventory, technology has far outpaced development and acquisition, and many commercially available systems are neither well suited nor easily adaptable to military use. As the United States is not the only bastion of technological advancement, we must make a concerted effort to foster developments to ensure we do not fall behind the advances of any current or potential adversaries.

4. Need

Paramount among the choices for development of future technological systems is a solution to address shortcomings in mission planning and analysis. Our increasing reliance on superiority in command, control, communications, computers, and intelligence (C4I) highlights the need for more than current systems can provide and interim solutions will offer. All of the new operational concepts put forward in *Joint*

Vision 2010 will rely significantly on advanced C4I systems and capabilities. But in particular, "Dominant maneuver will require forces that are adept at conducting sustained and synchronized operations from dispersed locations." [Ref. 1: p. 20] The ability to meet this mandate will depend on systems that offer extremely capable tools with superior flexibility and interoperability.

Joint Vision 2010 addresses the need to exploit technological advances through the development of a new framework where foundations are built on improved C4I achieved through information superiority [Ref. 1]. Joint doctrine refines this vision by presenting C4I For The Warrior (C4IFTW): "What the Warrior Needs: a fused, real time, true representation of the battlespace — an ability to order, respond and coordinate horizontally and vertically to the degree necessary to prosecute his mission in that battlespace." [Ref. 7: p. I-1]. Additionally, it defines the requirements and capabilities for this global information infrastructure and system: "Warfighters must have C4 systems that are interoperable, flexible, responsive, mobile, disciplined, survivable, and sustainable." [Ref. 7: p. ix]

USSOCOM has also taken detailed steps in identifying needs in the area of mission planning and decision support. They have developed a strategy that addresses their special C4I needs and requirements by outlining a doctrine, architecture, and investment strategy to support and improve SOF operational capability along with a plan for implementation [Ref. 8].

Despite this recognized need for a superior technology system to assist commanders, staffs, and their forces in all aspects of operations from mission planning to

execution, no satisfactory system has yet been developed or fielded. Quite the contrary, most units, including conventional and special operations forces, use many of the same techniques that have been in use since World War II and throughout the Cold War era.

To further elaborate the need for a solution to this problem, an explanation of the current planning process is useful. Current military mission preparation and analysis, for both conventional and special operations forces, requires tremendous effort for detailed planning, resourcing, analysis, rehearsal, and synchronized execution to produce the desired success. Units at all levels of command must coordinate their actions with those above, below, and around them, to ensure the overall mission is successful. Commanders and staffs must accomplish many interdependent tasks as they determine their courses of action (COA) to accomplish missions during both the deliberate and compressed planning sequences. This preparation includes having a thorough understanding of the many requirements, as well as the ability to meld them into a workable plan that can be easily understood and successfully implemented. Successfully preparing these mission requirements sets the stage for the accomplishment of the mission, and is therefore a critical precursor. Understandably, any means to enhance the abilities of a commander and staff in preparation for or control of a mission can yield significant benefit toward overall mission success.

Readily available commercial tools do not provide the needed capabilities, flexibility, or adaptability required in supporting complex military operations, nor address the specific requirements outlined in either *Joint Vision 2010* or the USSOCOM documents. Additionally, currently fielded military planning systems that are in use

today are also significantly lacking in flexibility, usually offer only limited static solutions, and adhere to monolithic standards [Ref. 9]. Any possible solution must therefore overcome current shortcomings and address the specific needs mentioned.

5. Current Systems

Many systems that address one, or several, of the requirements from above currently exist, but at present, there is no single system capable of addressing all of these requirements [Ref. 10]. In an effort to spearhead the development of new technologies to support these warfighter requirements, the Department of Defense (DoD) created the Defense Information Systems Agency (DISA) [Ref. 11]. Specifically, the mission of DISA is "to plan, engineer, develop, test, manage programs, acquire, implement, operate, and maintain information systems for C4I and mission support under all conditions of peace and war." [Ref. 12]

DISA manages the Defense Information Infrastructure (DII), which is an attempt to integrate all DoD communication networks, hardware, and software and construct a common operating environment (COE) to support the information requirements of warfighters. The DII is made up of four components: the Defense Information System Network (DISN), the Defense Message System (DMS), the Global Command and Control System (GCCS), and the Global Combat Support System (GCSS). [Ref. 12]

The DISN provides the hardware-based network infrastructure, as a communications backbone, to support global, strategic, and tactical connectivity. It is made up of the transmission paths and support structures, over which data may pass. The

Secret Internet Protocol Router Network (SIPRNET) is the secret layer of the DISN. [Ref. 13]

The DMS is a recent improvement over previous systems that provides a more flexible, commercial off-the-shelf (COTS) system for e-mail and multi-media messaging services using the DISN. The DMS is an integrated suite of applications and not a network, nor an information processing or planning tool. [Ref. 14]

Unlike the previous two components that are basically only support structures, the GCCS is a mid-term implementation of the C4IFTW concept. It provides functionality to address some of the outlined C4I requirements, including applications providing a COE, and mission applications providing planning and assessment tools [Ref. 15]. However, as an interim solution, it lacks significant strength in features and functionality that are critical to the success of the C4IFTW vision. Additionally, it is a static system that is fairly inflexible to enhancement or cross-platform/system connectivity, and will undoubtedly have to be abandoned in its current form to allow future advances in capabilities to be implemented in a follow-on system.

GCSS is another C4IFTW-based system that was developed to provide improved combat service support to warfighters as envisioned in *Joint Vision 2010*'s concept of "focused logistics." It is also an interim solution, and as with GCCS, provides only limited functionality at the present time and cannot currently be integrated with GCCS or any other information support system. [Ref. 16]

These products from DISA represent significant advances over the capabilities of all previous systems, but still fall well short of the outlined requirements for a true

information system solution. The significant shortcomings of these current systems lie primarily in connectivity, collaboration, timeliness, and functionality. Current development and procurement practices also seem to contribute to limitations in functionality based on contractor proprietary considerations, poor understanding of required versus desired capabilities, and uncoordinated/unsynchronized development of alternative or competing systems between the Service components who focus on different features and schedules.

In addition to the DoD and each Service component, along with many of their subordinate elements, USSOCOM, as a unified combatant command, is also dedicating considerable effort to developing a C4I system that supports its unique requirements. USSOCOM currently has several of its own interim solutions in use, principally SOFPARS and SWAMPS, which they recognize are also proprietary, disjointed, and inflexible, thereby placing limits on their usefulness for meeting the needs of mission planning, coordination, and synchronization [Ref. 10].

6. Mission Planning, Analysis, Rehearsal, and Execution (MPARE)

In an extensive effort to address critical Special Operations mission challenges, USSOCOM began the Mission Planning, Analysis, Rehearsal, and Execution (MPARE) initiative in 1997. The goal of MPARE is to provide SOF commanders, staffs, and operators a totally integrated "system of systems" with which they can efficiently plan, analyze, rehearse, and execute the full spectrum of Special Operations missions. This system will also provide communications services and collaborative capabilities between elements both vertically and horizontally to facilitate information flow and

synchronization. It is intended to be ubiquitous, to support all operations in training and combat environments, as well as handle routine administrative functions. The key components of MPARE will enable SOF units to collaboratively plan missions from geographically separate locations, analyze different COAs, preview and rehearse options, and monitor execution in real-time. [Ref. 10]

Although still in the early developmental stage, MPARE is laying the groundwork for an in-depth understanding of the true requirements for the C4I systems of the future which will not only greatly benefit USSOCOM, but also the DoD and the Services who have been trailing behind with their own systems. MPARE has passed Milestone 0, and work is progressing on the Capstone Requirements Document (CRD) and component Concept of Operations (CONOPS) documents [Refs. 10, 11]. The current and previous Commanders in Chief (CINCs) of USSOCOM have made MPARE a high priority, and as such, a full-time team of military and civilian personnel, along with contractor support, is dedicated to MPARE development. As part of the requirement determination stage, meetings with members of the MPARE team and key personnel from all of the Theater Special Operations Commands (SOCs) provide invaluable input from the future users of the system. These meetings have uncovered numerous common issues, many of which are already planned for incorporation in MPARE, and others that have since been added to provide improved functionality [Refs. 17, 18, 19, 20]. Once completed, MPARE will provide tremendous capabilities to SOF units, with many applications useful to conventional forces, as well.

7. Operations Research Applications

The development of operations research was spearheaded by the Allied military forces during World War II in an effort to solve the tremendously large strategic and operational problems that resulted from conducting military operations on a global scale [Ref. 21]. The United States continues to this day to expend significant effort and resources to further advances and training in operations research in support of military problems, having recognized its leverage early on, and throughout its incorporation into many private-sector industries.

Many current military planning and support systems offer tools that employ operations research techniques, but the rapid increase in complexity of military operations since the end of the Cold War, along with advances in technology, have rendered these systems either obsolete or insufficient. More powerful and flexible tools for future systems and capabilities outlined in *Joint Vision 2010* and USSOCOM initiatives will continue to depend on operations research techniques, not only to facilitate their development, but also for incorporation within these systems.

Commanders and staffs can benefit from advances made using operations research techniques in all aspects of military operations, including mission planning, decision support, and logistics management. These benefits underscore the relevance of continued study in the area of operations research applications to military problems to stay ahead of our current and potential adversaries in a rapidly changing world.

B. PROBLEM

USSOCOM has outlined a need for a capable system to support the conduct of Special Operations missions. One of the critical components to the success of such a system is the ability to synchronize these missions. Current methods for addressing this need fall far short of the requirement.

Once given a mission and set of goals and objectives, commanders and staffs have to conduct extensive preliminary work during mission planning and analysis to determine what must be done, who must do it, what is needed, and how it should be accomplished. This itself is a significant challenge, usually compounded in difficulty by the common constraint of extremely limited preparation time before mission execution. Components of this process involve breaking down a mission into its specified, implied, and essential tasks; identifying critical decision points, developing a concept of the operation and COAs; wargaming the COAs; and producing the operations order (OPORD). To help develop the best possible plans for success, commanders and staffs need to conduct wargaming and analysis to determine if their plans are feasible and accomplish all objectives as desired, and also to choose the best among the alternatives developed. Then, they must develop a synchronization matrix that ties all units and required resources to support the operation to an interdependent time schedule based on expected mission status and probable enemy COAs. This process is very time-consuming and prone to errors. Additionally, since one of the critical end products, the synchronization matrix, is usually constructed by hand on a large sheet of paper or an overlay, any change to an event, time, or status requires complete reconstruction. Due to the very complex

nature of military operations, and in particular, Special Operations, synchronization is extremely difficult because numerous decisions, personnel, equipment, supplies, and actions must come together at critical times and locations throughout the battlespace to produce the desired effect of success in the assigned mission objectives. Clearly, the current process does not support rapid or flexible planning, nor facilitate any sensitivity analysis.

Many commercial software tools currently exist that perform functions similar to synchronization planning; however, none of them are well suited to military operations. These commercially available tools perform functions such as project management or resource and event tracking, but none provide the required capabilities, flexibility, or adaptability required in supporting complex military operations. In addition, most only operate on single machines or small local area networks (LANs), and all gear their functionality toward commercial applications. Consequently, military commanders and staffs have shunned these commercially available tools and relied on their time-tested, manual methods of planning, analysis, and synchronization.

Special Operations, as well as conventional military operations, have unique requirements that demand special capabilities that are neither provided by commercially available systems, nor available on the current generation of limited functionality military planning systems. A tool that would support commanders and staffs in accomplishing the complex task of synchronization planning and generate useful products in a rapid, flexible, and distributed environment would become a key component in the MPARE initiative.

C. STATEMENT OF THESIS

The purpose of this thesis is to develop a mission planning and analysis tool to support Special Operations Forces commanders and staffs by identifying and presenting critical mission events, relationships, and dependencies in a simple and understandable format. Many of the technologies that support the needs of the mentioned desired future systems are available now, but have not been combined into a working system. Most efforts in development are attempting to produce a complete system with full functionality to satisfy all needs when fielded. This approach requires intense and coordinated effort, along with substantial time and funding. This thesis does not attempt to put forward a complete solution; rather it presents a working technical demonstration that can be incorporated into a larger system while still demonstrating specific desired functionality.

This tool will assist special operations commanders and staffs to conduct mission planning and analysis by operating in a collaborative and dynamic environment that allows simple task and event entry and analysis. It will rapidly produce synchronization matrices and scheduling products that are easily updated as changes occur during any phase of the operation, and also allow mission sensitivity analysis through visual depiction of impacts based on task, event requirement, or resource availability changes. In addition to its functionality, this tool will also provide USSOCOM with one possible direction for further development and possible incorporation into the MPARE system, as well as demonstrate the power and importance of operations research tools to military decision-makers.

This thesis consists of seven chapters and three appendices. Chapter II outlines the operations research theory behind project management systems and the methodology behind component design. Chapter III describes the actual software design that implements the model formulation, its requirements, capabilities, architecture, and functionality. Chapter IV presents a Special Operations scenario designed to be unclassified, but representative enough to demonstrate the capabilities and applicability of the system. Chapter V examines the results generated by the system when applied to the illustrative scenario. Chapter VI describes what areas of the system might benefit from additional research and examination. Chapter VII offers conclusions about both the system and research conducted. Appendix A contains the code that implements the algorithms outlined in Chapter II required for the Critical Path Method Solver and showcases the structure of a software component. Appendix B contains the detailed list of activities and events that compose the illustrative scenario put forward in Chapter IV. Appendix C provides a walk-through of the system through the use of screen shots captured during numerous system operations.

II. THEORY AND FOUNDATIONS

Network flows is a problem domain that spans a broad range of fields, including applied mathematics, operations research, engineering, management, and computer science [Ref. 22].

A. NETWORK FLOWS AND GRAPHS

Many optimization problems can best be solved by means of a network representation. Networks can represent nearly any physical or conceptual system that has interdependencies between its components. Graphs then represent these networks in a very simple, yet powerful manner, allowing modeling and analysis techniques to be applied to solve problems related to the underlying systems. The development of efficient algorithms for some of these network and graph models allows them to be solved with much more efficiency than with traditional linear optimization techniques.

Of particular interest is critical path analysis of networks for project management, which can be readily applied to military operational planning, just as it has to many large-scale complex engineering scheduling problems. Critical path analysis highlights the complexities and relationships of activities or tasks that make up a project or mission, and allows for detailed analysis, simple modification, and flexible evaluation to support decision making. [Ref. 23] The critical path of a network is the chain of activities or tasks that, if any are started later or their duration increases, the time to complete the entire project will be prolonged. Float is the amount of spare time, or slack, that exists between the earliest and latest times an activity can start or finish. All activities on the

critical path have zero float. Also note that float is the only means of identifying the critical path. [Ref. 24]

B. SOLUTION METHODS

There are two extensively used critical path analysis project management techniques that can be applied to military operational planning: the Critical Path Method (CPM) and the Program Evaluation and Review Technique (PERT). There are many similarities between the two methods, and both are principally concerned with planning, scheduling, and control, which are key components to the success of all military operations.

Both techniques help answer detailed questions about dependencies between tasks and events in a project or mission, and can easily support comparative analysis when faced with questions about scheduling or resource changes or disruptions. They both support decision-making without requiring complex calculations or analysis by the user. [Refs. 25, 26] Each technique has both advantages and disadvantages depending on the type of mission or problem of interest; some of their differences are highlighted below.

1. Critical Path Method (CPM)

This method focuses on tradeoffs between resource costs and completion time in large complex projects. CPM assumes the time required to complete individual tasks in a project is known with certainty. This is an assumption that greatly simplifies the underlying calculations while providing consistent results, but may not be as applicable to a project where there are large or unknown variances in the execution or completion times of component tasks that may severely affect completion times. [Ref. 25]

The system put forward in this research employs CPM for simplicity of use, timeliness and consistency of results, and to reduce the reliance on assumptions by the user that require more information about component tasks than may be available or verifiable.

2. Program Evaluation and Review Technique (PERT)

Although similar in purpose, PERT differs primarily from CPM in that it assumes that task completion times are uncertain and independent of one another. PERT models the uncertainty of task completions based on assumptions about the distribution and likelihood of the time required to complete the component tasks. This technique requires additional input from the user about these distributions that may not be intuitive or readily available, but when exercised properly PERT can reveal interdependencies or problem areas that would not have otherwise been discovered with a purely deterministic approach. [Refs. 25, 26]

Understandably, the stochastic nature of PERT makes it much more complex and harder to implement effectively as a user friendly application than CPM. This difficulty has led PERT to be used more in research and development projects, while CPM is used in many commercial enterprises including construction and industrial production. [Refs. 25, 26]

Although military operations involve great uncertainty, assumptions about the distributions or variance of random quantities, especially those related to the actions or responses of enemy forces, is very difficult. Consequently, PERT may be more harmful

than helpful when doing military planning if improper assumptions are used in mission formulation, rather than simply using a deterministic approach such as CPM.

C. TASK REPRESENTATIONS

All networks are composed of nodes and arcs. The visual representation of a network is a graph where either a circle or square represents a node, and an arrow represents an arc connecting two nodes.

In order to represent projects, or military missions as a network, the component tasks, events, and dependencies must be represented in the network graph as activities and events. Activities represent a part of the mission or plan such as a specific task that requires dedication of resources and a period of time to complete. Events, on the other hand, represent a particular instant in time at which a specific part of the plan (an activity) will start or finish. [Ref. 23]

There are two typical conventions for representing the graph nodes and arcs as events and activities. Each representation method has both advantages and disadvantages.

1. Activity on Node (AON)

In the Activity on Node (AON) convention, nodes represent activities as well as the start and finish events of an activity while arrows are used only as a means to represent interdependence between activities [Ref. 24]. The AON convention is powerful in its simplicity as all relevant information is contained within the nodes of the graph without reliance on information in both the arcs and the nodes which requires more detailed computations and tracking.

An example of an AON activity is shown in Figure 1 with many of its characteristic fields including the earliest start time (EST), the latest start time (LST), a description, the length or duration of the activity, and total time available to shift without disrupting the current schedule (total float).

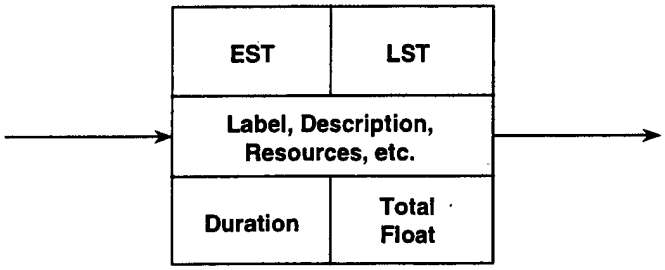


Figure 1. AON Activity [After Ref. 24]

The graph can be as detailed or as sparse as necessary to convey the required information about the mission and its component tasks. An example of a simple AON network is shown in Figure 2 indicating a project composed of two tasks where Activity A must be completed before Activity B can begin.

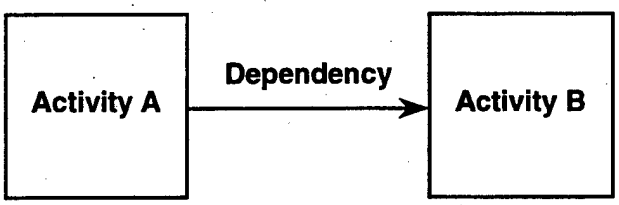


Figure 2. AON Network

The system put forward in this thesis uses the AON convention due to its simplicity in construction and representation and ease of solving. AON is also used in many commercially available project management systems.

2. Activity on Arrow (AOA)

In the Activity on Arrow (AOA) convention, arcs represent activities while nodes represent events [Ref. 24]. The need to maintain critical information on both arcs and

nodes in the AOA convention, in contrast to AON, makes AOA networks more complex and their solutions more involved. In addition, since the dependencies of activities is indicated by sequence of events, there may be times when artificial, or "dummy," activities need to be inserted in a graph to establish dependencies between events that could not otherwise be represented in the AOA convention. This need for dummy activities is a major drawback in the AOA convention, and has led to greater adoption of the AON convention, such as in this research.

An example of an AOA activity with start and finish events is shown in Figure 3. Also shown are many of the characteristic fields of both the activity and its events including the earliest event time (EET), the latest event time (LET), the duration of the activity, as well as associated labels.

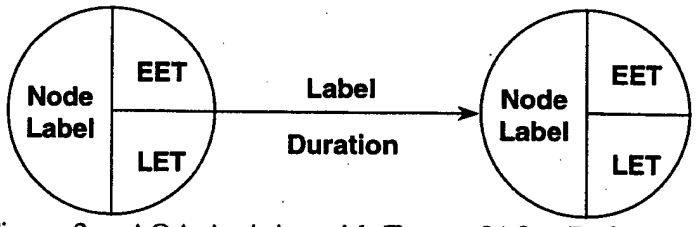


Figure 3. AOA Activity with Events [After Ref. 24]

An example of a simple AOA network is shown in Figure 4 indicating a project consisting of two activities and three events with a dummy activity required to demonstrate dependence of the completion of Activity A at Event 3 on the completion of Activity B at Event 2. The requirement for dummy activities in AOA networks often makes them much larger and more complex than their associated AON representation. This further supports the use of the AON convention in this thesis to reduce the burden on the user in designing the network representation of a military operation to be analyzed.

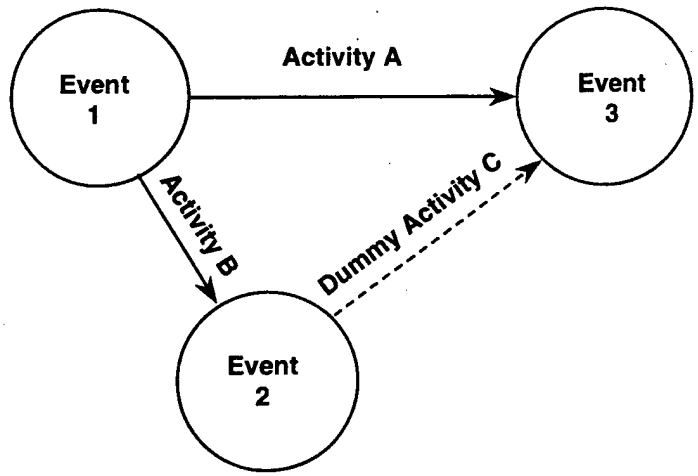


Figure 4. AOA Network

D. COMPONENTS

The idea of components as a software design methodology greatly simplifies the design of a system by allowing independent development and expansion of capability without restricting currently available or future functionality. This is a very powerful paradigm that departs from traditional design and overcomes development limitations that hinder many, if not all, of the current military planning and support systems, as well as many similar-in-function commercially available systems.

1. Characteristics

Components are small software programs or objects that perform specific functions designed to operate easily with other components and larger applications. These components must have well-designed standardized interfaces so that they can interact seamlessly with any other components or programs that also meet the same interface requirements. [Refs. 11, 27, 28] The power of components, when implemented properly, is their ability to perform their designed function without any knowledge of or interest in the other components or applications that they interact with. This

independence eliminates "hard-wiring" which limits usefulness and functionality, as well as prevents separation from a parent application and reuse elsewhere. Some examples are a map display tool component and a network-solving algorithm component.

2. Loosely Coupled Component Architecture

A group of faculty and students at the Naval Postgraduate School, spearheaded by Professors Gordon H. Bradley and Arnold H. Buss, have designed and begun implementation of a logical extension and architectural interpretation of the component-based methodology for software development called the Loosely Coupled Components (LCC) Project. This project focuses on leveraging the power of a highly flexible component architecture to support the rapid development and construction of military planning and analysis tools and systems that will operate seamlessly over extensible networks on heterogeneous computing hardware and software systems. [Ref. 29]

The architecture provides a framework for the independent design and creation of military planning and execution components that can be combined rapidly and inexpensively to fulfill wide-ranging operational needs and extended as necessary. The research goal of the project group is to provide answers to the call for advanced military planning and execution systems and capabilities outlined in such documents as *Joint Vision 2010* using developing COTS information technology. [Ref. 29]

A representation of the Loosely Coupled Components Architecture is shown in Figure 5. The process of design and use of components is a continuous cycle conducted as requirements arise. During planning for a mission, or as a crisis develops, military analysts can pull together existing components such as maps, presentation tools, and

algorithms, and then develop and incorporate completely new and independent components that might include additional models and tools in support of this new and specific requirement. Then, when the next mission or crisis arrives, due to their flexible design, the components that were created previously can be reused.

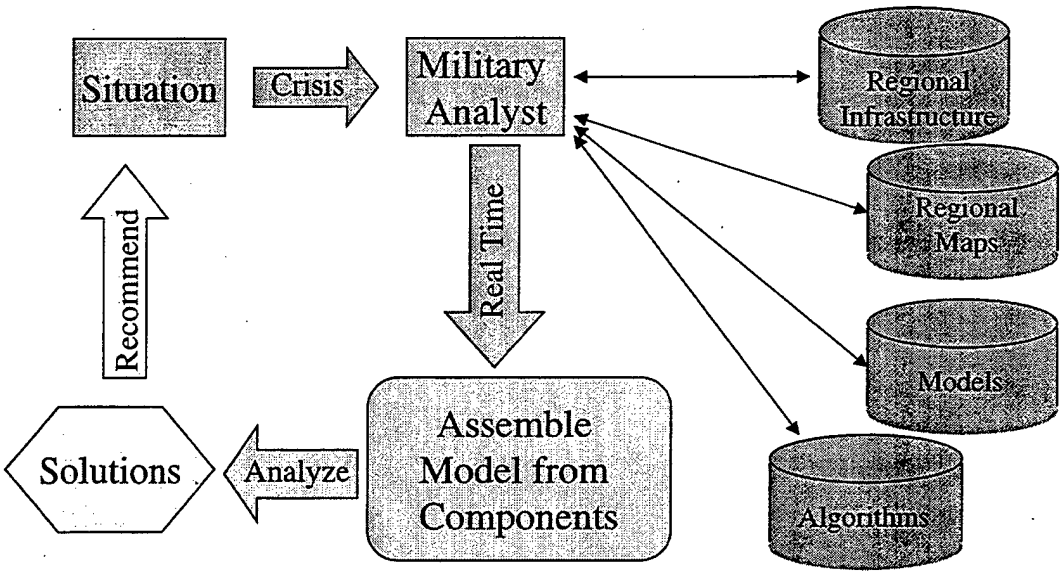


Figure 5. Loosely Coupled Components Architecture [After Ref. 30]

This architecture offers significant advantages over current and legacy military planning and analysis tools that are static, monolithic, inflexible, and in many cases proprietary. Although there are ongoing efforts to integrate legacy systems, the enhancements these provide are insufficient to provide the interoperability, platform independence, and flexibility required of desired systems, such as MPARE, that can be provided using the Loosely Coupled Component Architecture. [Refs. 9, 29]

Several currently available components include tools for conducting discrete-event simulations, map-based planning, and network and graph theory design and analysis [Refs. 9, 29]. There are many other components in development, including this thesis research, that not only add additional independent capabilities, but leverage the

power of existing components by integrating several components to tackle more complex problems.

The Java [Ref. 31] programming language was selected to implement the Loosely Coupled Components Architecture because it offers many advantages over other available languages. Java is an excellent object-oriented programming (OOP) language that incorporates support for components and for embedded networking and internet protocols that allow seamless incorporation of distributed and collaborative capabilities that are essential to the next generation of military planning and execution systems. Java also provides capabilities in platform independence and dynamic loading that dramatically increase the capability and applicability of traditional computer systems to support advanced functionality applications beyond anything currently available. [Refs. 9, 29]

3. König

König [Ref. 32] was developed by MAJ Leroy A. Jackson of the TRADOC Analysis Center-Monterey (TRAC-Monterey) as an application programmer interface (API) to provide a set of graph and network objects and algorithms to model and solve problems in a loosely coupled component framework. König components offer significant capabilities for real-time dynamic, distributed analysis due to their loosely coupled design. König objects can represent complex network and graph structures with numerous associated attributes that can be dynamically added or modified. Additionally, the König API defines a component framework for implementation of network algorithms that can be used to act on the network objects to conduct detailed analysis.

Several sample algorithms are implemented including Dijkstra's shortest path algorithm, Kruskal's minimum spanning tree algorithm, and a maximum flow labeling algorithm. [Ref. 33] An example of the output provided by the Dijkstra's shortest path algorithm on a sample König graph is shown in Figure 6.

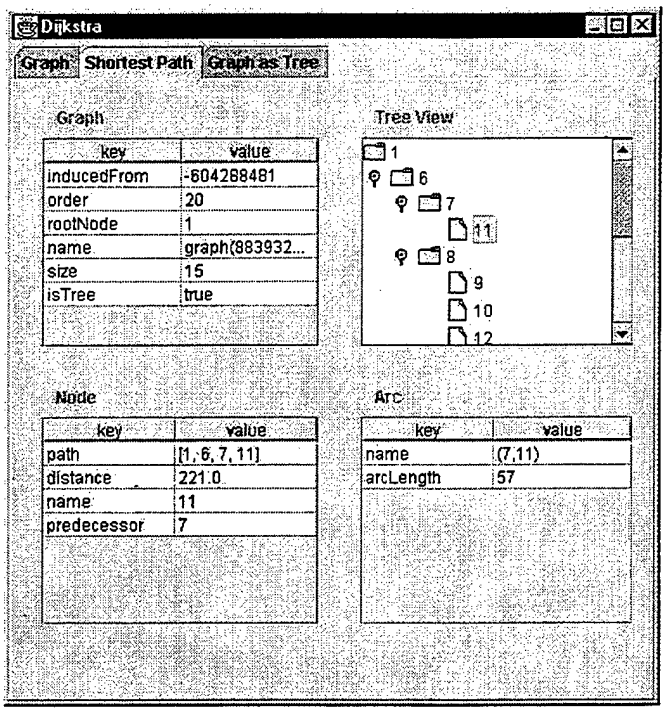


Figure 6. König Dijkstra Shortest Path Algorithm Output

This thesis attempts to take advantage of many of the capabilities of König as a design framework beyond the basic but powerful functionality provided.

4. Thistle

CPT Norbert Schrepf, a German Army officer, developed another set of loosely coupled components collectively called Thistle [Ref. 34] in support of his thesis, a *Visual Planning Aid for Movement of Ground Forces in Operations Other Than War* [Ref. 35]. The Message Center component of Thistle provides a fundamental distributed communication capability for all loosely coupled components to share information in a

simple fashion. Thistle also provides a dynamic map and overlay display tool called Flora that is extremely powerful for planning and monitoring military operations in accordance with accepted doctrinal symbology. There are also several other extremely useful tools that complement and leverage the power of other loosely coupled components and provide a common display framework for not only mapping information, but also any type of type of data that can presented in a visual manner. [Ref. 35] An example of Flora used as a map display tool is shown in Figure 7.

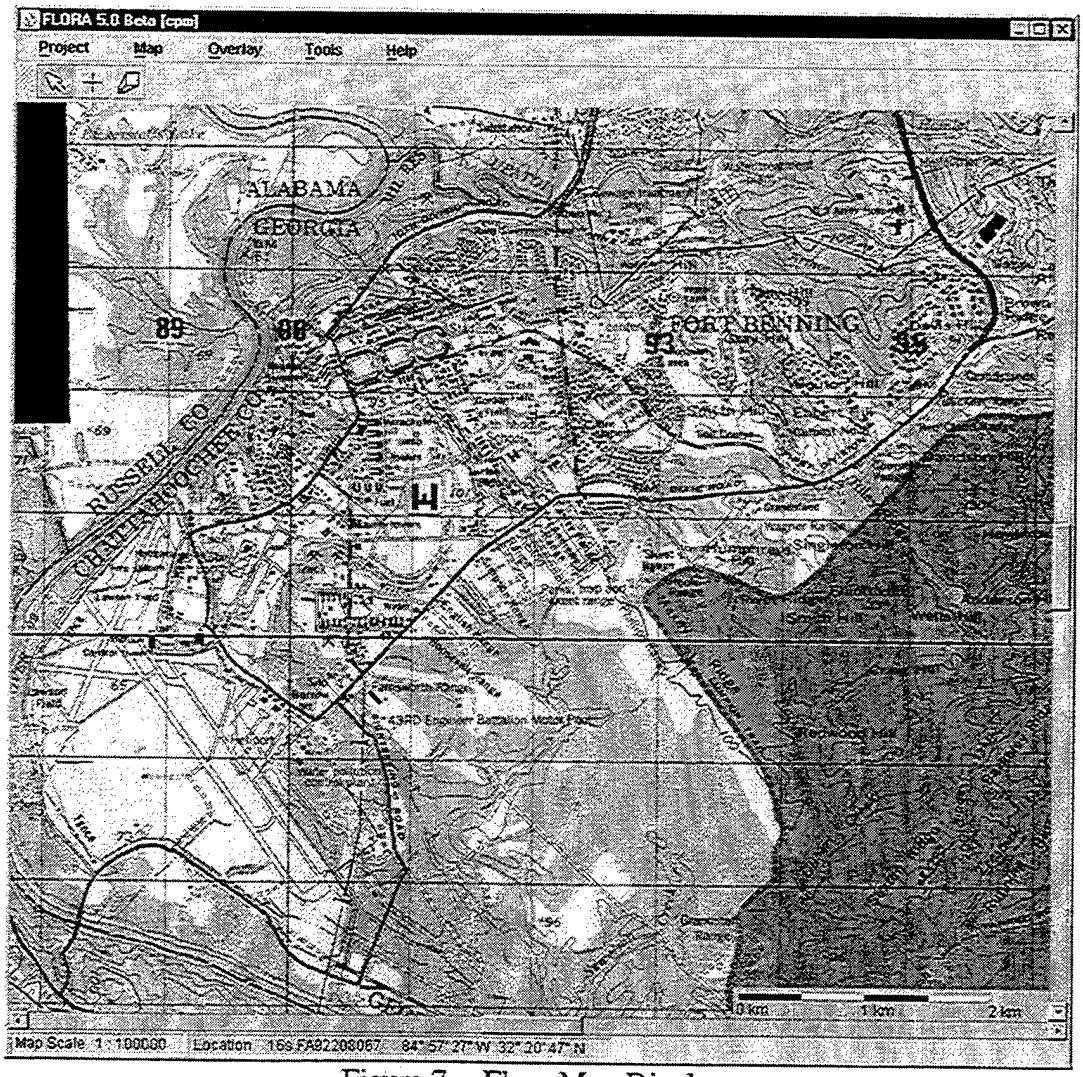


Figure 7. Flora Map Display

As with König, this thesis attempts to leverage the power of already existing loosely coupled components such as Thistle to provide even greater capabilities.

E. ALGORITHMS

Solving for the critical path in a network is essentially looking for the longest path in both directions between the start and finish points of a project to determine where there is no flexibility of movement, or slack, in either direction. The result of the forward pass longest path solution from the start point to an event represents the earliest event time (EET) of that event and the earliest start time (EST) for any activities originating at that event. Similarly, the result of the backward pass longest path solution from the finish point to an event represents the latest event time (LET) of that event and the latest finish time (LFT) for any activities culminating at that event. [Refs. 22, 36, 37]

The longest path can be solved as a linear programming problem as shown in Figure 8 [Ref. 22] using standard linear optimization techniques, or more efficiently using the network algorithms described later.

Maximize: $$\sum_{\{j:(i,j)\in A\}} c_{ij} \cdot x_{ij}$$

Subject to: $$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} = \begin{cases} -1 & i = s \\ 0 & \forall i \in N - \{s,t\} \\ 1 & i = t \end{cases}$$

$$x_{ij} \geq 0 \quad \forall (i,j) \in A$$

Define:

$G(A,N)$	graph G composed of the set of arcs A and the set of nodes N
x_{ij}	binary use variable for arc (i,j)
c_{ij}	duration of arc (i,j)
s	the start node
t	the finish node

Figure 8. Longest Path Linear Program

The two key network algorithms that are needed to solve for the critical path in a project or mission are a topological sort algorithm and a longest path algorithm. The functions of both algorithms can actually be combined into one dual-purpose algorithm making an extremely efficient critical path solver that is used in this thesis.

1. Topological Sort

The topological sort algorithm is necessary in critical path analysis to verify that the network is ordered properly from mission start to finish and that there are no cycles. A cycle, where an activity would loop back to an already completed event, would only occur due to a data entry or logic error. A cycle causes an infinite duration loop preventing the mission from ever completing. The topological sort is a very efficient algorithm that can be solved in linear time proportional to the number of arcs in the network with worst case complexity of $O(|A|)$ [Ref. 36]. A representation of a simple topological sort algorithm is shown in Figure 9 [Refs. 22, 36].

2. Longest Path

The longest path algorithm involves a very simple operation where each node in a network is examined in topological order for the greatest distance between the node and all of the nodes that occur before it, its predecessors, using a series of pair-wise comparisons leading back to the start node. In each pair-wise comparison, the larger of the current longest path associated with the node or the sum of a predecessor node's longest path and the length of the arc connecting them will become the new longest path for a particular node. As with the topological sort algorithm, the longest path in a

network can be solved in linear time proportional to the number of arcs in the network, again with worst case complexity of $O(|A|)$ [Ref. 36].

As mentioned previously, a topological sort algorithm can easily be combined with a longest path algorithm to form a more efficient joint algorithm. This joint algorithm is the preferred network solution method when a network has not previously been topologically ordered. A representation of the joint topological sort and longest path algorithm is shown in Figure 10 [Refs. 22, 36].

```
algorithm TopologicalSort;
data G(A,N), the graph G composed of the set of arcs A and the set of nodes N
begin
    for ∀i ∈ N do indegree(i) ← 0;
    for ∀(i, j) ∈ A do indegree(j) ← indegree(j) + 1;
    LIST ← Ø;
    next ← 0;
    for ∀i ∈ N do
        if indegree(i) = 0 then LIST ← LIST ∪ {i};
    while LIST ≠ Ø do
    begin
        select a node i ∈ LIST;
        LIST ← LIST − {i};
        next ← next + 1;
        order(i) ← next;
        for ∀(i, j) ∈ A(i) do
        begin
            indegree(j) ← indegree(j) − 1;
            if indegree(j) = 0 then LIST ← LIST ∪ {j};
        end;
    end;
    if next < n then G(A,N) contains a directed cycle
    else G(A,N) is acyclic and LIST contains a topological ordering of nodes;
end;
```

Figure 9. Topological Sort Algorithm

3. Critical Path Method

The critical path of a project can be easily solved using a joint longest path and topological sort algorithm on the forward pass through the network and a standard longest path algorithm on the backward pass. The results of these operations on a network representation of a military mission will provide all information necessary to conduct synchronization planning and analysis.

```
algorithm LongestPath with TopologicalSort;
data G(A,N), the graph G composed of the set of arcs A and the set of nodes N
begin
    for ∀i ∈ N do
    begin
        indegree(i) ← 0;
        longestpath(i) ← −∞;
    end;
    for ∀(i, j) ∈ A do indegree(j) ← indegree(j) + 1;
    LIST ← Ø;
    next ← 0;
    for ∀i ∈ N do
    begin
        if indegree(i) = 0 then
        begin
            LIST ← LIST ∪ {i};
            longestpath(i) ← 0;
        end;
    end;
    while LIST ≠ Ø do
    begin
        select a node i ∈ LIST;
        LIST ← LIST − {i};
        next ← next + 1;
        order(i) ← next;
        for ∀(i, j) ∈ A(i) do
        begin
            indegree(j) ← indegree(j) − 1;
            longestpath(j) ← max{longestpath(j), longestpath(i) + duration(i, j)}
            if indegree(j) = 0 then LIST ← LIST ∪ {j};
        end;
    end;
    if next < n then G(A,N) contains a directed cycle
    else
    begin
        • G(A,N) is acyclic;
        • LIST contains a topological ordering of nodes;
        • longestpath(i) represents the longest path from the start node to node i;
    end;
end;
```

Figure 10. Longest Path Algorithm

III. SYSTEM DESIGN

> To a conscientious commander, time is the most vital factor in his planning. By proper foresight and correct preliminary action, he knows he can conserve the most precious elements he controls, the lives of his men. So he thinks ahead as far as he can. He keeps his tactical plan simple. He tries to eliminate as many variable factors as he is able. He has a firsthand look at as much of the ground as circumstances render accessible to him. He checks each task in the plan with the man to whom he intends to assign it. Then — having secured in almost every instance his subordinates' wholehearted acceptance of the contemplated mission and agreement on its feasibility — only then does he issue an order. [Ref. 38: p. I-1]
>
> General Mathew B. Ridgway
> *The Korean War*

A. REQUIREMENTS

The Special Operations Mission Planning and Analysis System (SOMPASS) has been designed to address the needs of USSOCOM and Special Operations Forces. This and similar systems that embrace the goals of MPARE will be useful to the DoD and all the Services due to their inherent "jointness." This is especially important for USSOCOM because it must deal with all of these disparate players on a continuous basis. The system is required to:

- Execute on any hardware platform used by SOF
- Provide on-the-fly incorporation of new programs and capabilities
- Operate across any network connection to all participants at all levels
- Execute quickly and efficiently on devices with limited memory and storage
- Expand easily to support additional users and processing requirements as needed

While this system alone cannot answer all the needs of MPARE, it is designed to be an integral component in a larger system that will provide military decision-makers the tools to achieve the goals of Ridgway's conscientious commander.

B. CAPABILITIES

SOMPASS is designed to meet the requirements outlined above and provide not only a demonstration of capabilities to foster further development, but also a working product that can be used to solve current problems. As with all other loosely coupled components, this system is written using the Java 2 programming language [Ref. 31] developed by Sun Microsystems. Several requirements are achieved by implementing the system in Java; others are achieved by using the Loosely Coupled Components design methodology.

1. Platform Independent

Platform independence is achieved by using the Java programming language. A significant capability of Java and an advantage over many other programming languages is its inherent platform independence. Programs can be written and compiled once on one computer platform and then be executed without modification on a number of other computers and operating systems. Currently, Java programs execute on the computers and operating systems listed in Table 1. Additionally, Sun Microsystems has made the source code publicly available so that other developers may port it to additional operating systems and platforms. [Refs. 39, 40] This ability to operate on any hardware system without modification, from the largest mainframe to the smallest handheld device, offers incredible power and flexibility.

Java provides this portability by compiling programs into "byte code" that is not processor specific. The byte code is interpreted by a program running called a Java Virtual Machine (JVM) that executes on each host computer.

Operating System	CPU	Company
Windows NT/9x	Intel	Various
Solaris	SPARC	Sun Microsystems
AIX		IBM
DG/UX 4.2	Intel	Data General Corporation
DIGITAL OpenVMS	Alpha	Digital Equipment Corporation
DIGITAL Unix	Alpha	Digital Equipment Corporation
HP-UX		Hewlett-Packard
IRIX		Silicon Graphics
Linux	Intel	Blackdown.org
Linux	MkLinux, P-Mac, PPC	Tyler
MacOS	Motorola	Apple
NetWare		Novell
OS/2	i386	IBM
OS/390, OS/400		IBM
SCO	i386	SCO
UnixWare	i386	SCO
VxWorks		Wind River Systems
Windows NT	Alpha	Digital Equipment Corporation

Table 1. Java Supported Platforms and Operating Systems [After Refs. 39, 40]

The JVM converts the byte codes into native instructions that are executed on the host computer. Once a Java Virtual Machine is implemented for a particular processor and operating system, it can then execute any compiled Java program. [Refs. 9, 29, 41, 42] Sun Microsystems promotes the promise of the Java platform as "Write Once, Run Anywhere" to highlight the reductions in effort required in fielding new software that would otherwise have to be implemented separately on numerous platforms to provide equivalent functionality [Ref. 43]. In addition to the ability to run programs on any platform, a more recent capability was developed to provide a set of platform independent graphical user interface (GUI) controls, collectively known as Swing, to allow a common visual appearance across platforms [Ref. 44]. This common appearance will promote ease of use and a reduction in training requirements since a person who is accustomed to using a Swing-based application on one platform will not have to relearn

the functionality if he must use the same application on a different platform; the programs will be identical in appearance.

USSOCOM and SOF units use different computer systems with different processors, capabilities, and operating systems. In fact, these units have the most diverse range of systems within the DoD due to their unique missions and requirements. As such, they have the greatest challenge to interoperate and intercommunicate, not only among themselves, but also with external agencies, forces, and nations. Programs written in Java avoid this obstacle without any effort by the designer or user. [Refs. 9, 11, 29]

This system, along with other components, offers the ability to provide cross-platform functionality so that users working on a desktop computer at a home-station location or at a forward operational base (FOB) can operate identically to an element operating on a notebook computer or other small-scale computing device running the same applications without modification or adaptation from a forward operating location (FOL) or in a hostile environment. Users can become proficient at operating this system in a training or administrative environment on a standard system, and then be able to put that proficiency to task when they move to a smaller device for tactical deployments and operations without the need for retraining.

2. Dynamic Loading and Viewing

Mission planning does not end with the onset of the execution phase. Continuous monitoring, or "battle tracking," during all phases of an operation are critical to mission success, thus a capability to react immediately to changing or unforeseen events is critical to any military decision support system. Java provides another incredibly powerful

capability, called "dynamic loading," that supports this critical operational need. [Refs. 29, 41] The loosely coupled component architecture used by this system builds on Java's dynamic loading and execution to provide enhanced capabilities that offer real-time updates to software and capability enhancements, even after the system has been started. For example, a new algorithm or capability needed at a remote location can be written and compiled elsewhere, sent to the unit in need across any available network connection, and then incorporated into ongoing planning or analysis seamlessly to solve a problem without any need to restart the system. [Ref. 29]

Additionally, the system put forward in this research can receive updated information about units or items of interest and display those changes real-time, without any action required by the user. This is extremely relevant when dealing with mission areas such as intelligence, surveillance, and reconnaissance, as well as traditional battle tracking. The Loosely Coupled Component design of the system allows for dynamic viewing of information that can be coordinated with other components such as Thistle and König. [Refs. 33, 35]

3. Distributed

Special Operations Forces usually operate in diverse and disparate environments; the ability to have access to and share critical mission information from remote locations offers a significant advantage over current isolated systems. Planning of these complex operations can involve numerous forces, agencies, and activities whose actions and efforts must be coordinated to achieve their objectives, yet are usually unable to consolidate at a single location, thereby necessitating the system to operate in a

distributed manner. [Refs. 9, 11] The network-focused nature of Java allows for simple sharing and distribution of data and programs required to operate this system. This capability allows the system to inter-operate with different components and elements of data that may be physically in different locations. Some resources may be on the local system, while others might be accessed across a network connection from a system that is on a different continent.

Java's distributed computing support functionality goes beyond merely providing on-demand sending and receiving of data and components across networks. It also is able to execute programs and functions on distant computers through remote method invocation (RMI). Another technology, the Common Object Request Broker Architecture (CORBA), is supported in Java to provide standards-based interoperability and connectivity so that even non-Java programs and applications can work together in a distributed manner. Together, Java's RMI and CORBA support offer tremendous potential to enhance all loosely coupled components, including this system, with distributed computing capabilities to enhance operational efficiency and effectiveness. [Refs. 9, 11]

4. Thin Client

Another advantage of this system is that it can operate as a thin client. The idea behind thin client applications is to allow clients running applications and using services to be extremely small while the bulk of the computational requirements and data processing activities occur on a powerful supporting server [Ref. 45]. Since in many military and most Special Operations environments the computing devices are much

smaller and less powerful than the platforms available to planners in a home-station environment, an application must be small enough so that it can run efficiently on these smaller field devices while providing the same capabilities and availability of information to the user. Each user need only load the elements of the application that are relevant to support the mission at hand, thereby further reducing memory and storage requirements while always retaining the flexibility to retrieve additional elements and capabilities should the situation or mission change. This is a capability that is impossible to achieve with existing applications. [Refs. 9, 11, 29]

5. Extensible

Other services and government agencies, forces of other nations, and non-governmental organizations (NGOs) often play key roles in special operations missions, most notably operations other than war (OOTW). SOMPASS is designed to support these complex command and control chains by offering extensibility provided by its loosely coupled component architecture. This capability allows for great flexibility in response and support for growth in complexity and scale. Other components may be designed and integrated with this system as needed, or all may operate independently, whatever the situation dictates. In addition, the components are specifically designed to be reusable without modification and adaptable to different situations that require similar capabilities. As mentioned before, components can be accessed over available networks and incorporated on the fly, allowing for tremendous power in handling problems that arise during any phase of an operation. This system can support a large number of clients that may grow or shrink during an operation, depending on the phase, each with their

own needs, yet all provided with the view and analysis of available information supporting their specific requirements. Many current systems do not offer this flexibility of support, thereby reducing their usefulness and restricting the potential capabilities of their users. By being extensible, this system also has the capability to integrate many participants that cannot currently receive and update mission information due to hardware, communications, and other restrictions. [Refs. 9, 11, 29]

C. MODEL

"The essence of the operations research activity lies in the construction and use of models." [Ref. 21: p. 4] Models are just simplified representations of real systems that provide the means to solve complex problems [Ref. 21]. The system put forward in this thesis uses the theory and foundations put forward in Chapter II to implement a usable product that can assist commanders and staffs in planning, analysis, and decision-making by providing a set of tools that allow the construction of models of military operations as networks and applying solution algorithms and display components to simplify the complex task of mission synchronization.

1. Network Representation

A military operation is represented in this system as a network contained within a König graph object. The graph is composed of numerous nodes and arcs that are entered by planners to represent the different elements of the operation from inception to completion along with the dependencies that define the relationships between the elements. König has no notion of a visual representation of a graph so SOMPASS contains tools that convert the nodes and arcs of a graph into representations of the nodes

and arcs that are understood by and can be displayed on a Flora map display component to allow a view of the network and simplify manipulation and modification of its elements.

This system uses an AON representation for simplicity and ease of use to both eliminate the need for incorporating dummy activities in the network and reduce the computational requirements since all attributed associated with events and activities are contained within the nodes. Additionally, a CPM approach, rather than PERT, is used to solve a graph for its critical path and develop the synchronization matrix and execution checklists. CPM does not rely on complex probabilistic distribution assumptions. PERT requires the user to provide probability distributions that might yield erroneous or misleading results if improper parameters for the distributions are used.

Since the system is dynamic and distributed, changes in the military operation that occur at any time can easily be made to the graph, its nodes and arcs, or the properties associated with them, either automatically, or by any user from any location that is connected to the system. Then all participants and monitors will automatically receive and incorporate these changes in their systems without any additional effort.

2. Tasks and Properties

After receiving a mission, commanders and their staffs have many responsibilities during the planning process, regardless of whether it will be the deliberate or compressed planning sequence. Some of these responsibilities include breaking down a mission into its specified, implied, and essential tasks; identifying critical decision points, developing a concept of the operation and COAs; wargaming the COAs; and producing the OPORD.

In this system, an operation is broken down into its component elements: the tasks that participating units must accomplish, and the events that identify the phases and states of progress. These tasks and events come from the planning process and must then be input into the system. Each task that must be accomplished by a unit in support of the mission is translated to an activity of the model that is represented as a node on the graph since this system uses an AON representation. These nodes must also contain information that represents the beginning and end events of each activity.

Each node on the graph has many important pieces of information that must be associated with it in order to be useful in identifying it as well as solving for the critical path of the operation and presenting the results in a useful manner. König provides a means to easily store, modify, and reference these critical elements of information with a graph and its nodes and arcs through the use of object property references. These properties associated with each element of a König graph can be dynamically added, modified, queried, and removed as needed providing very powerful capabilities for graph manipulation. [Ref. 33] Some examples of mission activity properties are the name of the responsible unit, an associated location, time required to complete the task, equipment involved, and whether the activity is on the critical path of the operation. As stated before, the nodes of an AON graph must also contain the events that begin and end each activity and this information can also be stored as properties of the nodes.

The arcs in a graph on these AON networks serve only to connect the nodes of events and activities to show dependency for activity completion and may be assigned a property by the system as to whether an arc is on the critical path. By nature of König,

the arcs may also be assigned other dynamic properties, but this system is only concerned with arcs for precedence. Properties may also be added to the graph itself, and several are added by the system to provide information such as the total duration of the mission.

This system takes full advantage of the power in König to maintain dynamic property attributes. This system introduces a set of tools that allow the user to easily add, edit, view, or delete these properties in a graphical manner.

D. ARCHITECTURE

In order to develop a useful and powerful mission planning and analysis support system, there must be an underlying architecture that is well designed and structured. This system provides that necessary architecture by building to and incorporating the loosely coupled component methodology and providing a simple and user-friendly graphical user interface. In addition, this system also subscribes to and takes advantage of another powerful methodology that is implemented in Swing, the Model-View-Controller (MVC) architecture.

1. Loosely Coupled Components

This system is composed of several newly developed loosely coupled components and incorporates existing components. One of the new components provided is the CPM solver algorithm and supporting functions. Since it was designed around the loosely coupled component architecture, it can be used in conjunction with this system, used in several previous planning system tools, or used in tools that may be developed in the future. It can be used with other König algorithm components, or it can be used alone.

Another component developed for this system is a set of visual graph design and editing tools that work together with other loosely coupled component sets, namely König and Thistle, to rapidly build and manipulate the operational networks required for conducting critical path analysis. These tools can also be used outside this system for other generic or specific tasks involving the construction or modification of networks in a visual manner in other loosely coupled systems.

Also, a graphical user interface component was built to tie together the critical path solver algorithm with the graph design tools and present the system specific outputs: the mission synchronization matrix and execution checklists. Together, these components provide a useful and powerful mission planning and analysis support system that is dynamic, flexible, and component based.

2. Graphical User Interface (GUI)

The GUI developed for this system was designed to be simple and user-friendly so that it could be easily learned and accepted by users of all experience levels and backgrounds. In this manner, simplicity would speed up familiarity so that users will accept it as a useful tool. The GUI of any application is usually the most difficult component to develop successfully, and it is the most critical for acceptance. If data entry is cumbersome or confusing, errors will be common and user frustration will be high, leading to reduced productivity or avoidance. Similarly, if information is presented poorly, then the value of that information is degraded since it cannot be easily understood or acted upon. Military personnel are extremely demanding of their equipment and use it under greatly varying adverse conditions, much more so than their civilian counterparts.

Consequently, any military planning or execution system must provide a superior interface, as well as useful tools to be accepted and adopted. This system attempts to provide such an interface.

3. Model-View-Controller (MVC)

The Model-View-Controller architecture makes interactive systems more loosely coupled and provides for greater scalability and easier maintenance or adjustment. Each of the elements can be modified independently later without requiring changes to any of the others. [Refs. 44, 46] A representation of the interactions involved in the MVC architecture can be seen in Figure 11.

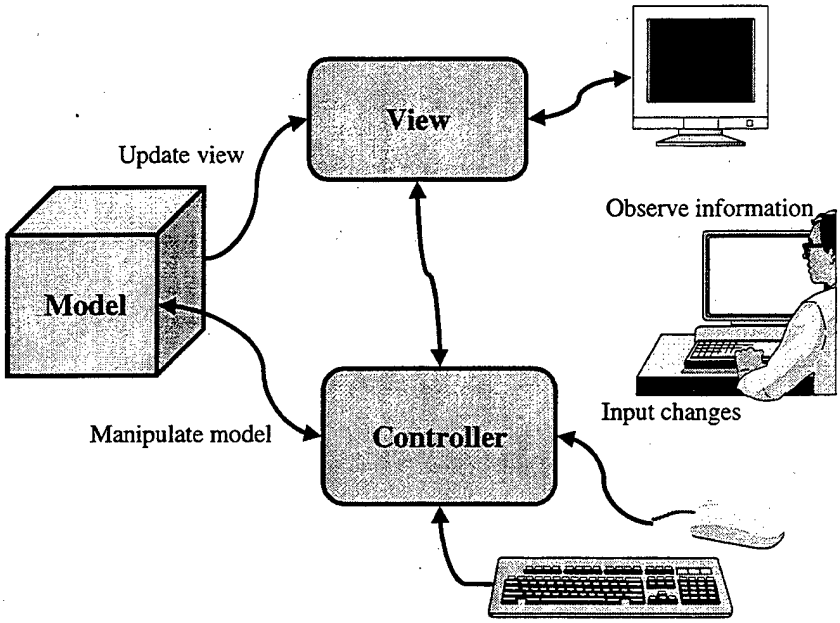


Figure 11. Model-View-Controller Architecture

The model contains the underlying data and performs all computations and adjustments as necessary. The view displays the appropriate representation of the data from the model. And the controller responds to all user actions and notifies the model and view when changes occur. The controller and view also work tightly together so that the user

can see changes from the actions directed by the controller that do not get sent to the model, such as object selection and manipulation. [Ref. 46] Swing implements the MVC architecture in its graphical display and control components to also take advantage of its tremendous power and flexibility [Ref. 44].

The MVC architecture is also used in this system to offer greater flexibility and control over operation and interaction. In this system, the König graph of a mission and critical path solver operate as elements of the model, Flora acts as the view, and the mouse, keyboard, or network act as the controller. All components communicate through the Thistle Message Center to receive and post their updates providing a very dynamic structure. Figure 12 highlights the loosely coupled component architecture as well as the MVC design incorporated in this system.

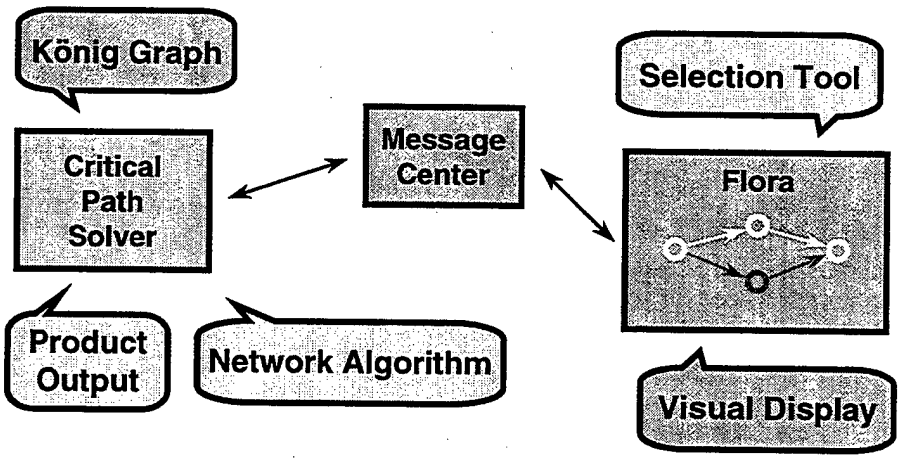


Figure 12. System Architecture

E. IMPLEMENTATION

1. Description

The system presented in this research is designed to provide SOF commanders and staffs a tool that will assist them in mission planning and analysis by reducing the

time and effort required to produce operational support documents including the mission synchronization matrix and unit execution checklists. Additionally, it will allow dynamic adjustment and updates to facilitate wargaming and sensitivity analysis of COAs as well as highlight potential bottlenecks during the rehearsal and execution phases if linked to real-time operational data.

This system is written entirely in Java due to the many advantages and features detailed previously. This system also takes advantage of the powerful graphical display and control components of Java's Swing for its GUI to ensure a standard cross-platform appearance and provide functionality not available with other languages or design tools.

The actual Java code for the majority of components put forward in this system is not included in this thesis report, but is available from the Operations Research Department at the Naval Postgraduate School or the author. However, the code for the implementation of the critical path solver algorithm that was presented in Chapter II is provided in Appendix A to give the reader an idea of how a loosely coupled component is structured, and how an algorithm is translated from pseudo-code to an executable program. The code for this component also provides the reader with a view of the power of the König API to solve network flow and graph theory problems.

2. Operation

This system is designed to be user-friendly and simple to operate so that it will be accepted and used for mission planning and analysis by operational units. To that end, highlighting views of the tools available will present a simple introduction to the system and its operation. Additionally, a notional illustrative scenario of a special operations

mission is presented in Chapter IV, and used in an operational demonstration of the system to provide the reader with a view of system functionality and capability. To further aid the reader and potential users, a series of screen shots taken during program execution involving the illustrative scenario, along with detailed explanations, is included in Appendix C to provide a "walk-through" format user manual of the system. Several views of system components showing a simple example network will introduce the reader to what will be seen later in the more complex and detailed presentation.

When the main program is launched, several components are executed to allow use of the system. The Critical Path Solver control panel starts with an empty project as shown in Figure 13 and a blank Flora map display object is presented for use as a project design and display board as shown in Figure 14.

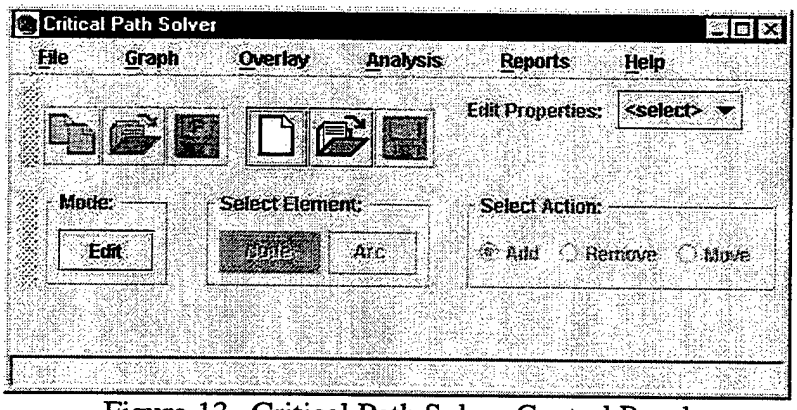


Figure 13. Critical Path Solver Control Panel

The Message Center is also activated in the background to handle system and network communications, but it cannot be seen.

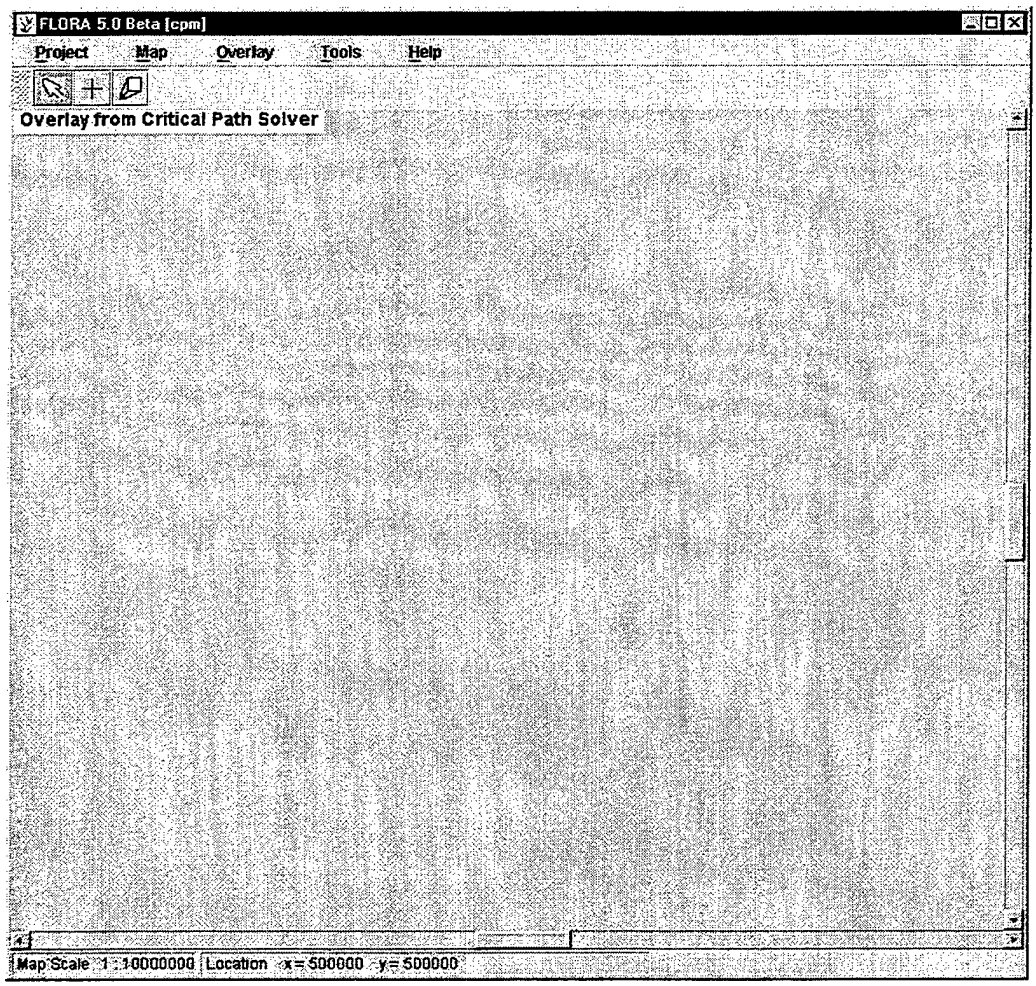


Figure 14. Blank Flora Map Display for Project Design

The graph editing tool bar is detachable, as shown in Figure 15, which makes editing the project graph easier since it contains all necessary display editing tool controls and can be displayed alongside the Flora display without interfering with editing operations.

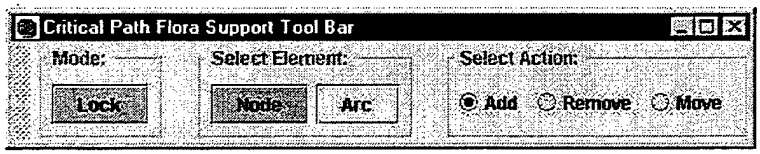


Figure 15. Detachable Graph Editing Tool Bar

There are also several components used for editing the properties associated with activities, or nodes, as well as the overall project graph. Examples of these editing

windows are shown in Figure 16 and Figure 17, respectively. These components provide a convenient way to display information about a task or the mission that may be changing due to remote or automated inputs, or allow direct manipulation of properties by the user.

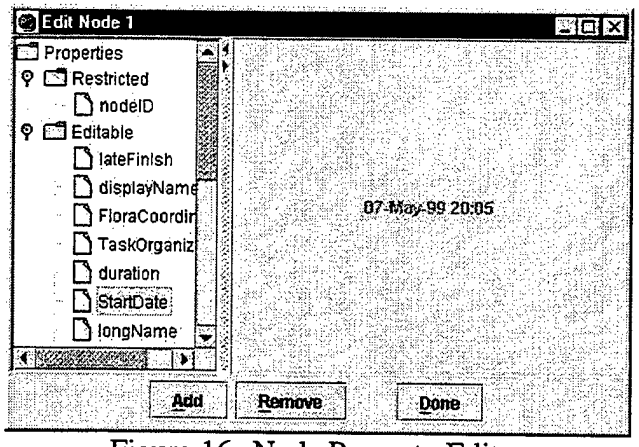


Figure 16. Node Property Editor

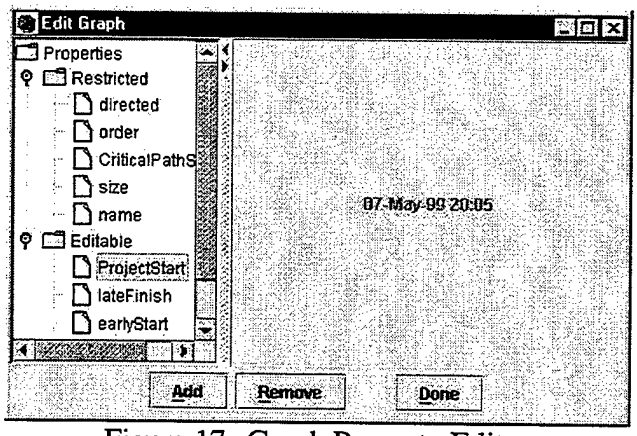


Figure 17. Graph Property Editor

Nodes are represented on the Flora map display as circles with unique reference numbers next to them, and arcs are displayed as arrows from the predecessor node to the subsequent node such as in the simple example network shown in Figure 18. In addition to the property display and editing components of Critical Path Solver, Flora provides a built-in capability to select a visible element and display some properties associated with it.

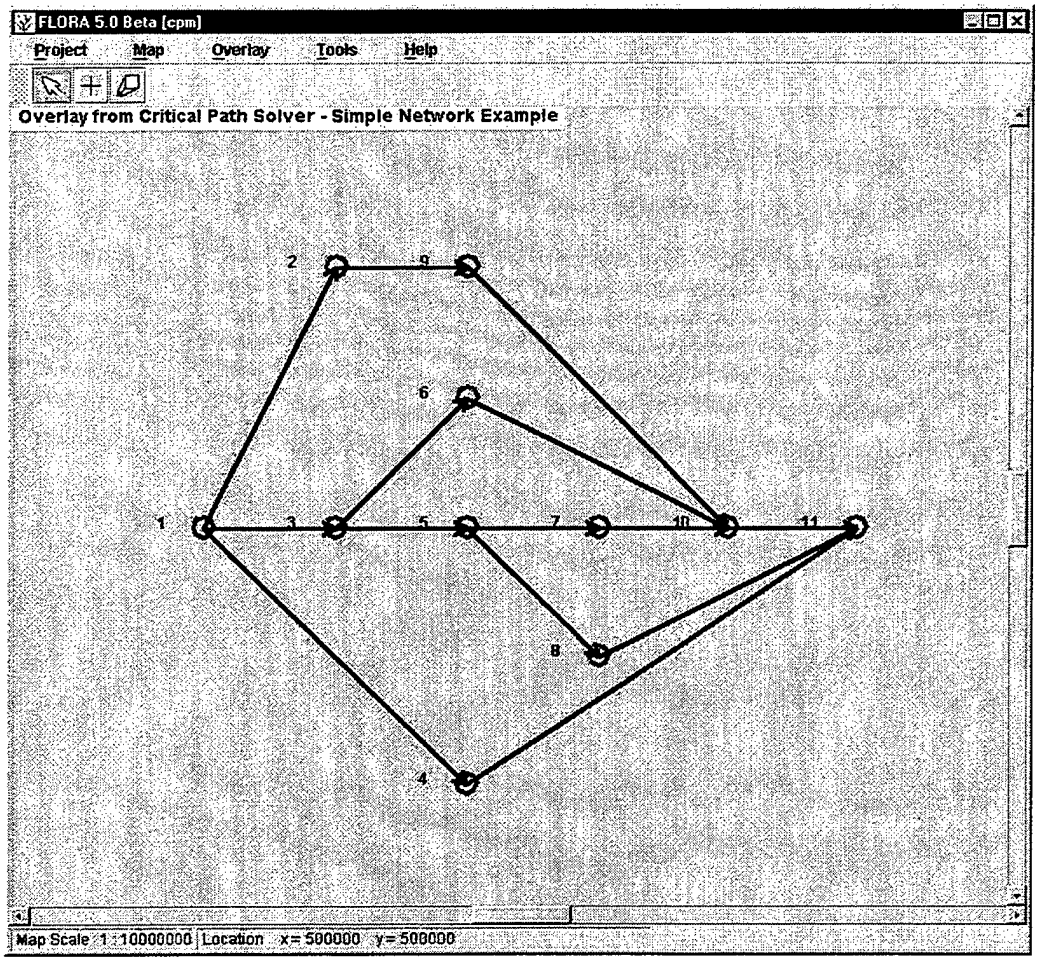


Figure 18. Simple Network Example

When Flora is operating as a map display tool such as in Figure 7 found in Chapter II, this feature is used to display unit or equipment icons, but when used with the Critical Path Solver, Flora can display some of the properties of nodes on a graph as shown in the example node in Figure 19. This feature may be useful when editing is not required. As implemented in Flora, this display method does not automatically update the display to reflect any changes in node properties. This can be done with the node property editor tool.

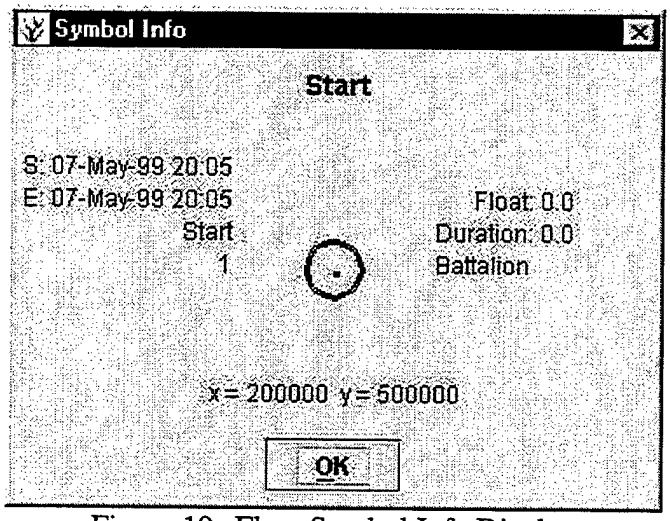


Figure 19. Flora Symbol Info Display

If a cycle is found when attempting to solve for the critical path of a project, a warning message such as in Figure 20 is displayed, and if possible to determine, the offending node is highlighted in red so that the user can take action to correct the error.

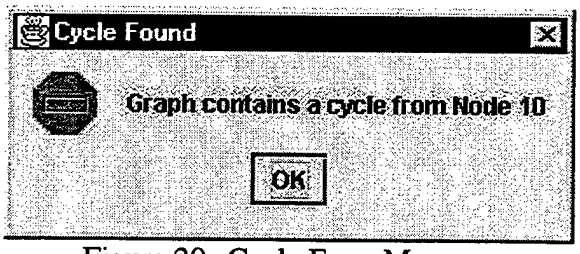


Figure 20. Cycle Error Message

If there are no cycles in the project graph, the computed critical path is then highlighted in blue, and appropriate properties are added to the graph, nodes, and arcs, and all project participants are updated automatically of the changes through the Message Center. The blue critical path highlighting is clearly visible in the on-screen display, but is difficult to discern in the black and white representation shown in Figure 21. A representation of a solved network in color can be found in Appendix C.

At any time during system operation, the current project and all of its associated elements can be saved for later action or modification, or a new project can be started, providing great flexibility to users.

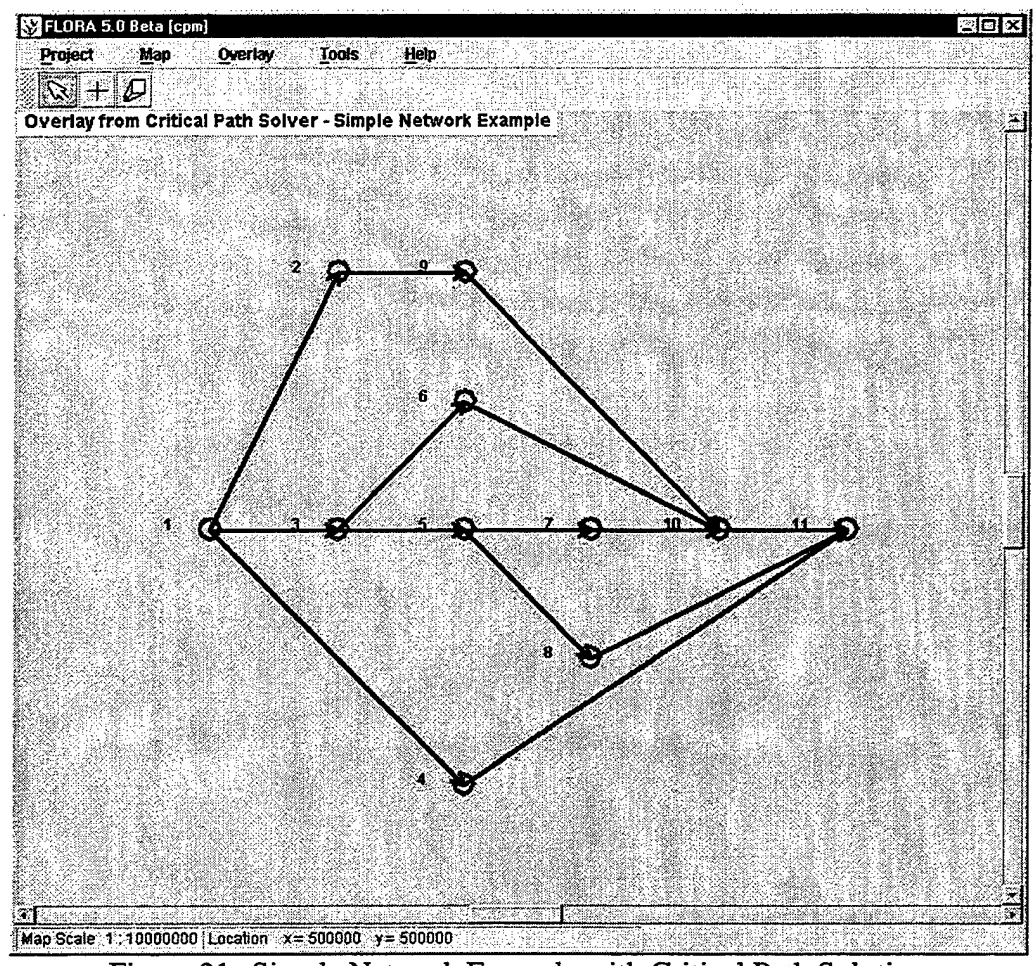


Figure 21. Simple Network Example with Critical Path Solution

3. Products

After successfully solving for the project critical path, the principal features of this system can be exercised — they produce the mission synchronization matrix and unit execution checklists.

The synchronization matrix produced by this system presents a time sequenced, unit-hierarchical view of the mission similar to one that would be produced manually during the mission planning sequence. The synchronization matrix produced by this system is more valuable than those constructed manually because of its interactive and dynamic properties. The left column displays all units involved in the mission, from the controlling headquarters at the top, down through all subordinate elements under their respective parent units. Across the top heading is a listing of critical times in the mission from start to completion. In the body of the matrix are the mission tasks and events associated with their responsible unit and the required times of action or completion. All tasks that are on the critical path of the operation have asterisks ("*") before their names in order to highlight their importance to the viewer. The highlighting helps convey this critical element of information to people who will not see the critical path on the network graph view. An example view of a synchronization matrix is shown in Figure 22.

Synchronization Matrix for Simple Network Example

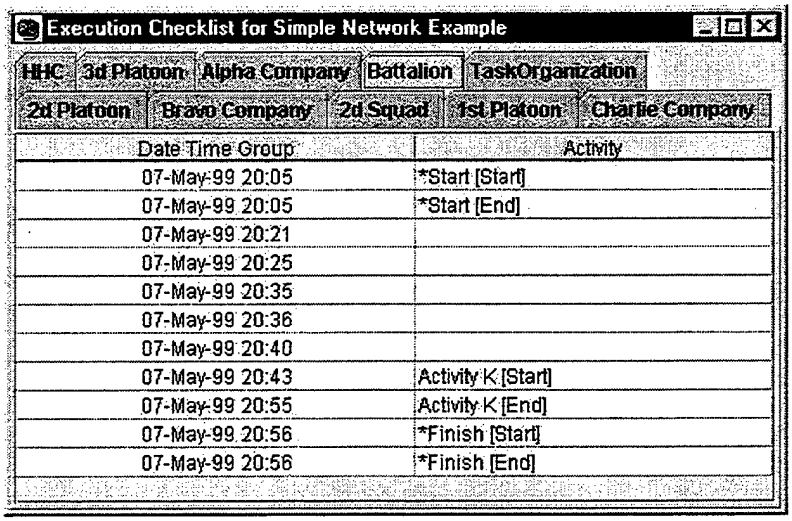
Unit	07-May-99 20:05	07-May-99 20:05	07-May-99 20:21	07-May-99 20:25	07-May-99 20:35	07-May-99 20:36	07-May-99 20:40	07-May-99
TaskOrganization								
Battalion	*Start [Start]	*Start [End]						Activity K [S
Bravo Company								
2d Platoon		Activity A [Start]	Activity A [End], ...	Activity J	Activity J	Activity J [End]		
Charlie Company		*Activity B [Start]	*Activity B	*Activity B [End]				
1st Platoon				*Activity D [Start]...	Activity E [End], ...	*Activity D	*Activity D [End]	
2d Squad							*Activity H [Start]	*Activity H
HHC		Activity C [Start]	Activity C	Activity C	Activity C [End]			
Alpha Company								
3d Platoon							Activity G [Start]	Activity G [E

Figure 22. Example Synchronization Matrix

Unit listings on the left edge can be collapsed to hide their subordinate elements so that a user can easily view the information he is concerned with for a particular unit, or group of units of interest. The table can also be scrolled to view information that is contained in

the matrix but not currently visible due to screen size limitations on whatever system it is being viewed.

The execution checklists produced by this system are presented as selectable by a unit tab across the top so that only the checklist for the unit of interest is visible to avoid clutter or confusion. The checklist is shown as a vertical timeline from earliest at the top to latest at the bottom with the time of action or completion shown in the left column, and the required task or event to the right. As with the synchronization matrix, tasks that are on the critical path of the operation are highlighted with an asterisk ("*"). The execution checklist table is also scrollable to allow viewing information not in the present window view. An example view of a unit execution checklist is shown in Figure 23.



Date Time Group	Activity
07-May-99 20:05	*Start [Start]
07-May-99 20:05	*Start [End]
07-May-99 20:21	
07-May-99 20:25	
07-May-99 20:35	
07-May-99 20:36	
07-May-99 20:40	
07-May-99 20:43	Activity K [Start]
07-May-99 20:55	Activity K [End]
07-May-99 20:56	*Finish [Start]
07-May-99 20:56	*Finish [End]

Figure 23. Example Execution Checklist

As mentioned previously, there are more detailed use and capability examples and system function explanations in Appendix C, along with color figures to aid in visualization.

IV. ILLUSTRATIVE SCENARIO

This special operations scenario is designed to demonstrate the capabilities and applicability of the mission planning and analysis system put forward in this thesis. This scenario is not based on any known actual previous or planned operations or exercises; it is entirely notional and designed to be unclassified.

A. SITUATION

This scenario is based on the conduct of a special operations emergency deployment readiness exercise (EDRE) involving a Joint Special Operations Task Force (JSOTF) in an area of operations (AO) in the vicinity of Fort Benning, in Columbus, Georgia. The JSOTF has been formed for the exercise and is headquartered at Hunter Army Airfield (AAF) in Savannah, Georgia. All participating elements are either stationed at or operating from a FOB at Hunter AAF for the duration of the exercise.

B. TASK ORGANIZATION

The JSOTF is composed of the following forces:

- JSOTF Headquarters (HQ)
- 75th Ranger Regiment HQ
- 1st Battalion, 75th Ranger Regiment
- 3d Battalion, 3d Special Forces Group (SFG)
  - 1 x Special Operations Command and Control Element (SOCCE)
  - 1 x Special Forces Operational Detachment Charlie (SFOD C/ODC)
  - 1 x Special Forces Operational Detachments Alpha (SFOD A/ODA)
- 3d Battalion, 160th Special Operations Aviation Regiment (SOAR)
  - 2 x Special Operations Aviation Detachments (SOAD)
- 4th Special Operations Squadron (SOS), 16th Special Operations Wing (SOW)
  - 2 x AC-130U Spooky II Gunships
- 20th SOS, 16th SOW
  - 4 x MH-53J Pave Low III Helicopters

The following map in Figure 24 shows the location and composition of forces in the JSOTF.

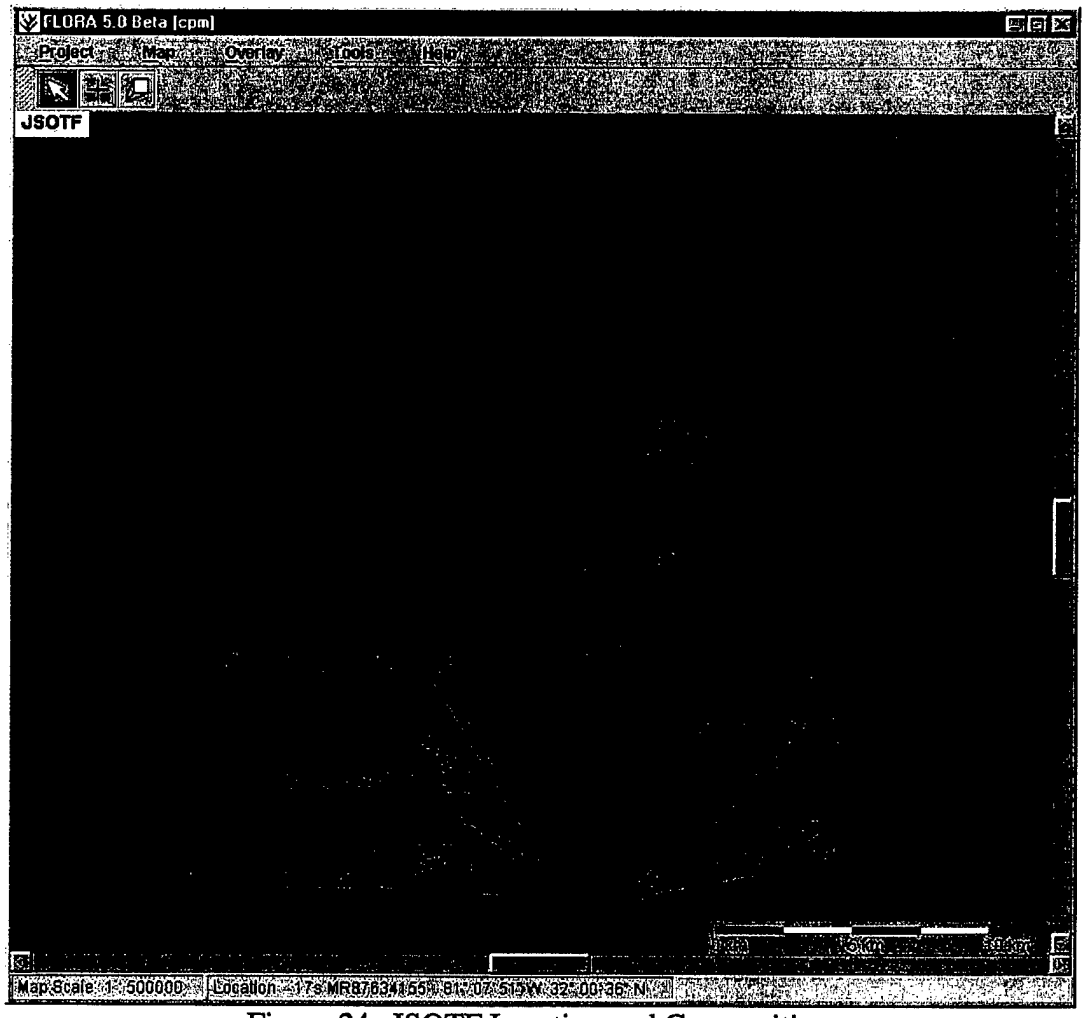


Figure 24. JSOTF Location and Composition

C. MISSION DEVELOPMENT

1. Situation

A suspected critical enemy facility has been identified by national intelligence assets and must be destroyed. The site is located in an urban environment in the vicinity of several light and motorized enemy units. The enemy units are expected to reinforce the security personnel at the facility to defend the site if it is believed to be in danger.

The following map in Figure 25 shows the mission AO with the objective highlighted in red.

Figure 25. Mission Area of Operations

2. Mission

The JSOTF will conduct special reconnaissance (SR) on the suspected facility to verify its purpose and identify defensive capabilities. If the objective is validated as an appropriate target, the JSOTF will conduct a raid to destroy the facility and collect evidence from the site to return for further analysis.

3. Concept of the Operation

An SFOD A team from 3-3 SFG will be lifted from Hunter AAF by a SOAD from 3-160 SOAR consisting of 2 x MH-60Ls and inserted into the AO to conduct SR in the vicinity of the objective, including target analysis (TA), and report back their findings to the JSOTF. If the objective is verified, a second SOAD, also from 3-160 SOAR, consisting of 10 x MH-47Ds and 10 x MH-60Ls will lift Task Force (TF) Ranger (RGR), composed of 1-75 Ranger, from Hunter AAF and insert them into the AO onto four of six potential helicopter landing zones (HLZs). The operation will be supported from the air by elements of the 20th SOS consisting of 4 x MH-53Js and elements of the 4th SOS consisting of 2 x AC-130Us. In the strike phase, the ODA will provide additional target acquisition intelligence. TF Ranger will destroy the objective, recover and collect any available intelligence information regarding the facility, and be extracted back to Hunter AAF. The map in Figure 26 shows the desired operational state for the conduct of actions on the objective during the strike phase. The ODA will extract separately after conducting battle damage assessment (BDA) of the objective and post-strike reconnaissance of the AO, and also return to Hunter AAF. After all units have returned to Hunter AAF, the JSOTF will conduct a debriefing and after action review (AAR) and then conclude the exercise.

D. MISSION TASKS, EVENTS, AND DEPENDENCE

The mission outlined in this illustrative scenario must be broken down into its component tasks and events in order to demonstrate the system capabilities and suitability. A list of the mission tasks and events, along with their associated properties

and assigned units is located in Table 2 in Appendix B. This list is not exhaustive, but sufficient to demonstrate functionality without need for classification, and serves as the foundation for the project network to be entered into this system for analysis.

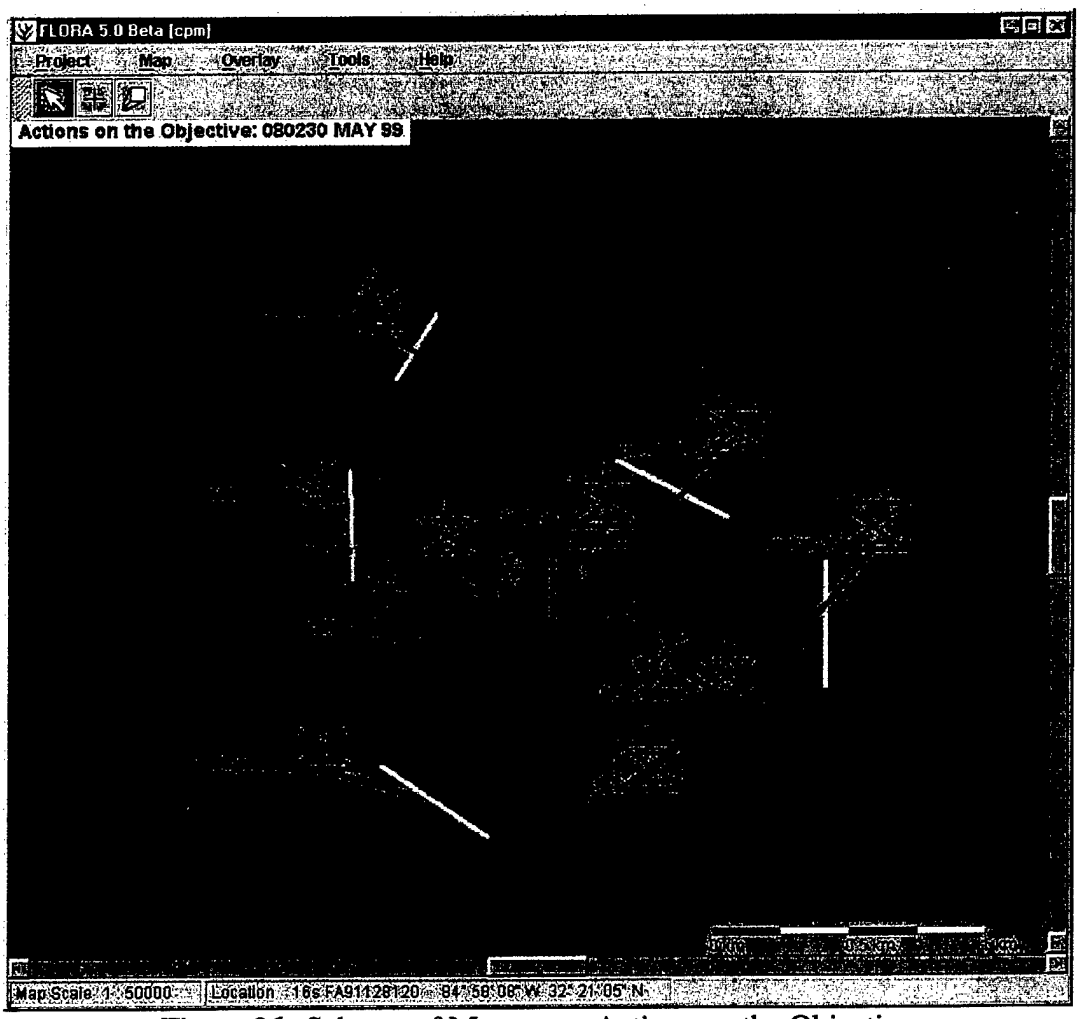


Figure 26. Scheme of Maneuver: Actions on the Objective

E. IMPLEMENTATION

The network of the illustrative scenario built with SOMPASS consisted of 23 unit elements and headquarters, 94 nodes representing the mission tasks found in Appendix B, and 186 arcs depicting the dependencies between the tasks. A view of the graph of the mission network is shown in Figure 27.

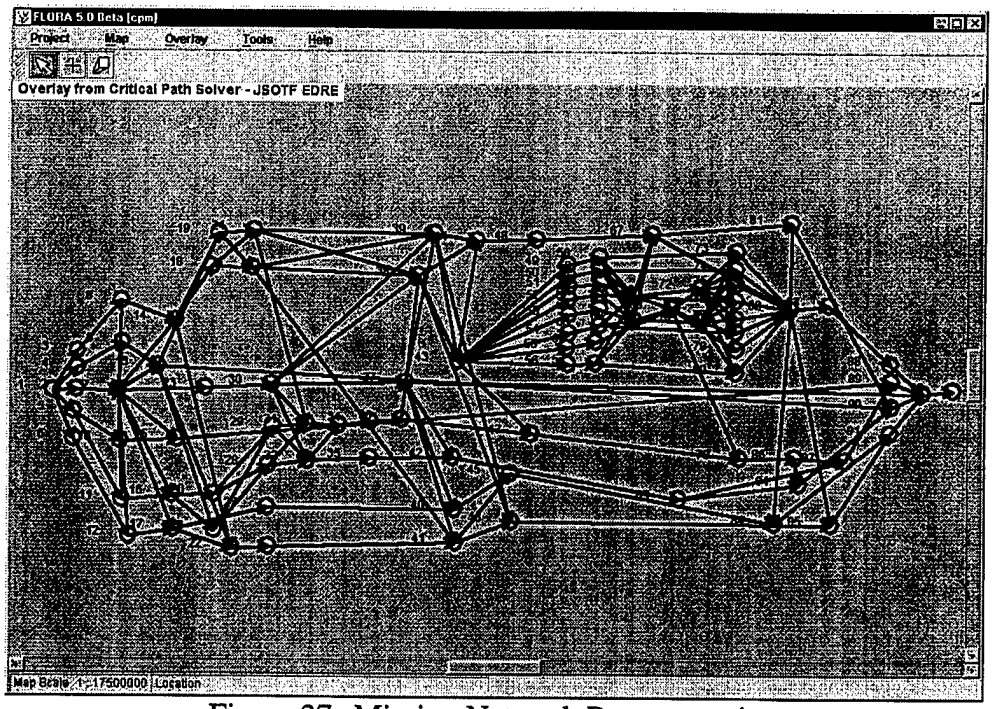


Figure 27. Mission Network Representation

For greater visibility and perspective, a zoom-in view of the major tasks associated with conduct of actions on the objective is shown in Figure 28.

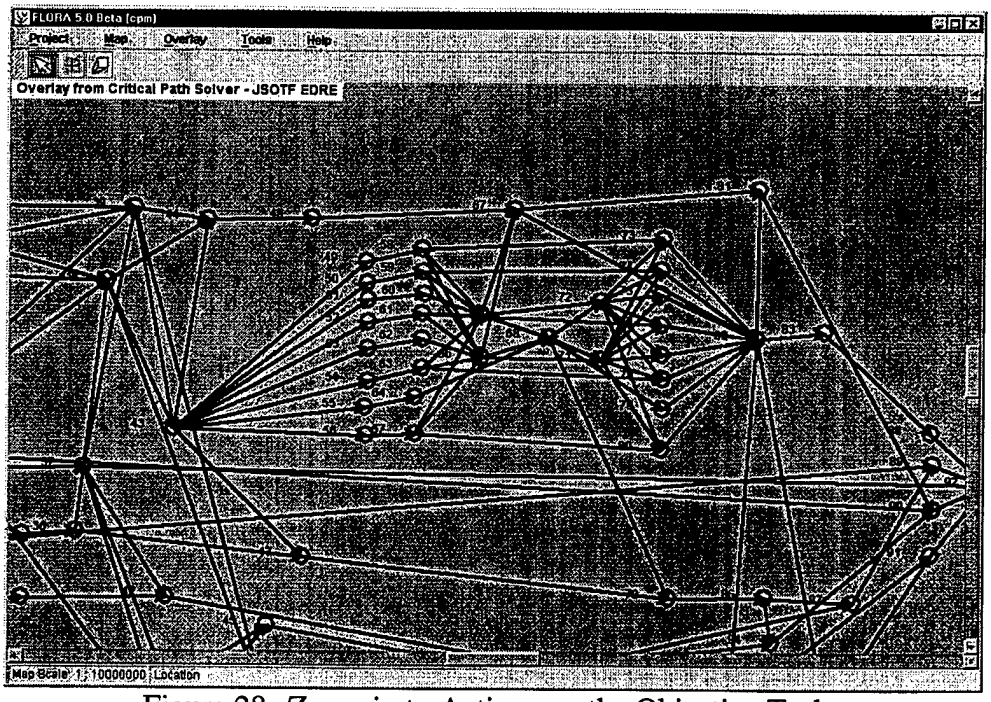


Figure 28. Zoom in to Actions on the Objective Tasks

V. ANALYSIS

In the end, the product of effective synchronization is maximum use of every resource to make the greatest contribution to success.

FM 100-5, *Operations* [Ref. 3: p. 2-9]

A. RESULTS

SOMPASS solved for and highlighted the critical path on the illustrative scenario network, and produced the synchronization matrix and execution checklists all within five seconds. A view of the solved network, which on the screen highlights the critical path in blue, is shown in Figure 29 as a black and white rendition.

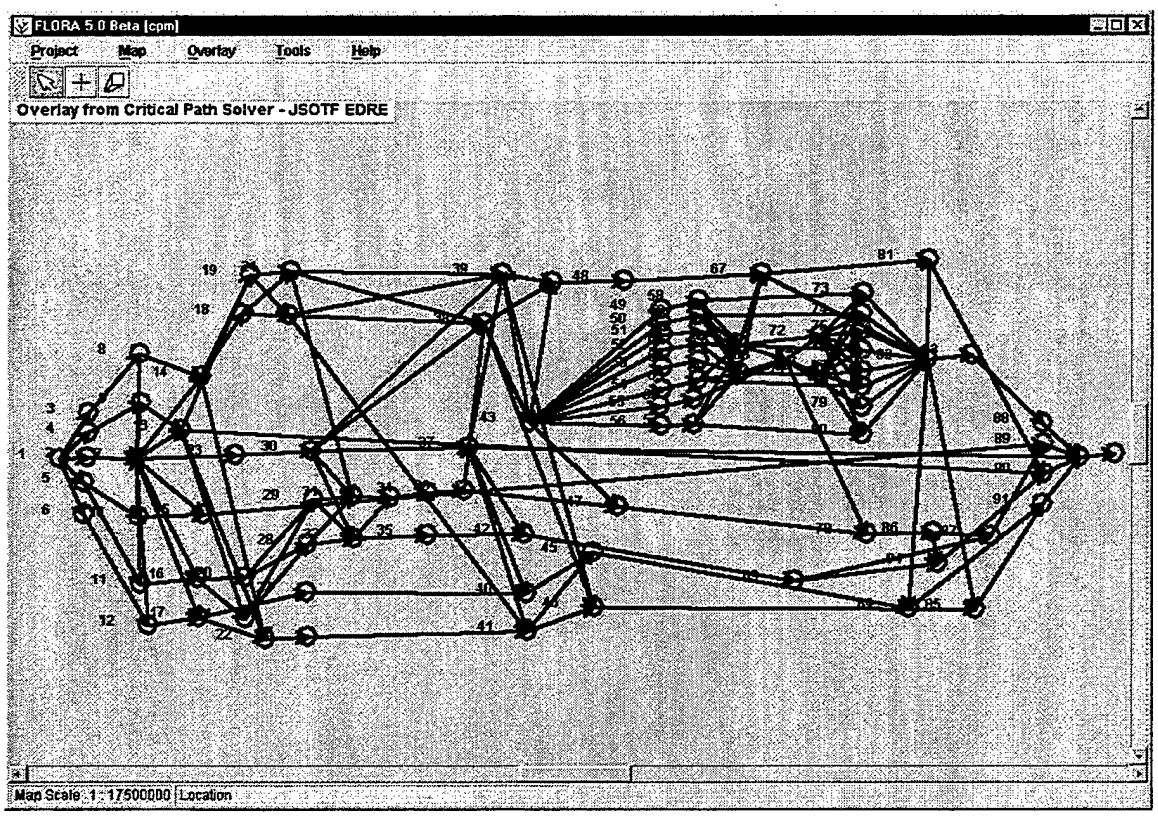


Figure 29. Solved Mission Network

There are several regions of the network where simultaneous parallel tasks are being conducted by multiple units, such as during receipt of the warning order and initial

mission planning. The concurrence of these tasks, in conjunction with their dependence, leads to a solution with multiple critical paths that is clearly apparent to a user viewing the solved network on a screen.

A representative view of the resulting synchronization matrix for the illustrative scenario is shown in Figure 30.

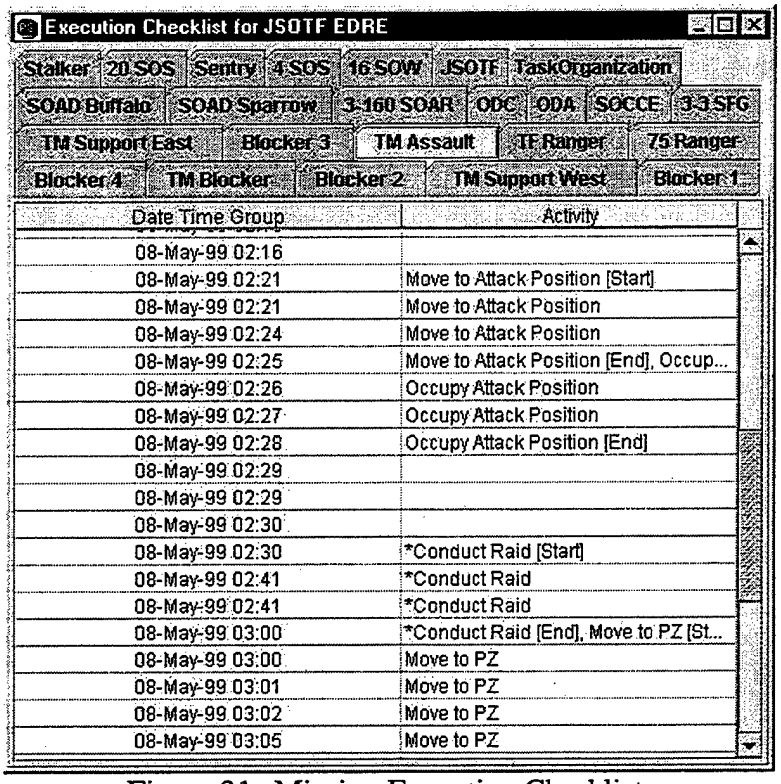
Synchronization Matrix for JSOTF EDRE

Unit	29-Apr-99 12:16	29-Apr-99 12:16	29-Apr-99 13:16	30-Apr-99 13:16	30-Apr-99 17:16	02-May-99 17:16	02-May-99 17:16	03-Ma
TaskOrganization								
JSOTF	*Stand Up JSO...	*Stand Up JSO...	Issue Warning ...	*Issue OPORD ...	*Issue OPORD ...	*Await Execute ...	*Await Execute ...	*Awai
75 Ranger		*Receive Warni...	*Receive Warni...					
TF Ranger			*Conduct Initial ...	*Conduct Initial ...	*Complete Mis...	*Complete Mis...	*Conduct Rehe...	*Cond
TM Blocker								
Blocker 4								
TM Support West								
Blocker 2								
TM Support East								
Blocker 1								
TM Assault								
Blocker 3								
3-160 SOAR		*Receive Warni...	*Receive Warni...	*Conduct Initial ...	*Complete Mis...	*Complete Mis...		
SOAD Buffalo						*Conduct Rehe...	*Conduct Rehe...	*Cond
SOAD Sparrow						Conduct Rehea...	Conduct Rehea...	Condu
3-3 SFG		*Receive Warni...	*Receive Warni...					
SOCCE								
ODC			*Conduct Initial ...	*Conduct Initial ...	Complete Missi...	Complete Missi...		
ODA			*Conduct Initial ...	*Conduct Initial ...	Complete Missi...	Complete Missi...	Conduct Rehea...	Condu
16 SOW		*Receive Warni...	*Receive Warni...	*Conduct Initial ...	Complete Missi...	Complete Missi...		
20 SOS								
Stalker						Conduct Rehea...	Conduct Rehea...	Condu
4 SOS								
Sentry						Final Mission P...	Final Mission P...	

Figure 30. Mission Synchronization Matrix

It is also possible to view the multiple critical paths in this operation by looking at several of the early columns in this synchronization matrix where several units have asterisks, indicating critical path tasks, occurring during the same time periods.

A representative view of the resulting unit execution checklists for the illustrative scenario is shown in Figure 31. The tab and time region shown in Figure 31 highlights Team Assault, the unit conducting the actual assault on the objective. As should be expected, the execution of the raid is an event on the critical path for this operation, since it is the main effort for the mission, and is indicated by asterisks before the event name.



Date Time Group	Activity
08-May-99 02:16	
08-May-99 02:21	Move to Attack Position [Start]
08-May-99 02:21	Move to Attack Position
08-May-99 02:24	Move to Attack Position
08-May-99 02:25	Move to Attack Position [End], Occup...
08-May-99 02:26	Occupy Attack Position
08-May-99 02:27	Occupy Attack Position
08-May-99 02:28	Occupy Attack Position [End]
08-May-99 02:29	
08-May-99 02:29	
08-May-99 02:30	
08-May-99 02:30	*Conduct Raid [Start]
08-May-99 02:41	*Conduct Raid
08-May-99 02:41	*Conduct Raid
08-May-99 03:00	*Conduct Raid [End], Move to PZ [St...
08-May-99 03:00	Move to PZ
08-May-99 03:01	Move to PZ
08-May-99 03:02	Move to PZ
08-May-99 03:05	Move to PZ

Figure 31. Mission Execution Checklists

B. SENSITIVITY ANALYSIS

An extremely powerful feature of SOMPASS is the ability to dynamically change properties associated with tasks, thereby allowing real-time sensitivity analysis or status updates. An example that demonstrates this ability involves two tasks in the operation: Node 70 – ODA Conducts BDA Post-strike Recon, and Node 82 – TF Ranger Conducts Pickup Zone (PZ) Operations. Figure 32 shows the initial solution of the mission network with the symbol displays for the two nodes in question. Node 70 is currently on the critical path, while Node 82 is not. Note that Node 82 is currently projected to last 8 minutes, with a calculated total float time of 7 minutes. This float time means that PZ operations could last as long as 15 minutes without affecting the critical path or total

duration of the mission, while any time longer than 15 minutes would delay completion of the entire operation.

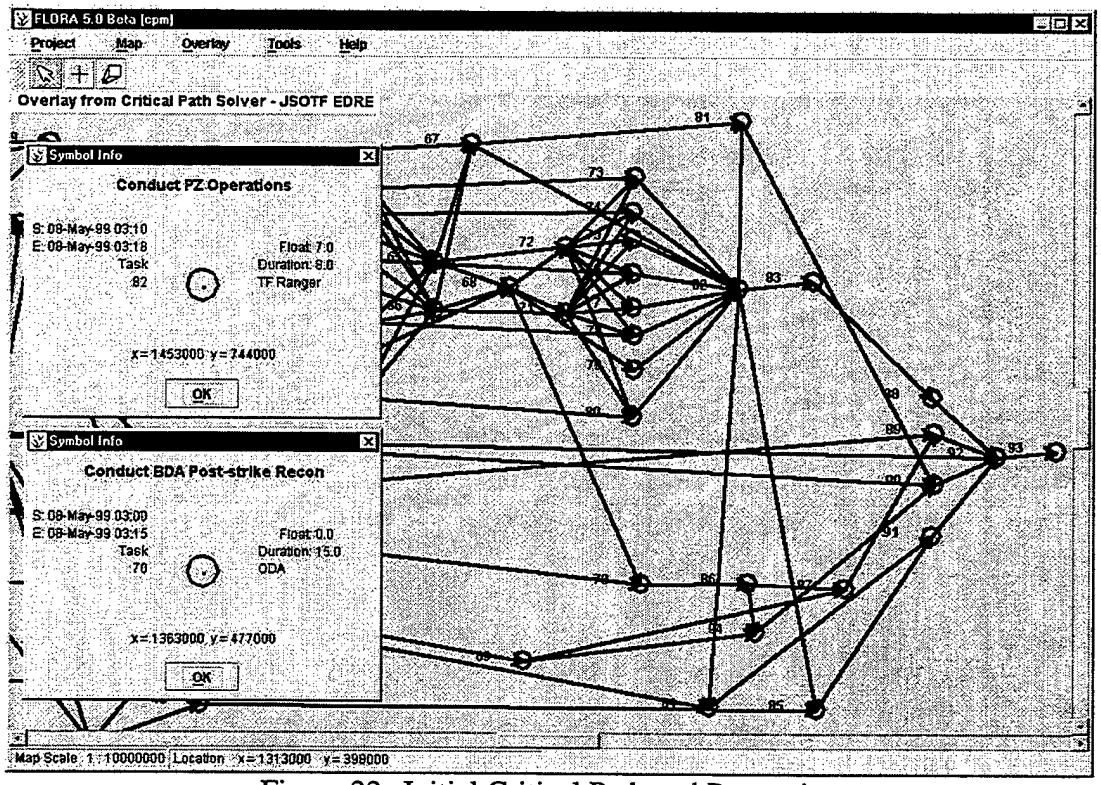


Figure 32. Initial Critical Path and Properties

Since conducting battalion-level PZ operations at night in a hostile environment is a very difficult and complex operation, it is conceivable that it may take longer than the currently projected time of 8 minutes.

If a planner wanted to analyze what changes in the operation would occur if the projected time to conduct PZ operations were to take as long as 16 minutes, rather than the current projection, he need only change the property value for the duration of that task, and resolve. Any changes in the critical path will be immediately displayed, and a new synchronization matrix and set of execution checklists will be generated. In this example, Figure 33 shows a change of the duration property for Node 82 from 8 minutes

to 16 minutes, and Figure 34 shows the resulting updated critical path and symbol displays for Nodes 70 and 82.

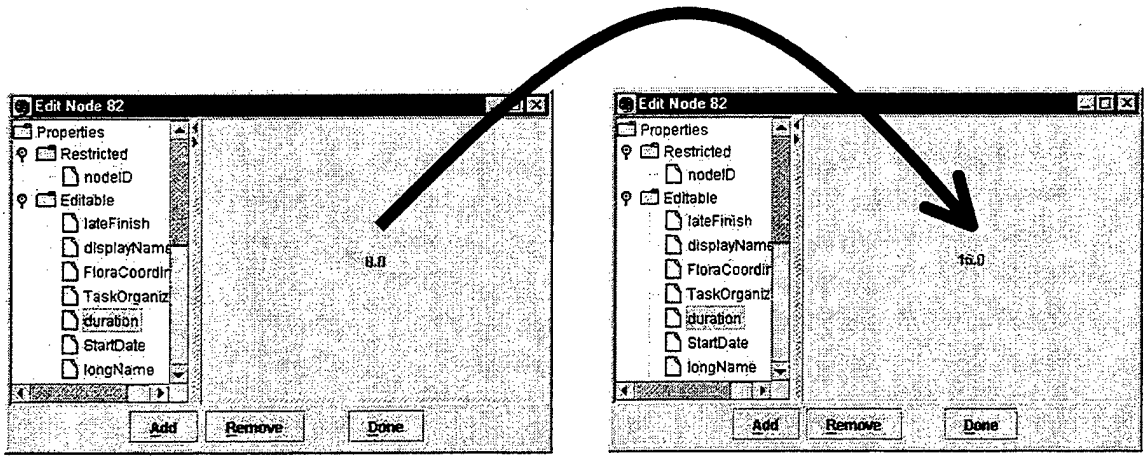


Figure 33. Modification of a Task Property

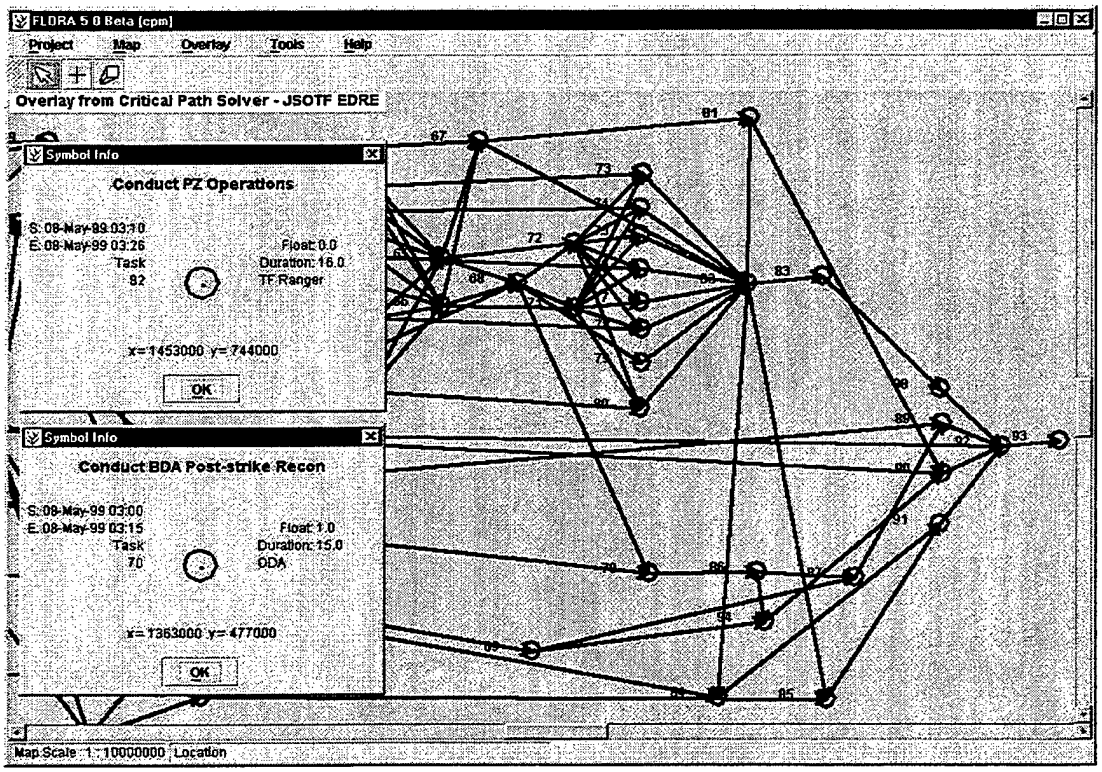


Figure 34. Updated Critical Path and Properties

In this instance, because there were only 7 minutes of float time available during PZ operations, a change of 8 minutes caused this task to displace BDA and post-strike recon

from the critical path and also extend the duration of the mission by one minute. Note also in Figure 34 that Node 70 now reflects one minute of slack time available since it is no longer on the critical path.

The speed at which these types of comparisons can be conducted greatly facilitates "what-if?" and course-of-action analysis that would otherwise take considerably longer. The capability also exists to receive real-time updates on task status over a network from involved units during the execution phase of an operation that would allow real-time analysis of the situation and highlight any potential choke points or delays.

C. APPLICABILITY

SOMPASS has the potential to offer increased mission planning efficiency and rapid analysis capabilities, both in training and operational environments. As mentioned above, it could also be integrated into such areas as rehearsal and execution battle tracking for real-time status monitoring. The highlighting of critical path events on the synchronization matrix and unit execution checklists also offers potential benefit to commanders to indicate where they may want to be located to best lead and direct an operation at its critical junctures.

In this thesis, only the participating units have been presented as the primary key associated with mission tasks, as the battlefield operating system (BOS) for maneuver, but many other areas could be represented just as easily to support operational needs or requirements. Some possibilities include all other battlefield operating system (BOS) components, decision points, and other key activities or time-phased events.

VI. THE NEXT LEVEL

This thesis presents a working system that can be used to assist in mission planning and analysis, but many additional capabilities could be developed and incorporated to further enhance its capabilities. Some possible avenues for development are described below.

A. RESOURCE MANAGEMENT

In operations that are logistically restrictive, or involve combat service support units or elements, the ability to examine scarce resource allocation and distribution would be useful. Features such as consumption and resupply monitoring, resource leveling algorithms, and trend analysis graphing would complement current features.

B. PRECENDENCE NETWORKS – MULTIPLE DEPENDENCY

SOMPASS currently supports normal finish-to-start single dependency networking, but some complex operations may involve more intricate relationships between activities and events. The ability to support more complex relationships between activities would allow more types of operations to be modeled and solved to provide additional user functionality.

A complex network involving non-standard or multiple dependencies is referred to as a 'precedence network.' The variations of precedence networks include a start-to-start dependency, also known as 'lag-start,' a finish-to-finish dependency, also known as 'lag-finish,' a start-to-finish dependency, or any combination of several dependencies. [Ref. 24]

C. AUTOMATED DATA CALCULATION AND SELECTION

Additional automated features would also provide additional functionality to users. For example, if geo-referenced locations were entered in conjunction with a unit and task, as well as a means of travel, an estimate for the travel time required could be determined automatically and provided as the initial duration of that task. Additionally, a set of selection or optimization routines might be developed and incorporated to select the best mode of travel, routes of transit, and resource requirements necessary for completion of a task. These determinations and requirements could then be forwarded across a network to the unit(s) that would support the mission, and added into their operational plan and network representation.

D. SIMULATION

Incorporation of independent or integrated simulations that might be used to model courses of action, engagement outcomes, threat activities, or other mission affecting factors could provide additional inputs or properties to represented operations and elements in SOMPASS to improve support to users. SOMPASS might also be used to display operational status of simulations or simulated operations to provide alternative views to conventional simulation outputs.

VII. CONCLUSION

This thesis provides a mission planning and analysis tool, the Special Operations Mission Planning and Analysis Support System (SOMPASS), for Special Operations Forces commanders and staffs to help reduce traditional mission planning preparation time and manual effort requirements, as well as enhance the capabilities for conducting analysis. While all services have strong interest in developing and acquiring new systems to leverage advances in information-age technology, USSOCOM, with its diverse composition of forces and nature of operations and environments, has many unique requirements that must be specifically addressed in order to provide systems that will be useful for their needs. These unique requirements are highlighted by the CINC, USSOCOM, in *Special Operations Forces, The Way Ahead* [Ref. 5], and embedded in the framework for their strategy for success: Mission Planning, Analysis, Rehearsal, and Execution (MPARE). SOMPASS is a system that was specifically designed to address the needs and requirements of USSOCOM, thereby providing a direction for further development and possible incorporation into the MPARE program.

SOMPASS uses an operations research approach and incorporates currently available advanced technologies to provide a powerful system to aid in mission planning and analysis. This system helps to reduce mission preparation time and effort by automating some of the requisite tasks involved, thereby giving commanders and staffs more time to focus on other aspects of mission preparation and execution. SOMPASS allows for dynamic property associations and rapid recalculation capabilities so that

many possible variations or contingencies may be examined for their effects on mission success before deciding on a final plan. This provides improved course of action (COA) analysis. Additionally, the dynamic and distributed capabilities of the system allow for multiple users to coordinate their efforts and share information to increase efficiency.

The automated production of synchronization matrices and execution checklists provides an extremely powerful capability to simplify mission synchronization which is a critical element to mission success. Their dynamic nature allows great flexibility for changes with minimal effort, either for analysis purposes or as the situation changes. These products can be distributed and updated over a network to all concerned elements in real-time. This eliminates the confusion generated by current hard-copy products that may be outdated by events and eliminates the need to reassemble staffs to distribute and discuss the latest changes. By highlighting the critical path tasks of an operation, commanders also have a means to identify potential decisive points so they may better determine where to best lead and direct the situation during key phases of an operation.

Due to its dynamic nature, SOMPASS can be used to support all operational phases of a mission, not just the planning phase. Real-time situation updates provided over a network, or entered during an operation will be rapidly incorporated to show any changes or effects on the current plan to allow steps to be taken to address the changing situation. This capability offers great flexibility in planning and response that could not be accomplished with traditional tools or methods.

SOMPASS is also simple to learn and use. Little training is required for proficiency beyond the description provided in this thesis and the walk-through of the

system provided in Appendix C. Users need not know anything about operations research or CPM to operate the system; they only need to know what tasks must be accomplished, and what order to complete them. Due to the distributed nature of the system, missions can be entered and built by many users simultaneously and combined together, thus allowing each unit to focus only on their area of responsibility while being provided the entire operation. Similarly, the workload can be shared within the same unit so as not to excessively burden one individual.

This thesis provides a system that can offer great benefits to USSOCOM and provide the groundwork for further development in support of MPARE. The design and system architecture of SOMPASS also allows for continued growth and enhancement without the need for total replacement and retraining, thereby meeting the needs of the Special Operations Forces today, and offering potential benefits to all of our Armed Forces in the future.

APPENDIX A. CRITCAL PATH SOLVER

Below is the Java code used to implement a loosely coupled component algorithm that can perform a topological sort on and solve for the critical path of a König graph.

```
package mil.navy.nps.cpm;

import java.io.FileInputStream;
import java.util.Iterator;
import javax.swing.JTabbedPane;
import mil.army.trac.konig.Arc;
import mil.army.trac.konig.ArcSet;
import mil.army.trac.konig.Graph;
import mil.army.trac.konig.GraphFrame;
import mil.army.trac.konig.GraphPanel;
import mil.army.trac.konig.Node;
import mil.army.trac.konig.NodeSet;

/**
 * Critical Path Method solver algorithm.
 * <P>
 * This class includes static methods required to compute the Critical Path as well
 * as several other network convenience functions.
 * <P>
 *
 * @version 1.0.0, 31 May 1999
 *
 * @author Keith A. Hattes
**/

public class CriticalPath {

// class constants

   /**
    * The default property name for the inDegree of a Node.
   **/
   public static final String IN_DEGREE = "inDegree";

   /**
    * The default property name for the outDegree of a Node.
   **/
   public static final String OUT_DEGREE = "outDegree";

   /**
    * The default property name for the results of a topological sort, or acyclic
    * ordering, of Nodes.
   **/
   public static final String TOPOLOGICAL_ORDER = "topologicalOrder";

   /**
    * The default property name for the Earliest Start Time (EST) of a Node.
   **/
   public static final String EARLY_START = "earlyStart";

   /**
    * The default property name for the Latest Start Time (LST) of a Node.
   **/
   public static final String LATE_START = "lateStart";

   /**
    * The default property name for the Earliest Finish Time (EFT) of a Node.
   **/
   public static final String EARLY_FINISH = "earlyFinish";

   /**
    * The default property name for the Latest Finish Time (LFT) of a Node.
   **/
   public static final String LATE_FINISH = "lateFinish";
```

```
    /**
     * The default property name for the Float Time of a Node.
    **/
    public static final String TOTAL_FLOAT = "totalFloat";

    /**
     * The default property name for whether an Arc or Node is on the Critical Path.
    **/
    public static final String ON_CRITICAL_PATH = "onCriticalPath";

    /**
     * The default property name for the total duration of a project.
    **/
    public static final String PROJECT_DURATION = "projectDuration";

    /**
     * The default property name for the duration of an event.
    **/
    public static final String DURATION = "duration";

    /**
     * The default property name for the must start by time of an event.
    **/
    public static final String REQUIRED_START = "requiredStart";

    /**
     * The default property name for the must finish by time of an event.
    **/
    public static final String REQUIRED_FINISH = "requiredFinish";

// class methods

    /**
     * Calculates the inDegree and OutDegree of all Nodes in a Graph and stores the
     * results as Node properties.
     *
     * @param aGraph the <CODE>Graph</CODE> on which to compute the degree.
     * @param inDegreeKey the property reference key to store the inDegree result.
     * @param outDegreeKey the property reference key to store the outDegree result.
    **/
    public static void setDegree(Graph aGraph, String inDegreeKey, String outDegreeKey) {
       NodeSet nodes = aGraph.getNodeSet();
       ArcSet arcs = aGraph.getArcSet();
       Node aNode;
       Arc anArc;
       int inDegree;
       int outDegree;
       for(Iterator i = nodes.iterator(); i.hasNext();) {
          aNode = (Node) i.next();
          aNode.putProperty(inDegreeKey, new Integer(0));
          aNode.putProperty(outDegreeKey, new Integer(0));
       }
       for(Iterator i = arcs.iterator(); i.hasNext();) {
          anArc = (Arc) i.next();
          aNode = (Node) anArc.getToNode();
          inDegree = ((Number) aNode.getProperty(inDegreeKey)).intValue() + 1;
          aNode.putProperty(IN_DEGREE, new Integer(inDegree));
          aNode = (Node) anArc.getFromNode();
          outDegree = ((Number) aNode.getProperty(outDegreeKey)).intValue() + 1;
          aNode.putProperty(outDegreeKey, new Integer(outDegree));
       }
    }

    /**
     * Calculates the inDegree and OutDegree of all Nodes in a Graph and stores the
     * results as Node properties in the default property reference keys.
     *
     * @param aGraph the <CODE>Graph</CODE> on which to compute the degree.
    **/
    public static void setDegree(Graph aGraph) {
       setDegree(aGraph, IN_DEGREE, OUT_DEGREE);
    }

    /**
     * Calculates the inDegree of each Node in a Graph and stores the result as a Node
     * property.
     *
```

```
 * @param aGraph the <CODE>Graph</CODE> on which to compute the degree.
 * @param inDegreeKey the property reference key to store the inDegree result.
**/
public static void setInDegree(Graph aGraph, String inDegreeKey) {
   NodeSet nodes = aGraph.getNodeSet();
   ArcSet arcs = aGraph.getArcSet();
   Node aNode;
   Arc anArc;
   int inDegree;
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      aNode.putProperty(inDegreeKey, new Integer(0));
   }
   for(Iterator i = arcs.iterator(); i.hasNext();) {
      anArc = (Arc) i.next();
      aNode = (Node) anArc.getToNode();
      inDegree = ((Number) aNode.getProperty(inDegreeKey)).intValue() + 1;
      aNode.putProperty(IN_DEGREE, new Integer(inDegree));
   }
}

/**
 * Calculates the inDegree of each Node in a Graph and stores the result as a Node
 * property in the default property reference key.
 *
 * @param aGraph the <CODE>Graph</CODE> on which to compute the degree.
**/
public static void setInDegree(Graph aGraph) {
   setInDegree(aGraph, IN_DEGREE);
}

/**
 * Calculates the outDegree of each Node in a Graph and stores the result as a Node
 * property.
 *
 * @param aGraph the <CODE>Graph</CODE> on which to compute the degree.
 * @param outDegreeKey the property reference key to store the outDegree result.
**/
public static void setOutDegree(Graph aGraph, String outDegreeKey) {
   NodeSet nodes = aGraph.getNodeSet();
   ArcSet arcs = aGraph.getArcSet();
   Node aNode;
   Arc anArc;
   int outDegree;
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      aNode.putProperty(outDegreeKey, new Integer(0));
   }
   for(Iterator i = arcs.iterator(); i.hasNext();) {
      anArc = (Arc) i.next();
      aNode = (Node) anArc.getFromNode();
      outDegree = ((Number) aNode.getProperty(outDegreeKey)).intValue() + 1;
      aNode.putProperty(outDegreeKey, new Integer(outDegree));
   }
}

/**
 * Calculates the outDegree of each Node in a Graph and stores the result as a Node
 * property in the default property reference key.
 *
 * @param aGraph the <CODE>Graph</CODE> on which to compute the degree.
**/
public static void setOutDegree(Graph aGraph) {
   setOutDegree(aGraph, OUT_DEGREE);
}

/**
 * Returns the inDegree of a Node in a Graph.  If it does not already exist, it
 * calculates the inDegree for the entire Graph first.
 *
 * @param aNode the <CODE>Node</CODE> of interest.
 * @param aGraph the parent <CODE>Graph</CODE>.
 * @param inDegreeKey the property reference key to query.
 *
 * @return the inDegree of the Node.
**/
public static int getInDegree(Node aNode, Graph aGraph, String inDegreeKey) {
```

```
    int inDegree;
    Object obj = aNode.getProperty(inDegreeKey, null);
    if(obj == null) {
       setInDegree(aGraph, inDegreeKey);
       inDegree = ((Number) aNode.getProperty(inDegreeKey)).intValue();
    }
    else {
       inDegree = ((Number) obj).intValue();
    }
    return inDegree;
 }

 /**
  * Returns the inDegree of a Node in a Graph.  If it does not already exist, it
  * calculates the inDegree for the entire Graph first and stores the result in the
  * default property reference key.
  *
  * @param aNode the <CODE>Node</CODE> of interest.
  * @param aGraph the parent <CODE>Graph</CODE>.
  *
  * @return the inDegree of the Node.
 **/
 public static int getInDegree(Node aNode, Graph aGraph) {
    return getInDegree(aNode, aGraph, IN_DEGREE);
 }

 /**
  * Returns the outDegree of a Node in a Graph.  If it does not already exist, it
  * calculates the outDegree for the entire Graph first.
  *
  * @param aNode the <CODE>Node</CODE> of interest.
  * @param aGraph the parent <CODE>Graph</CODE>.
  * @param outDegreeKey the property reference key to query.
  *
  * @return the outDegree of the Node.
 **/
 public static int getOutDegree(Node aNode, Graph aGraph, String outDegreeKey) {
    int outDegree;
    Object obj = aNode.getProperty(outDegreeKey, null);
    if(obj == null) {
       setOutDegree(aGraph, outDegreeKey);
       outDegree = ((Number) aNode.getProperty(outDegreeKey)).intValue();
    }
    else {
       outDegree = ((Number) obj).intValue();
    }
    return outDegree;
 }

 /**
  * Returns the outDegree of a Node in a Graph.  If it does not already exist, it
  * calculates the outDegree for the entire Graph first and stores the result in the
  * default property reference key.
  *
  * @param aNode the <CODE>Node</CODE> of interest.
  * @param aGraph the parent <CODE>Graph</CODE>.
  *
  * @return the outDegree of the Node.
 **/
 public static int getOutDegree(Node aNode, Graph aGraph) {
    return getOutDegree(aNode, aGraph, OUT_DEGREE);
 }

 /**
  * Returns a <CODE>NodeSet</CODE> of the Forward Star of a Node in a Graph.
  *
  * @param aNode the <CODE>Node</CODE> of interest.
  * @param aGraph the parent <CODE>Graph</CODE>.
  *
  * @return the Forward Star of the Node.
 **/
 public static NodeSet getForwardStar(Node aNode, Graph aGraph) {
    NodeSet forwardStar = new NodeSet();
    ArcSet arcs = aGraph.getArcSet();
    Arc anArc;
    for(Iterator i = arcs.iterator(); i.hasNext();) {
       anArc = (Arc) i.next();
```

```
      if(((Node) anArc.getFromNode()) == aNode) {
         forwardStar.add(anArc.getToNode());
      }
   }
   return forwardStar;
}

/**
 * Returns a <CODE>NodeSet</CODE> of the Reverse Star of a Node in a Graph.
 *
 * @param aNode the <CODE>Node</CODE> of interest.
 * @param aGraph the parent <CODE>Graph</CODE>.
 *
 * @return the Reverse Star of the Node.
**/
public static NodeSet getReverseStar(Node aNode, Graph aGraph) {
   NodeSet reverseStar = new NodeSet();
   ArcSet arcs = aGraph.getArcSet();
   Arc anArc;
   for(Iterator i = arcs.iterator(); i.hasNext();) {
      anArc = (Arc) i.next();
      if(((Node) anArc.getToNode()) == aNode) {
         reverseStar.add(anArc.getFromNode());
      }
   }
   return reverseStar;
}

/**
 * Conducts a topological sort of a Graph, and sets the acyclic ordering property
 * value of each Node in the Graph with its acyclic number.
 *
 * @param aGraph the <CODE>Graph</CODE> to conduct the topological ordering on.
 * @param inDegreeKey the property reference key containing the inDegree of Nodes.
 * @param topologicalOrderKey the property reference key to store the topological
 *        order sequence number of each Node.
 *
 * @return a shallow copy of the Graph's <CODE>NodeSet</CODE> sorted by the
 *         <CODE>TopologicalComparator</CODE>.
 *
 * @exception <CODE>CycleException</CODE> if the Graph contains a cycle.
**/
public static NodeSet topologicalSort(Graph aGraph, String inDegreeKey,
  String topologicalOrderKey) throws CycleException {
   Queue hold = new Queue();
   String tempDegree = "tempDegree";
   NodeSet nodes = aGraph.getNodeSet();
   int counter = 0;
   int temp;
   NodeSet star;
   Node aNode = null;
   Node nextNode = null;
   Node cycleNode;
   setInDegree(aGraph, inDegreeKey);
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      aNode.putProperty(topologicalOrderKey, new Integer(0));
      temp = ((Number) aNode.getProperty(inDegreeKey)).intValue();
      aNode.putProperty(tempDegree, new Integer(temp));
      if(temp == 0) {
         hold.push(aNode);
      }
   }
   while(!hold.isEmpty()) {
      aNode = (Node) hold.pop();
      aNode.putProperty(topologicalOrderKey, new Integer(++counter));
      star = getForwardStar(aNode, aGraph);
      for(Iterator i = star.iterator(); i.hasNext();) {
         nextNode = (Node) i.next();
         temp = ((Number) nextNode.getProperty(tempDegree)).intValue();
         nextNode.putProperty(tempDegree, new Integer(--temp));
         if(temp == 0) {
            hold.push(nextNode);
         }
      }
   }
```

```
    if(counter < nodes.size()) {
       cycleNode = nextNode;
       for(Iterator i = nodes.iterator(); i.hasNext();) {
          aNode = (Node) i.next();
          aNode.removeProperty(tempDegree);
          aNode.removeProperty(topologicalOrderKey);
       }
       throw new CycleException(cycleNode, aGraph.toString() + " contains a cycle.");
    }
    else {
       NodeSet orderedNodes = new NodeSet(new TopologicalComparator());
       for(Iterator i = nodes.iterator(); i.hasNext();) {
          aNode = (Node) i.next();
          aNode.removeProperty(tempDegree);
       }
       orderedNodes.addAll(nodes);
       return orderedNodes;
    }
 }

 /**
  * Conducts a topological sort of a Graph using the default inDegree property
  * reference key and sets the default acyclic ordering property value of each Node
  * in the Graph with its acyclic number.
  *
  * @param aGraph the <CODE>Graph</CODE> to conduct the topological ordering on.
  *
  * @return a shallow copy of the Graph's <CODE>NodeSet</CODE> sorted by the
  *         <CODE>TopologicalComparator</CODE>.
  *
  * @exception <CODE>CycleException</CODE> if the Graph contains a cycle.
 **/
 public static NodeSet topologicalSort(Graph aGraph) throws CycleException {
    return topologicalSort(aGraph, IN_DEGREE, TOPOLOGICAL_ORDER);
 }

 /**
  * Conducts a forward pass and topological sort of a Graph, and sets all associated
  * properties specified for use in critical path determination.
  *
  * @param aGraph the <CODE>Graph</CODE> to conduct the forward pass on.
  * @param durationKey the property reference key containing the length of the event.
  * @param inDegreeKey the property reference key containing the inDegree of Nodes.
  * @param topologicalOrderKey the property reference key to store the topological
  *        order sequence number of each Node.
  * @param earlyStartKey the property reference key to store the event Earliest Start
  *        Time.
  * @param earlyFinishKey the property reference key to store the event Earliest Finish
  *        Time.
  * @param lateFinishKey the property reference key to store the Latest Finish Time of
  *        the last event for synchronization with <CODE>backwardPass()</CODE>.
  * @param requiredStartKey the property reference key that has the must start by time
  *        for an event.
  * @param requiredFinishKey the property reference key that has the must finish by time
  *        for an event.
  *
  * @exception <CODE>CycleException</CODE> if the Graph contains a cycle.
 **/
 protected static void forwardPass(Graph aGraph, String durationKey, String inDegreeKey,
   String topologicalOrderKey, String earlyStartKey, String earlyFinishKey,
   String lateFinishKey, String requiredStartKey, String requiredFinishKey)
   throws CycleException {
    Queue hold = new Queue();
    String tempDegree = "tempDegree";
    NodeSet nodes = aGraph.getNodeSet();
    int counter = 0;
    int temp;
    double start;
    double theDuration;
    double finish;
    double max;
    NodeSet star;
    Node aNode = null;
    Node nextNode = null;
    Node cycleNode;
    for(Iterator i = nodes.iterator(); i.hasNext();) {
       aNode = (Node) i.next();
```

```
        aNode.putProperty(topologicalOrderKey, new Integer(0));
        temp = ((Number) aNode.getProperty(inDegreeKey)).intValue();
        aNode.putProperty(tempDegree, new Integer(temp));
        if(temp == 0) {
           hold.push(aNode);
           if(aNode.getProperty(requiredStartKey) == null) {
              start = 0.0;
           }
           else {
              start = ((Number) aNode.getProperty(requiredStartKey)).doubleValue();
           }

           theDuration = ((Number) aNode.getProperty(durationKey,
             new Double(0.0))).doubleValue();

           aNode.putProperty(earlyStartKey, new Double(start));
           aNode.putProperty(earlyFinishKey, new Double(start + theDuration));
        }
     }
     while(!hold.isEmpty()) {
        aNode = (Node) hold.pop();
        aNode.putProperty(topologicalOrderKey, new Integer(++counter));
        finish = ((Number) aNode.getProperty(earlyFinishKey,
          new Double(0.0))).doubleValue();
        star = getForwardStar(aNode, aGraph);
        for(Iterator i = star.iterator(); i.hasNext();) {
           nextNode = (Node) i.next();
           temp = ((Integer) nextNode.getProperty(tempDegree)).intValue();
           nextNode.putProperty(tempDegree, new Integer(--temp));
           start = ((Number) nextNode.getProperty(earlyStartKey,
             new Double(0.0))).doubleValue();
           theDuration = ((Number) nextNode.getProperty(durationKey,
             new Double(0.0))).doubleValue();
           max = Math.max(finish, start);
           nextNode.putProperty(earlyStartKey, new Double(max));
           nextNode.putProperty(earlyFinishKey, new Double(max + theDuration));
           if(temp == 0) {
              hold.push(nextNode);
           }
        }
     }
     if(counter < nodes.size()) {
        cycleNode = nextNode;
        for(Iterator i = nodes.iterator(); i.hasNext();) {
           aNode = (Node) i.next();
           aNode.removeProperty(tempDegree);
           aNode.removeProperty(topologicalOrderKey);
        }
        throw new CycleException(cycleNode, aGraph.toString() + " contains a cycle.");
     }
     else {
        aNode = aGraph.getNode(topologicalOrderKey, new Integer(nodes.size()));
        aNode.putProperty(lateFinishKey, aNode.getProperty(earlyFinishKey));
        for(Iterator i = nodes.iterator(); i.hasNext();) {
           aNode = (Node) i.next();
           aNode.removeProperty(tempDegree);
        }
     }
  }

  /**
   * Conducts a backward pass on a Graph, and sets all associated properties specified
   * for use in critical path determination.
   *
   * @param aGraph the <CODE>Graph</CODE> to conduct the forward pass on.
   * @param durationKey the property reference key containing the length of the event.
   * @param topologicalOrderKey the property reference key containing the topological
   *        order sequence number of each Node.
   * @param earlyStartKey the property reference key containing the event Earliest Start
   *        Time.
   * @param lateStartKey the property reference key to store the event Latest Start Time.
   * @param lateFinishKey the property reference key to store the event Latest Finish
   *        Time.
   * @param totalFloatKey the property reference key to store the event Float Time.
   * @param onCriticalPathKey the property reference key to set if an event is on the
   *        critical path.
  **/
```

```
protected static void backwardPass(Graph aGraph, String durationKey,
  String topologicalOrderKey, String earlyStartKey, String lateStartKey,
  String lateFinishKey, String totalFloatKey, String onCriticalPathKey) {
   NodeSet nodes = new NodeSet(
     new TopologicalComparator(ComparatorDirection.DESCENDING, topologicalOrderKey));
   nodes.addAll(aGraph.getNodeSet());
   double start;
   double theDuration;
   double finish;
   double min;
   double totalFloat;
   double tolerance = 1.0e-6;
   NodeSet star;
   Node aNode;
   Node priorNode;
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      finish = ((Number) aNode.getProperty(lateFinishKey,
        new Double(Double.MAX_VALUE))).doubleValue();
      theDuration = ((Number) aNode.getProperty(durationKey,
        new Double(0.0))).doubleValue();
      start = finish - theDuration;
      aNode.putProperty(lateStartKey, new Double(start));
      star = getReverseStar(aNode, aGraph);
      for(Iterator j = star.iterator(); j.hasNext();) {
         priorNode = (Node) j.next();
         finish = ((Number) priorNode.getProperty(lateFinishKey,
           new Double(Double.MAX_VALUE))).doubleValue();
         theDuration = ((Number) priorNode.getProperty(durationKey,
           new Double(0.0))).doubleValue();
         min = Math.min(start, finish);
         priorNode.putProperty(lateFinishKey, new Double(min));
         priorNode.putProperty(lateStartKey, new Double(min - theDuration));
      }
   }
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      start = ((Number) aNode.getProperty(earlyStartKey)).doubleValue();
      finish = ((Number) aNode.getProperty(lateStartKey)).doubleValue();
      totalFloat = finish - start;
      aNode.putProperty(totalFloatKey, new Double(totalFloat));
      if((totalFloat < tolerance) && (totalFloat > -tolerance)) {
         aNode.putProperty(onCriticalPathKey, new Boolean(true));
      }
   }
}

/**
 * Conducts all operations necessary to determine the critical path of a Graph, and
 * sets all associated properties specified for use in critical path determination.
 * Returns an <CODE>ArcSet</CODE> containing all of the Arcs on the critical path.
 *
 * @param aGraph the <CODE>Graph</CODE> to conduct the forward pass on.
 * @param durationKey the property reference key containing the length of the event.
 * @param projectDurationKey the property reference key to store the total duration
 *        of the project in the Graph.
 * @param inDegreeKey the property reference key containing the inDegree of Nodes.
 * @param topologicalOrderKey the property reference key to store the topological order
 *        sequence number of each Node.
 * @param earlyStartKey the property reference key to store the event Earliest Start
 *        Time.
 * @param lateStartKey the property reference key to store the event Latest Start Time.
 * @param earlyFinishKey the property reference key to store the event Earliest Finish
 *        Time.
 * @param lateFinishKey the property reference key to store the event Latest Finish
 *        Time.
 * @param totalFloatKey the property reference key to store the event Float Time.
 * @param requiredStartKey the property reference key that has the must start by time
 *        for an event.
 * @param requiredFinishKey the property reference key that has the must finish by time
 *        for an event.
 * @param onCriticalPathKey the property reference key to set if an event
 *        (Node or Arc) is on the critical path.
 * @param setDegree the flag to determine if calculation of Degrees needs to be
 *        conducted.
 *
 * @return all Arcs on the critical path.
```

```
 *
 * @exception <CODE>CycleException</CODE> if the Graph contains a cycle.
**/
public static ArcSet getCriticalPath(Graph aGraph, String durationKey,
  String projectDurationKey, String inDegreeKey, String outDegreeKey,
  String topologicalOrderKey, String earlyStartKey, String lateStartKey,
  String earlyFinishKey, String lateFinishKey, String totalFloatKey,
  String requiredStartKey, String requiredFinishKey, String onCriticalPathKey,
  boolean setDegree) throws CycleException {
   double start = Double.MAX_VALUE;
   double finish = Double.MIN_VALUE;
   double compare;
   NodeSet nodes = aGraph.getNodeSet();
   Node aNode;
   ArcSet arcs = aGraph.getArcSet();
   ArcSet theCriticalPath = new ArcSet();
   Arc anArc;
   boolean fromNode;
   boolean toNode;
   aGraph.removeProperty(earlyStartKey);
   aGraph.removeProperty(lateFinishKey);
   aGraph.removeProperty(projectDurationKey);
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      aNode.removeProperty(inDegreeKey);
      aNode.removeProperty(outDegreeKey);
      aNode.removeProperty(topologicalOrderKey);
      aNode.removeProperty(earlyStartKey);
      aNode.removeProperty(lateStartKey);
      aNode.removeProperty(earlyFinishKey);
      aNode.removeProperty(lateFinishKey);
      aNode.removeProperty(totalFloatKey);
      aNode.removeProperty(onCriticalPathKey);
   }
   for(Iterator i = arcs.iterator(); i.hasNext();) {
      anArc = (Arc) i.next();
      anArc.removeProperty(onCriticalPathKey);
   }
   if(setDegree) {
      setDegree(aGraph, inDegreeKey, outDegreeKey);
   }
   forwardPass(aGraph, durationKey, inDegreeKey, topologicalOrderKey, earlyStartKey,
     earlyFinishKey, lateFinishKey, requiredStartKey, requiredFinishKey);
   backwardPass(aGraph, durationKey, topologicalOrderKey, earlyStartKey, lateStartKey,
     lateFinishKey, totalFloatKey, onCriticalPathKey);
   for(Iterator i = nodes.iterator(); i.hasNext();) {
      aNode = (Node) i.next();
      compare = ((Number) aNode.getProperty(earlyStartKey,
        new Double(Double.MAX_VALUE))).doubleValue();
      if(compare < start) { start = compare; }
      compare = ((Number) aNode.getProperty(lateFinishKey,
        new Double(Double.MIN_VALUE))).doubleValue();
      if(compare > finish) { finish = compare; }
   }
   aGraph.putProperty(projectDurationKey, new Double(finish - start));
   aGraph.putProperty(earlyStartKey, new Double(start));
   aGraph.putProperty(lateFinishKey, new Double(finish));
   for(Iterator i = arcs.iterator(); i.hasNext();) {
      anArc = (Arc) i.next();
      aNode = (Node) anArc.getFromNode();
      fromNode = ((Boolean) aNode.getProperty(onCriticalPathKey,
        new Boolean(false))).booleanValue();
      aNode = (Node) anArc.getToNode();
      toNode = ((Boolean) aNode.getProperty(onCriticalPathKey,
        new Boolean(false))).booleanValue();
      if(fromNode && toNode) {
         anArc.putProperty(onCriticalPathKey, new Boolean(true));
         theCriticalPath.add(anArc);
      }
   }
   return theCriticalPath;
}

/**
 * Conducts all operations necessary to determine the critical path of a Graph, and
 * sets all associated default properties for use in critical path determination.
 * Returns an <CODE>ArcSet</CODE> containing all of the Arcs on the critical path.
```

```
     *
     * @param aGraph the <CODE>Graph</CODE> to conduct the forward pass on.
     *
     * @return all Arcs on the critical path.
     *
     * @exception <CODE>CycleException</CODE> if the Graph contains a cycle.
    **/
    public static ArcSet getCriticalPath(Graph aGraph)
      throws CycleException {
       return getCriticalPath(aGraph, DURATION, PROJECT_DURATION, IN_DEGREE, OUT_DEGREE,
         TOPOLOGICAL_ORDER, EARLY_START, LATE_START, EARLY_FINISH, LATE_FINISH,
         TOTAL_FLOAT, null, null, ON_CRITICAL_PATH, true);
    }

// main method

    /**
     * A simple test case that displays a GraphPanel with properties and critical path
     * results.
    **/
    public static void main(String[] args) {
       Graph aGraph = new Graph();
       String graphName = "test";
       if(args.length > 0) {
          graphName = args[0];
       }
       try {
          aGraph = new Graph(graphName + ".graph");
          aGraph.inputNodes(new FileInputStream(graphName + ".nodes"));
       }
       catch(Exception e) {
          System.err.println(e + " has File errors.");
          System.exit(0);
       }
       try {
          getCriticalPath(aGraph);
       }
       catch(CycleException e) {
          System.out.println(e);
          System.out.println("Cycle Node: " + e.getCycleNode());
          System.exit(0);
       }
       JTabbedPane aPane = new JTabbedPane();
       aPane.addTab(graphName, new GraphPanel(aGraph));
       GraphFrame aFrame = new GraphFrame(graphName);
       aFrame.getContentPane().add(aPane);
       aFrame.setVisible(true);
    }
}
```

APPENDIX B. ILLUSTRATIVE SCENARIO ACTIVITY AND EVENT LIST

Table 2 contains a list of the mission tasks and their properties associated with the illustrative scenario.

Task #	Task Description	Unit Responsible	Duration HH:MM	Succeeding Tasks
1	Stand Up JSOTF	JSOTF	00:00	2,3,4,5,6
2	Issue Warning Order	JSOTF	01:00	7
3	Receive Warning Order	JSOTF/75 Ranger	01:00	8
4	Receive Warning Order	JSOTF/3-160 SOAR	01:00	9
5	Receive Warning Order	JSOTF/3-3 SFG	01:00	10,11
6	Receive Warning Order	JSOTF/16 SOW	01:00	12
7	Issue OPORD	JSOTF	04:00	13,14,15,16,17,23
8	Conduct Initial Mission Planning	JSOTF/75 Ranger/TF Ranger	24:00	7,14
9	Conduct Initial Mission Planning	JSOTF/3-160 SOAR	24:00	7,13
10	Conduct Initial Mission Planning	JSOTF/3-3 SFG/SOCCE/ODC	24:00	7,15
11	Conduct Initial Mission Planning	JSOTF/3-3 SFG/SOCCE/ODA	24:00	7,16
12	Conduct Initial Mission Planning	JSOTF/16 SOW	24:00	7,17
13	Complete Mission Planning	JSOTF/3-160 SOAR	48:00	19,21,22,90
14	Complete Mission Planning	JSOTF/75 Ranger/TF Ranger	48:00	18,19,22
15	Complete Mission Planning	JSOTF/3-3 SFG/SOCCE/ODC	48:00	89
16	Complete Mission Planning	JSOTF/3-3 SFG/SOCCE/ODA	48:00	20,21
17	Complete Mission Planning	JSOTF/16 SOW	48:00	22,27
18	Conduct Rehearsals	JSOTF/75 Ranger/TF Ranger	48:00	24,25
19	Conduct Rehearsals	JSOTF/3-160 SOAR/SOAD Buffalo	48:00	24,25
20	Conduct Rehearsals	JSOTF/3-3 SFG/SOCCE/ODA	24:00	28,29
21	Conduct Rehearsals	JSOTF/3-160 SOAR/SOAD Sparrow	24:00	28,29
22	Conduct Rehearsals	JSOTF/16 SOW/20 SOS/Stalker	24:00	26
23	Await Execute Order	JSOTF	96:00	30
24	Final Mission Prep	JSOTF/3-160 SOAR/SOAD Buffalo	00:00	31,38,39
25	Final Mission Prep	JSOTF/75 Ranger/TF Ranger	00:00	38,39,41
26	Final Mission Prep	JSOTF/16 SOW/20 SOS/Stalker	00:00	41
27	Final Mission Prep	JSOTF/16 SOW/4 SOS/Sentry	00:00	40
28	Final Mission Prep	JSOTF/3-160 SOAR/SOAD Sparrow	00:00	31,32
29	Final Mission Prep	JSOTF/3-3 SFG/SOCCE/ODA	00:00	31,32
30	Issue Execute Order	JSOTF	00:00	31,32,37,38,39

Table 2. Mission Task List

Task #	Task Description	Unit Responsible	Duration HH:MM	Succeeding Tasks
31	Conduct Air Insertion	JSOTF/3-3 SFG/SOCCE/ODA	02:30	33
32	Conduct Air Insertion	JSOTF/3-160 SOAR/SOAD Sparrow	02:30	33,35
33	Conduct Infiltration	JSOTF/3-3 SFG/SOCCE/ODA	04:00	34
34	Conduct SR/TA	JSOTF/3-3 SFG/SOCCE/ODA	72:00	36
35	Return to Base	JSOTF/3-160 SOAR/SOAD Sparrow	02:30	42
36	Validate Target	JSOTF/3-3 SFG/SOCCE/ODA	00:00	37,47
37	Issue Strike Order	JSOTF	00:00	38,39,40,41,42,92
38	Conduct Air Insertion	JSOTF/75 Ranger/TF Ranger	02:30	43,44,46
39	Conduct Air Insertion	JSOTF/3-160 SOAR/SOAD Buffalo	02:30	43,44,46
40	Conduct Air Movement	JSOTF/16 SOW/4 SOS/Sentry	01:05	45
41	Support Air Insertion	JSOTF/16 SOW/4 SOS/Stalker	02:30	45,46
42	Await Extraction Launch	JSOTF/3-160 SOAR/SOAD Sparrow	00:00	69
43	Conduct LZ Operations	JSOTF/75 Ranger/TF Ranger	00:05	44,47,49,50,51,52,53,54 ,55,56
44	Move to Loiter Location	JSOTF/3-160 SOAR/SOAD Buffalo	00:20	48
45	Provide Strike Air Support	JSOTF/16 SOW/4 SOS/Sentry	00:00	84
46	Provide Strike Air Support	JSOTF/16 SOW/4 SOS/Stalker	00:00	85
47	Conduct Target Acquisition	JSOTF/3-3 SFG/SOCCE/ODA	00:00	70
48	Await Extraction Link-up	JSOTF/3-160 SOAR/SOAD Buffalo	00:00	67
49	Move to Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Blocker	00:04	58
50	Move to Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Blocker/Blocker 4	00:04	59
51	Move to SBF Position	JSOTF/75 Ranger/TF Ranger/TM Support West	00:05	60
52	Move to Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Support West/Blocker 2	00:05	61
53	Move to SBF Position	JSOTF/75 Ranger/TF Ranger/TM Support East	00:04	62
54	Move to Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Support East/Blocker 1	00:03	63
55	Move to Attack Position	JSOTF/75 Ranger/TF Ranger/TM Assault	00:04	64
56	Move to Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Assault/Blocker 3	00:03	57
57	Establish Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Assault/Blocker 3	00:02	65,66,80
58	Establish Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Blocker	00:04	65,66,73
59	Establish Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Blocker/Blocker 4	00:02	65,66,74
60	Occupy SBF Position	JSOTF/75 Ranger/TF Ranger/TM Support West	00:04	65
61	Establish Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Support West/Blocker 2	00:02	65
62	Occupy SBF Position	JSOTF/75 Ranger/TF Ranger/TM Support East	00:04	66
63	Establish Blocking Position	JSOTF/75 Ranger/TF Ranger/TM Support East/Blocker 1	00:02	65,66,78
64	Occupy Attack Position	JSOTF/75 Ranger/TF Ranger/TM Assault	00:03	68
65	Initiate SBF	JSOTF/75 Ranger/TF Ranger/TM Support West	00:00	67,68,72

Table 2 (continued)

Task #	Task Description	Unit Responsible	Duration HH:MM	Succeeding Tasks
66	Initiate SBF	JSOTF/75 Ranger/TF Ranger/TM Support East	00:00	67,68,71
67	Move to Extraction Site	JSOTF/3-160 SOAR/SOAD Buffalo	00:20	81,82
68	Conduct Raid	JSOTF/75 Ranger/TF Ranger/TM Assault	00:30	70,71,72,79
69	Move to Extraction Site	JSOTF/3-160 SOAR/SOAD Sparrow	02:30	87,94
70	Conduct BDA Post-strike Recon	JSOTF/3-3 SFG/SOCCE/ODA	00:15	86
71	Complete SBF	JSOTF/75 Ranger/TF Ranger/TM Support East	00:00	73,74,75,76,77,78,80
72	Complete SBF	JSOTF/75 Ranger/TF Ranger/TM Support West	00:00	73,74,75,76,77,78,80
73	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Blocker	00:08	82
74	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Blocker/Blocker 4	00:06	82
75	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Support West	00:09	82
76	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Support West/Blocker 2	00:07	82
77	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Support East	00:08	82
78	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Support East/Blocker 1	00:05	82
79	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Assault	00:10	82
80	Move to PZ	JSOTF/75 Ranger/TF Ranger/TM Assault/Blocker 3	00:02	82
81	Conduct Air Extraction	JSOTF/3-160 SOAR/SOAD Buffalo	02:30	90
82	Conduct PZ Operations	JSOTF/75 Ranger/TF Ranger	00:08	81,83,84,85
83	Conduct Air Extraction	JSOTF/75 Ranger/TF Ranger	02:30	88
84	Conduct Air Movement	JSOTF/16 SOW/4 SOS/Sentry	01:05	91
85	Support Air Extraction	JSOTF/16 SOW/4 SOS/Stalker	02:30	91
86	Conduct Exfiltration	JSOTF/3-3 SFG/SOCCE/ODA	00:10	87,94
87	Conduct Air Extraction	JSOTF/3-3 SFG/SOCCE/ODA	02:30	89
88	Conduct Debrief/AAR	JSOTF/75 Ranger	01:30	92
89	Conduct Debrief/AAR	JSOTF/3-3 SFG	01:30	92
90	Conduct Debrief/AAR	JSOTF/3-160 SOAR	01:30	92
91	Conduct Debrief/AAR	JSOTF/16 SOW	01:30	92
92	Conduct AAR	JSOTF	01:30	93
93	Stand Down JSOTF	JSOTF	00:00	-
94	Conduct Air Extraction	JSOTF/3-160 SOAR/SOAD Sparrow	02:30	90

Table 2 (continued)

APPENDIX C. SYSTEM DEMONSTRATION

A walk-through of many of the functions of SOMPASS using screen shots with descriptions is provided to assist users in learning how to use the system.

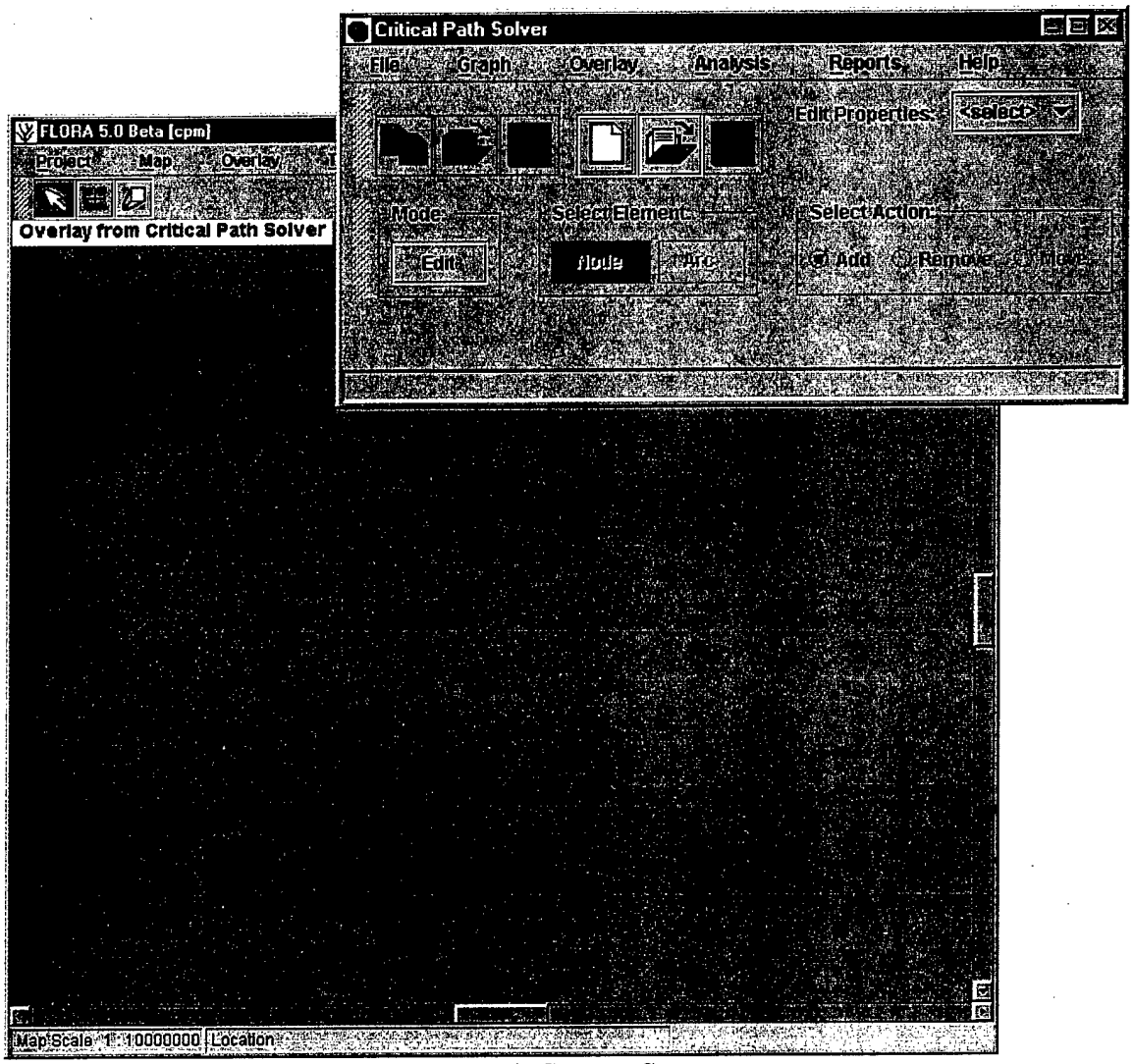


Figure 35. System Startup

The Critical Path Solver control panel and a blank Flora display will appear when SOMPASS is first run.

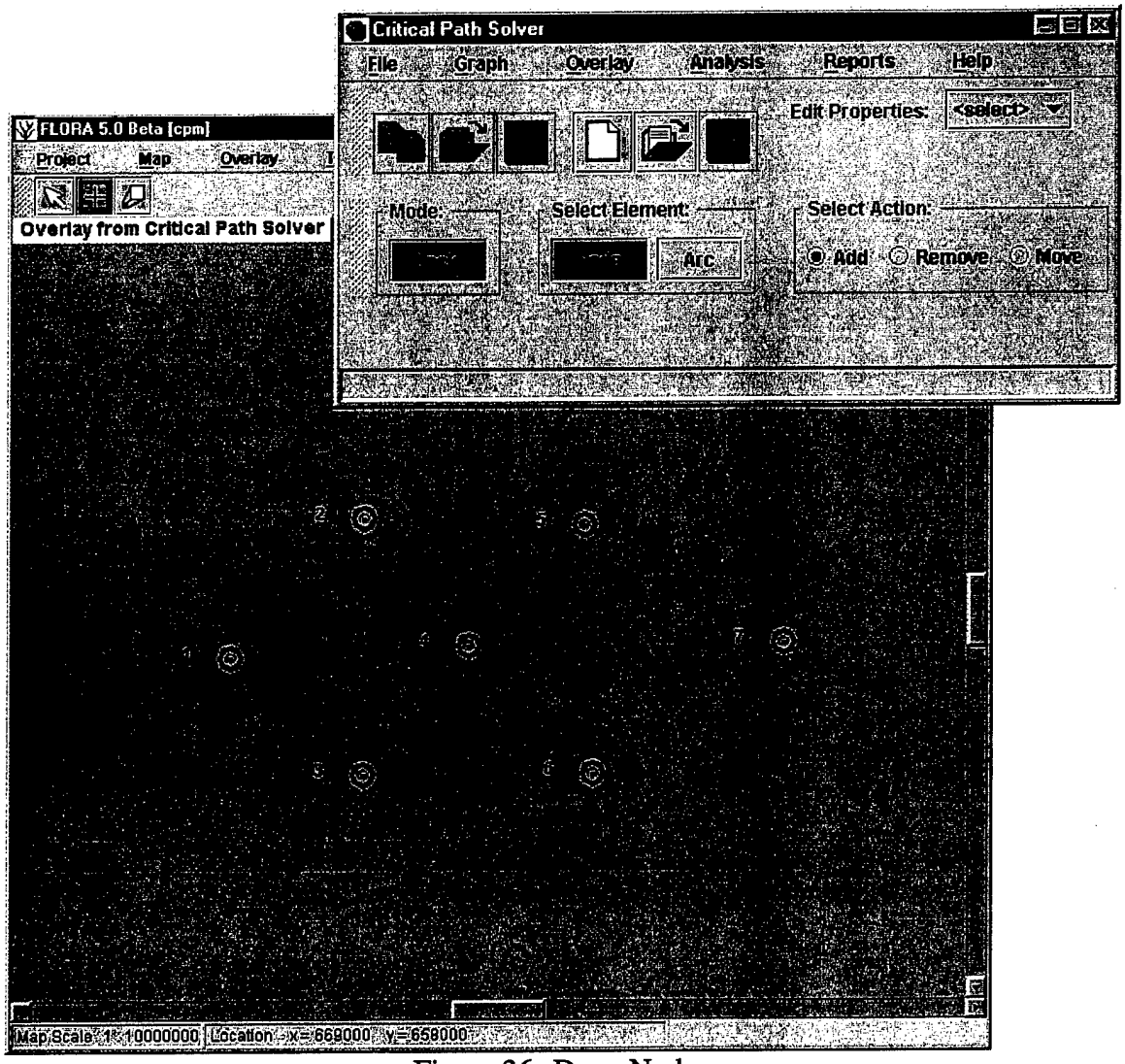


Figure 36. Draw Nodes

By using the Mode selector on the Critical Path Solver and the Edit button in Flora, the user can draw on the blank display to add, remove, or move nodes wherever a mouse click occurs. Confirmation is requested before a node is removed.

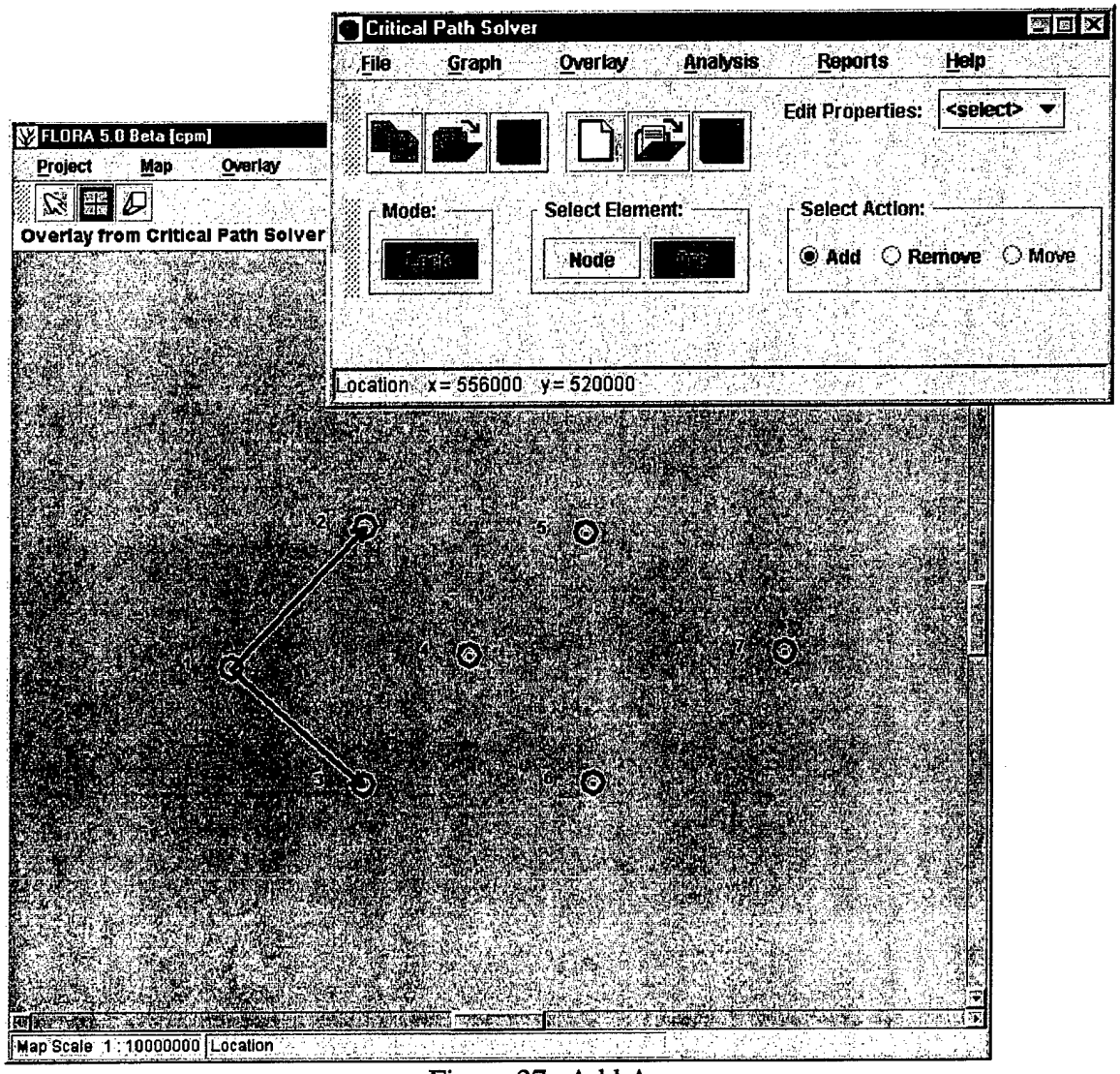


Figure 37. Add Arcs

Arcs can also be added or removed by clicking with the mouse on the starting node and then the ending node. Arcs cannot be moved because they will automatically move whenever the nodes they are associated with are moved. Arcs will also be removed automatically when either the start or end node is removed.

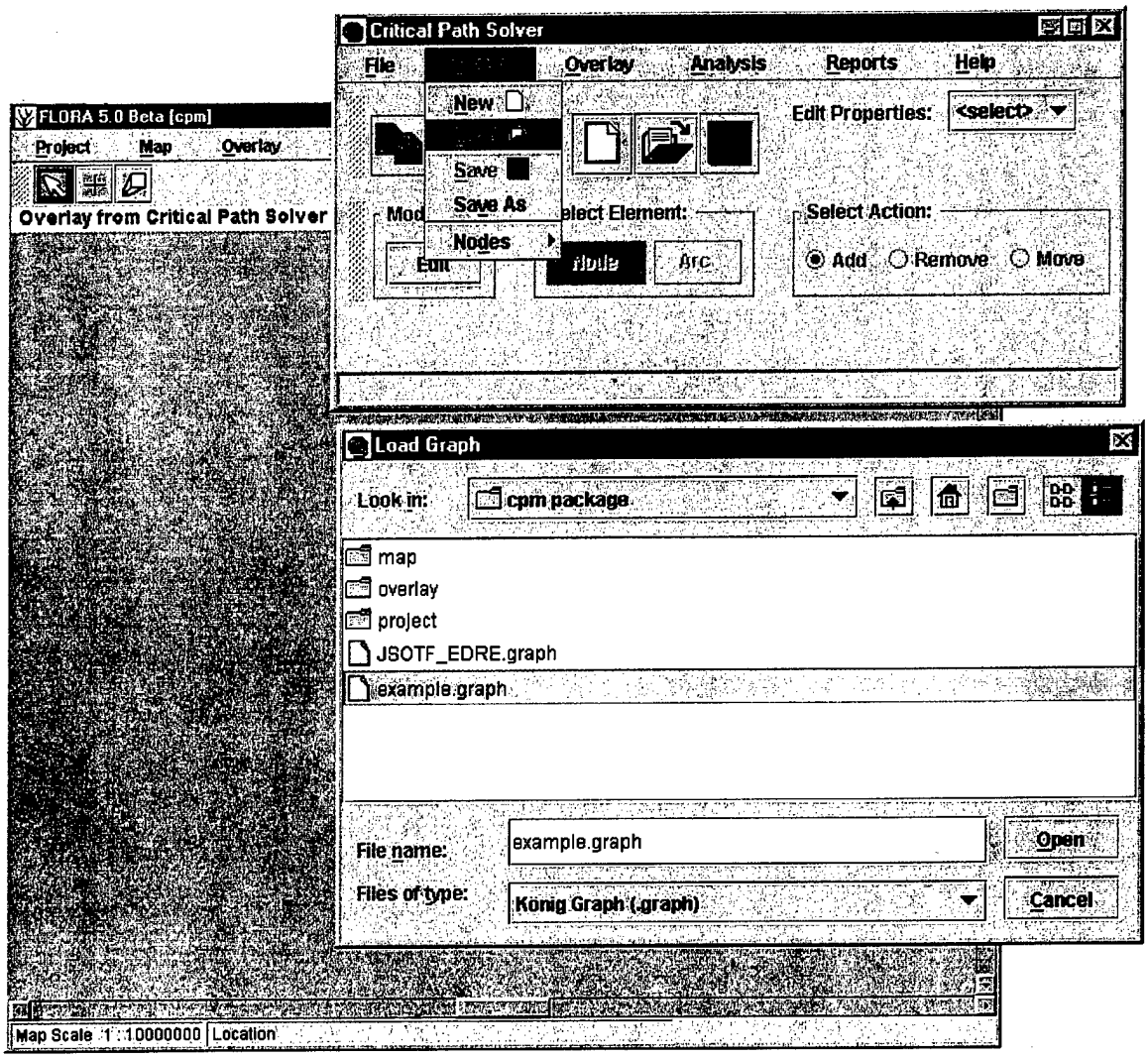


Figure 38. Load Graph

An already existing graph can be loaded to continue editing or perform other actions. The current graph can also be saved in a similar fashion using the Save As menu option.

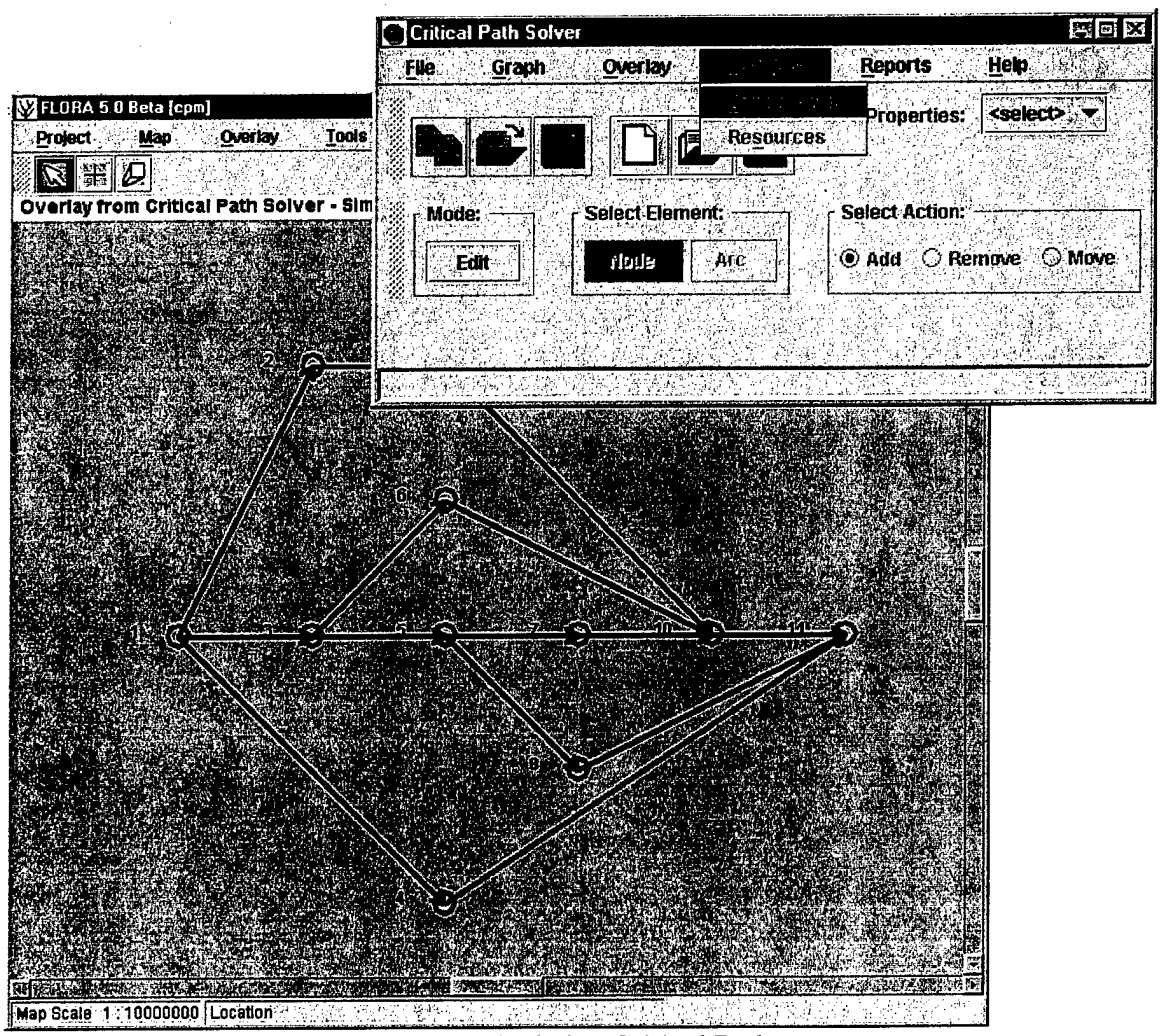


Figure 39. Solve Critical Path

Selecting the Critical Path option from the Analysis menu can solve the critical path for a graph. As mentioned in Chapter III, if a cycle is found, an error will be displayed, and the user must remove the erroneous arc that created the cycle before the critical path can be solved.

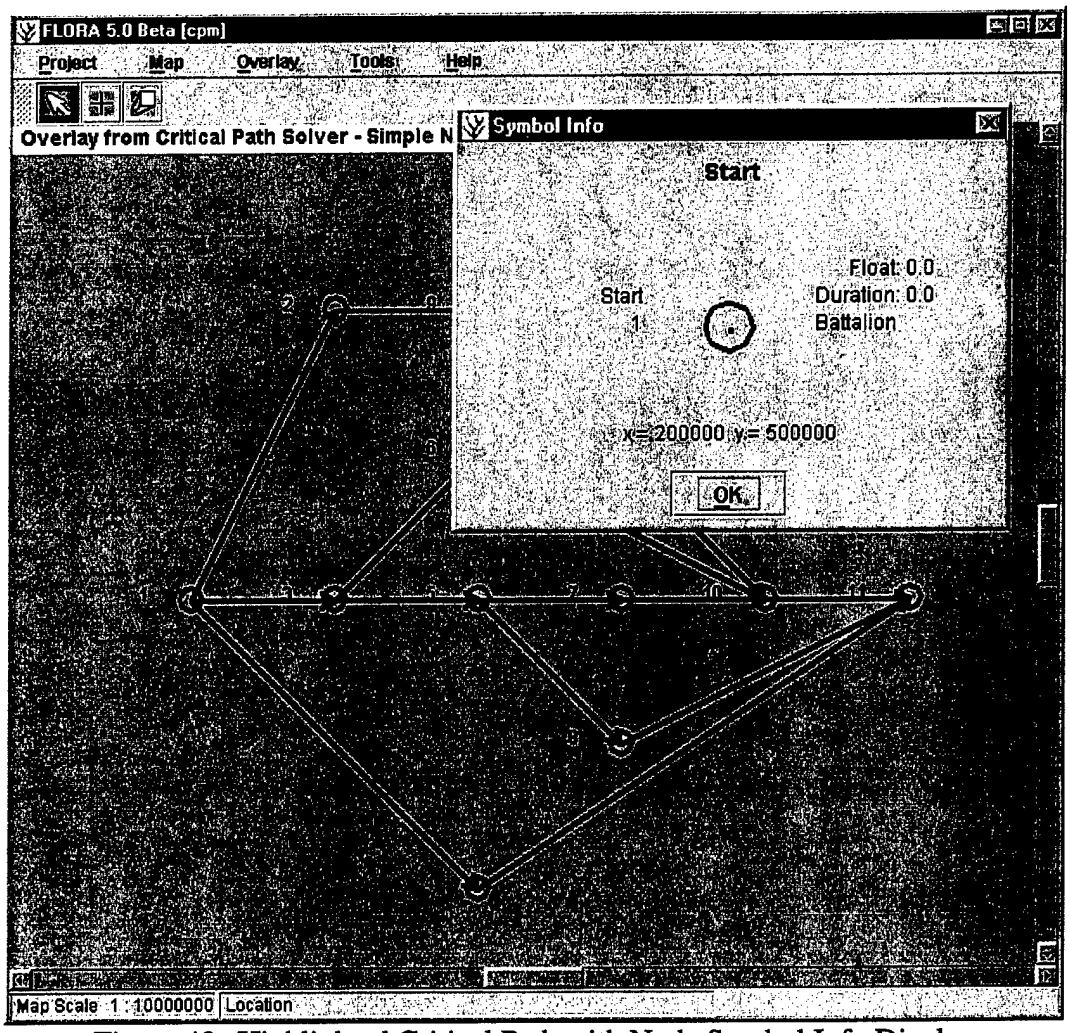


Figure 40. Highlighted Critical Path with Node Symbol Info Display

The critical path of the graph is highlighted in blue to alert the user to the nodes and dependencies that are crucial to the completion of the operation. Additional information about a node can be obtained by double clicking on it with the mouse. If Flora is in the Point mode then a Symbol Info window will appear; if Flora is in the Edit mode, an Edit Node window will appear to allow display, changing, adding, or deleting of node properties.

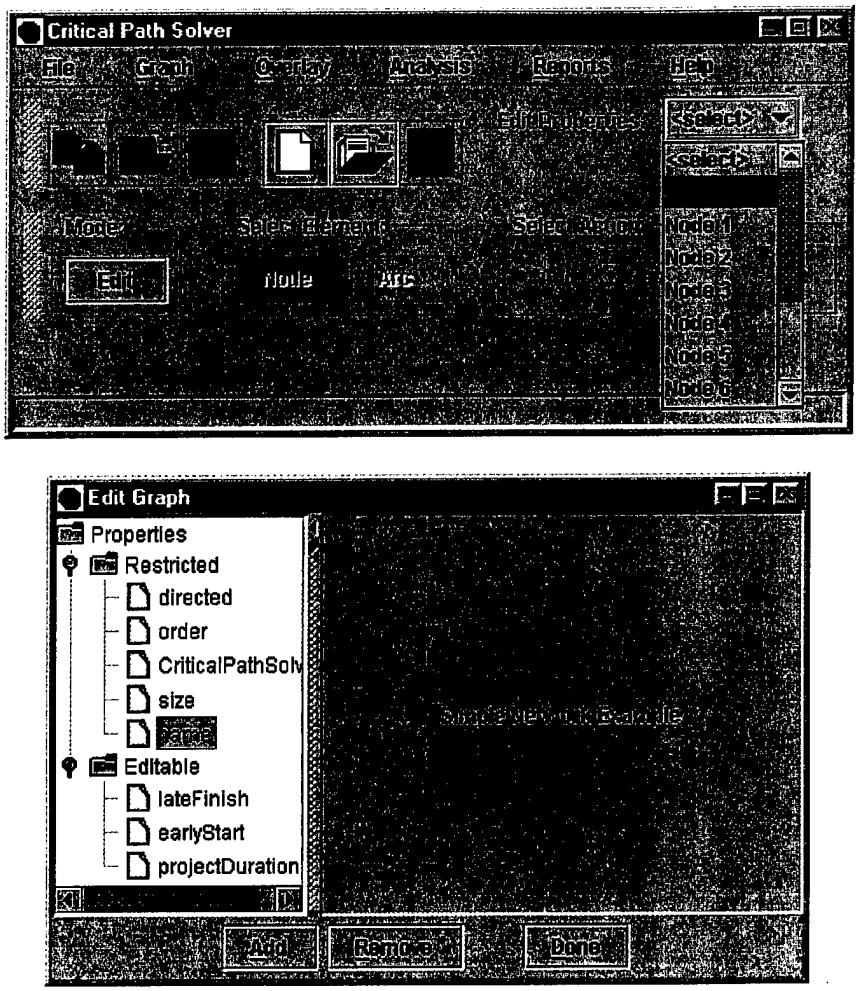


Figure 41. Edit Graph Window

Graph properties can be edited by selecting Graph from the Edit Properties list on the Critical Path Solver. Nodes can also be selected for editing this way in addition to double clicking on them in the Flora display. Restricted properties of either the graph or nodes cannot be edited or deleted, as they are crucial to the proper operation of SOMPASS.

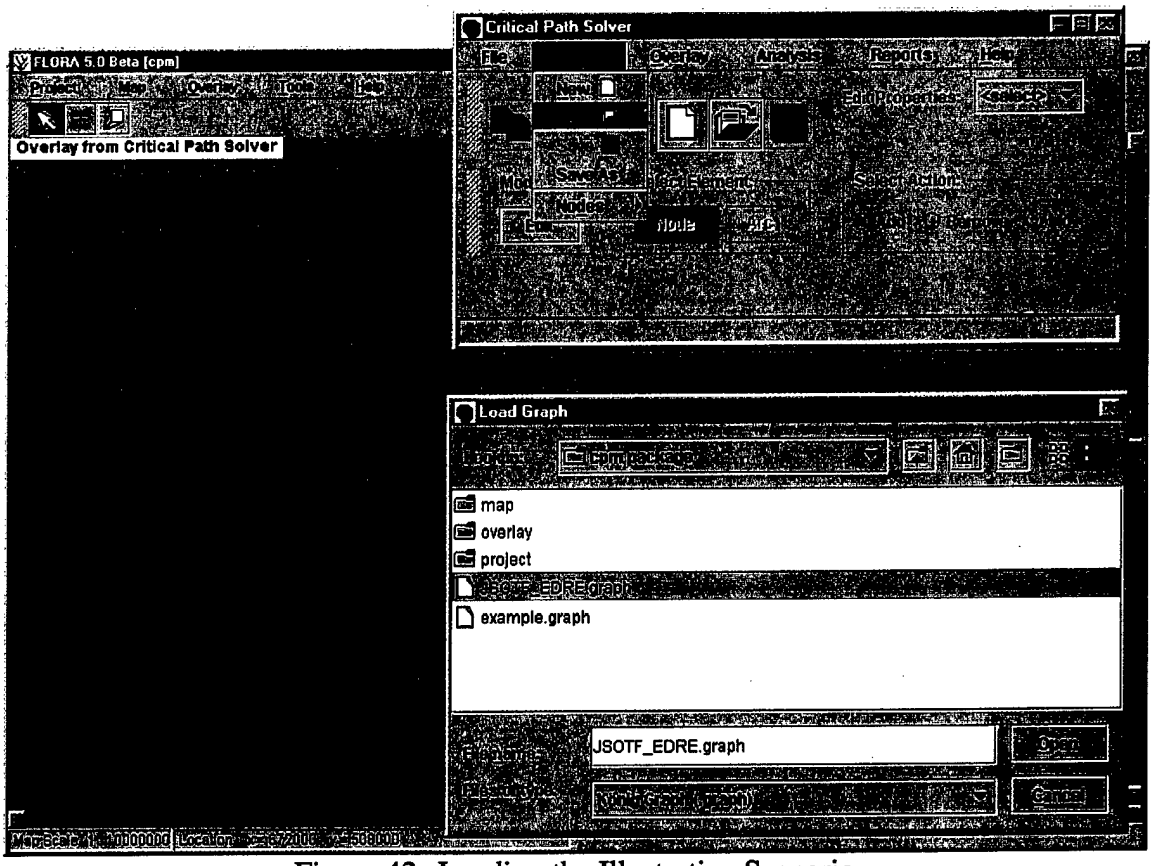


Figure 42. Loading the Illustrative Scenario

New projects and graphs can be loaded over a current one, but all unsaved information will be lost. Here the network for the illustrative scenario is loaded.

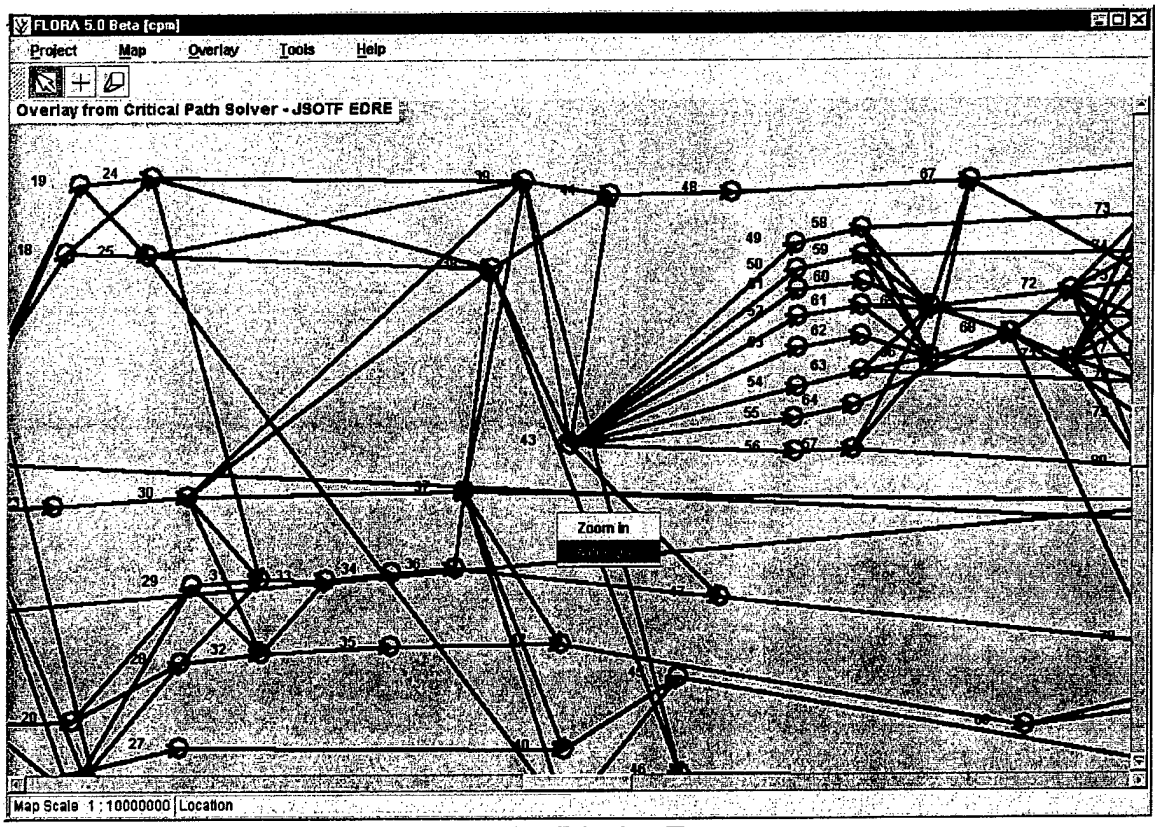


Figure 43. Display Zoom

The Flora display can be zoomed in to allow closer inspection of areas of the graph, or zoomed out to fit larger networks. Zooming can be accomplished by clicking the right mouse button in the Flora Point mode (as displayed) or Edit mode, using the option in the Map menu bar, or in the Flora Zoom mode by left-clicking to zoom in or right-clicking to zoom out.

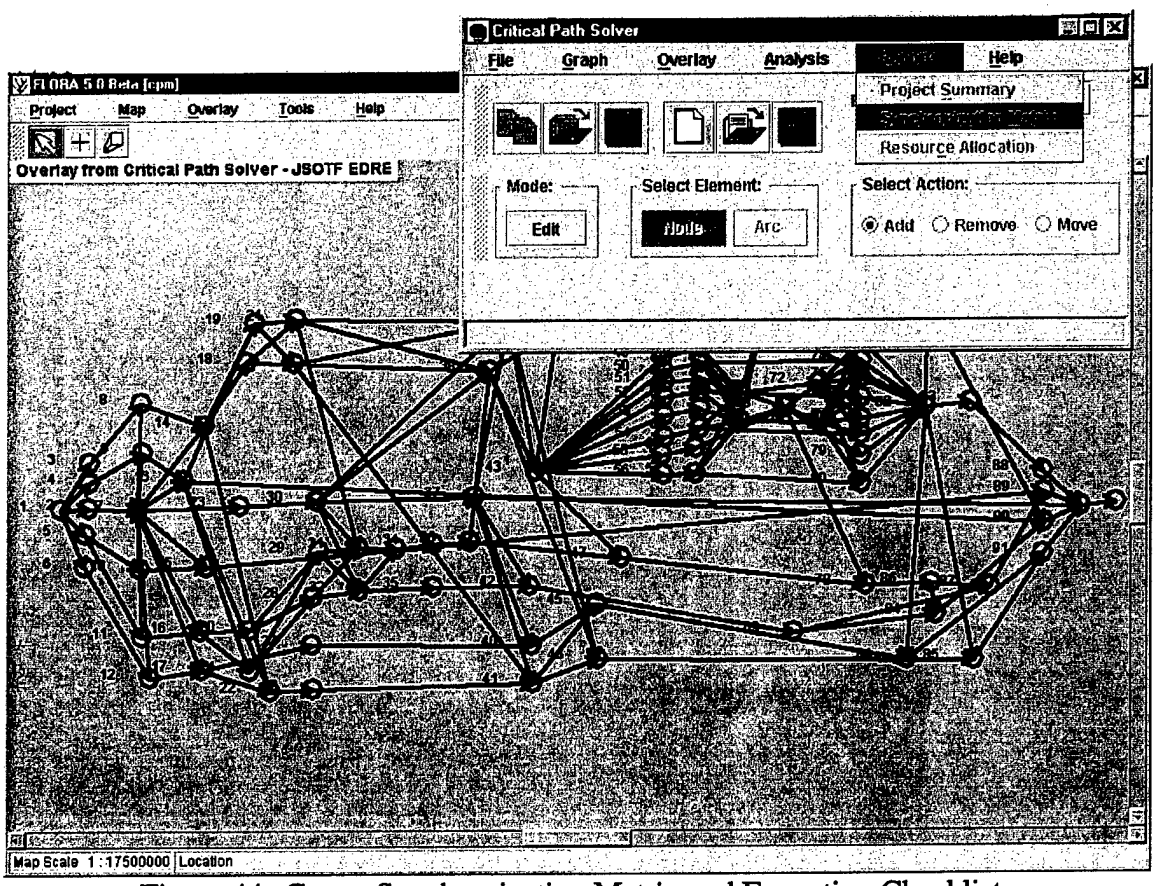


Figure 44. Create Synchronization Matrix and Execution Checklists

Selecting the Synchronization Matrix option from the Reports menu creates the synchronization matrix and execution checklists. The critical path will be solved at this time also, so it is not necessary manually solve the critical path first. The synchronization matrix and execution checklists will open in their own windows, and any previously existing ones will be replaced.

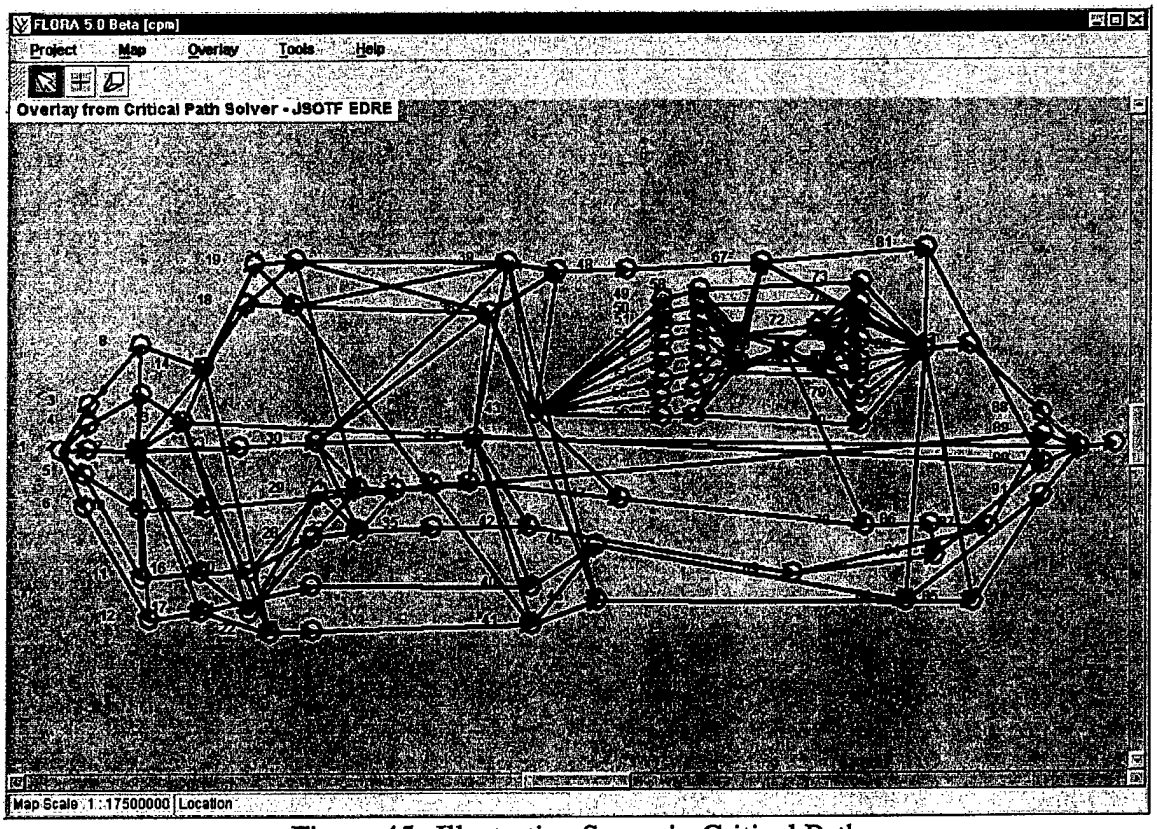


Figure 45. Illustrative Scenario Critical Path

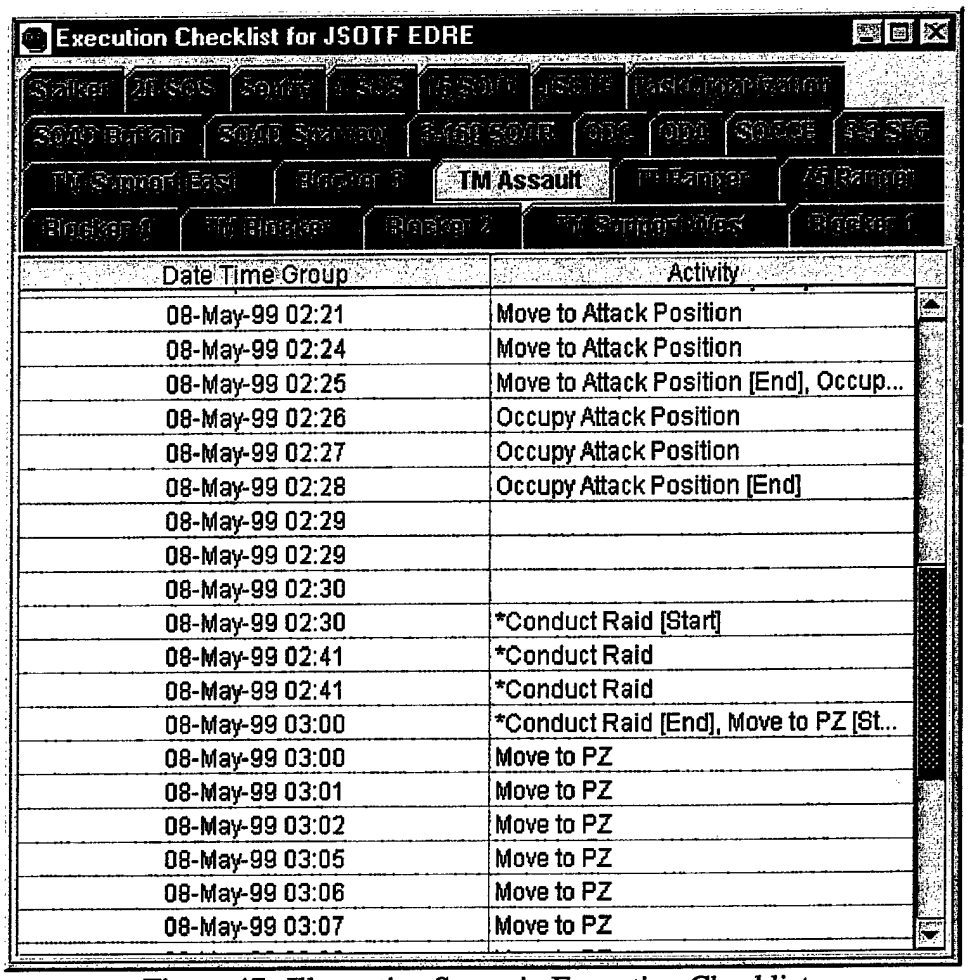
Note the parallel critical paths highlighted in blue. Parallel critical paths occur when concurrent activities have the same duration and have the same predecessor and successor tasks.

Synchronization Matrix for JSOTF EDRE

Unit	29-Apr-99 12:16	29-Apr-99 12:16	29-Apr-99 13:16	30-Apr-99 13:16	30-Apr-99 17:16	02-May-99 17:16	02-I
TaskOrganization							
JSOTF	*Stand Up JSO...	*Stand Up JSO...	Issue Warning ...	*Issue OPORD ...	*Issue OPORD ...	*Await Execute ...	*Aw
75 Ranger		*Receive Warni...	*Receive Warni...				
TF Ranger			*Conduct Initial ...	*Conduct Initial ...	*Complete Mis...	*Complete Mis...	*Co
TM Blocker							
Blocker 4							
TM Support W							
Blocker 2							
TM Support E							
Blocker 1							
TM Assault							
Blocker 3							
3-160 SOAR		*Receive Warni...	*Receive Warni...	*Conduct Initial ...	*Complete Mis...	*Complete Mis...	
SOAD Buffalo						*Conduct Rehe...	*Co
SOAD Sparrow						Conduct Rehea...	Con
3-3 SFG		*Receive Warni...	*Receive Warni...				
SOCCE							
ODC			*Conduct Initial ...	*Conduct Initial ...	Complete Missi...	Complete Missi...	
ODA			*Conduct Initial ...	*Conduct Initial ...	Complete Missi...	Complete Missi...	Con
16 SOW		*Receive Warni...	*Receive Warni...	*Conduct Initial ...	Complete Missi...	Complete Missi...	
20 SOS							
Stalker						Conduct Rehea...	Con
4 SOS							
Sentry						Final Mission P...	Fina

Figure 46. Illustrative Scenario Synchronization Matrix

Tasks on the critical path are preceded by an asterisk ("*") to highlight their importance. Task organization is apparent from the left column by level of indentation from higher level units.



Execution Checklist for JSOTF EDRE

TM Assault

Date Time Group	Activity
08-May-99 02:21	Move to Attack Position
08-May-99 02:24	Move to Attack Position
08-May-99 02:25	Move to Attack Position [End], Occup...
08-May-99 02:26	Occupy Attack Position
08-May-99 02:27	Occupy Attack Position
08-May-99 02:28	Occupy Attack Position [End]
08-May-99 02:29	
08-May-99 02:29	
08-May-99 02:30	
08-May-99 02:30	*Conduct Raid [Start]
08-May-99 02:41	*Conduct Raid
08-May-99 02:41	*Conduct Raid
08-May-99 03:00	*Conduct Raid [End], Move to PZ [St...
08-May-99 03:00	Move to PZ
08-May-99 03:01	Move to PZ
08-May-99 03:02	Move to PZ
08-May-99 03:05	Move to PZ
08-May-99 03:06	Move to PZ
08-May-99 03:07	Move to PZ

Figure 47. Illustrative Scenario Execution Checklists

As with the synchronization matrix, tasks on the critical path are preceded by an asterisk ("*"). Units are selected by clicking on their respective tabs.

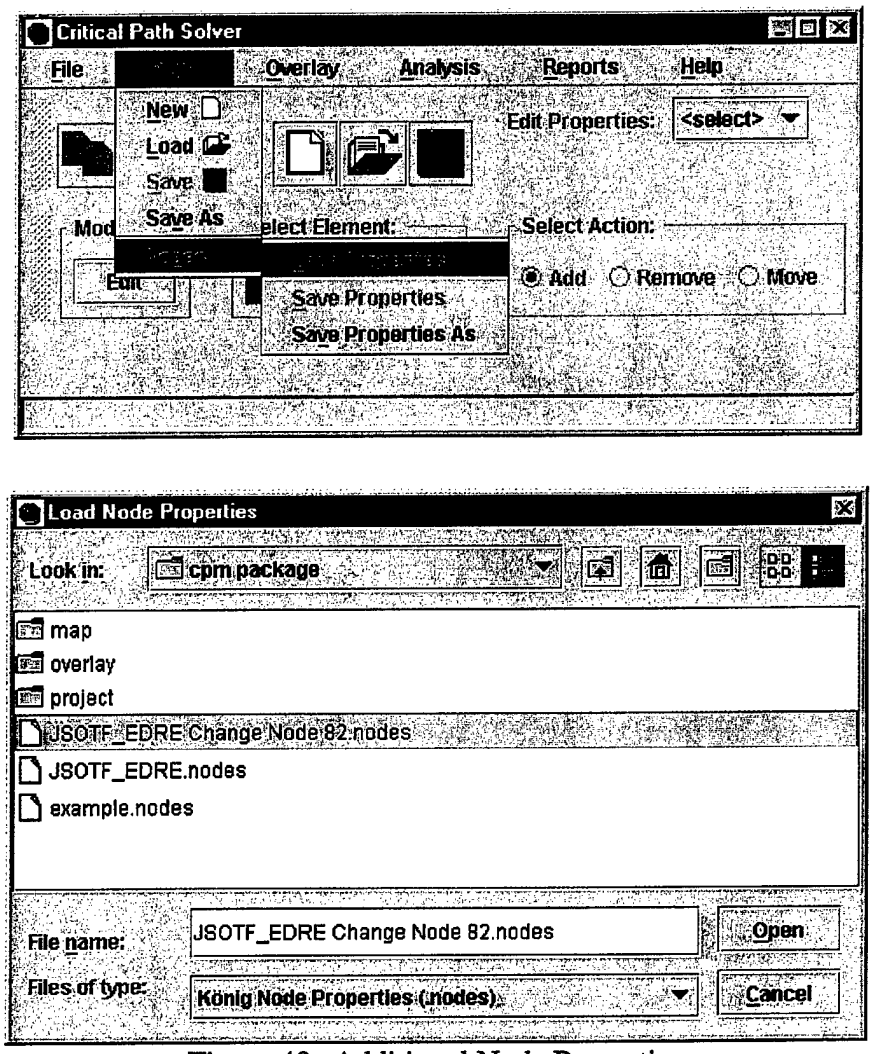


Figure 48. Additional Node Properties

Properties associated with nodes in a graph can be loaded or saved independently to add additional properties or overwrite current values. This provides additional flexibility in working with graphs, especially when multiple users are contributing to a project at different times and in different locations.

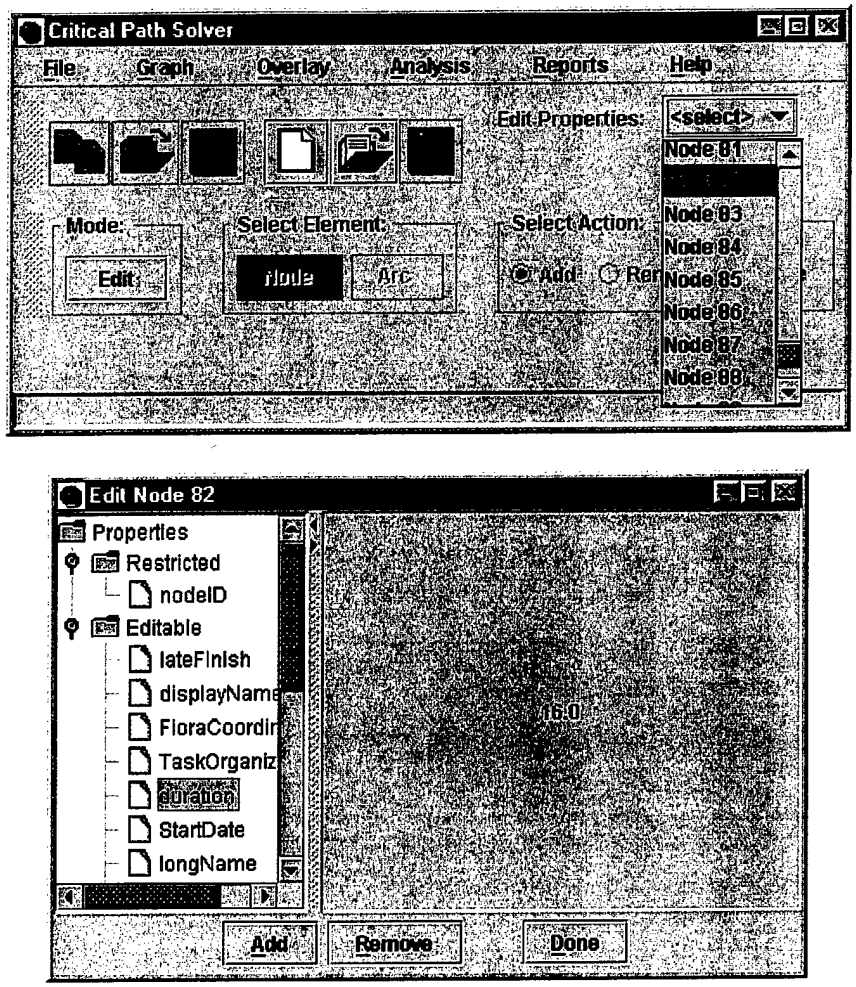


Figure 49. Edit Node Properties

Node properties can be edited, added, or deleted by selection from the edit list or double clicking in the Flora Edit mode. Here, the new duration for Task 82, PZ Operations, is now visible after loading in the updates node properties file for the illustrative scenario.

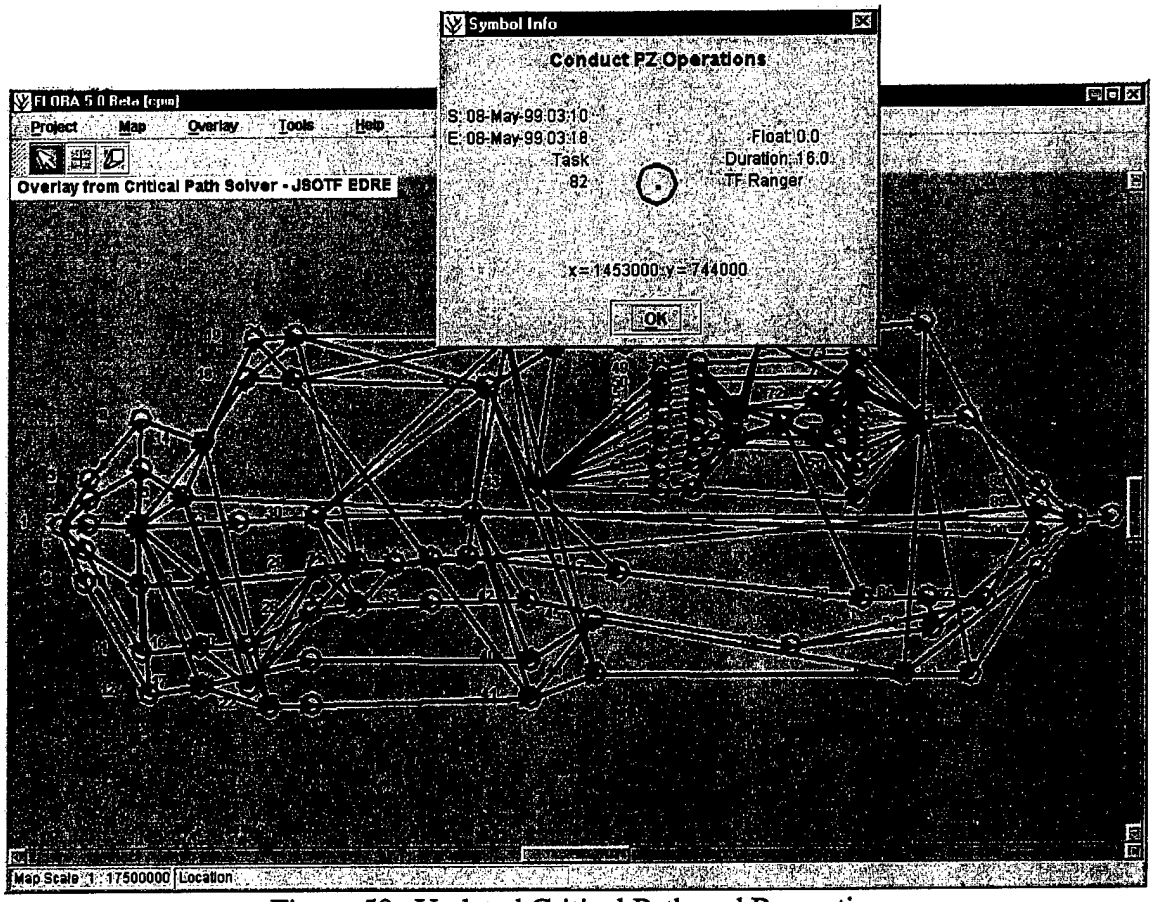


Figure 50. Updated Critical Path and Properties

The new mission critical path is highlighted in blue after changing the duration of one task beyond its available float time. The Symbol Info display reflects the updated properties and is also in blue to indicate that this node is on the critical path.

LIST OF REFERENCES

1. Chairman of the Joint Chiefs of Staff, *Joint Vision 2010: America's Military: Preparing for Tomorrow*, Chairman of the Joint Chiefs of Staff, 1996.

2. Joint Pub 1, *Joint Warfare of the Armed Forces of the United States*, Joint Staff, 10 January 1995.

3. Army Field Manual No. (FM) 100-5, *Operations*, Headquarters, Department of the Army, 14 June 1993.

4. Joint Pub 3-05, *Doctrine for Joint Special Operations*, Joint Staff, 17 April 1998.

5. Commander in Chief, United States Special Operations Command, *Special Operations Forces: The Way Ahead*, Commander in Chief, USSOCOM, 1998.

6. Commander in Chief, United States Special Operations Command, *SOF Vision 2020*, Commander in Chief, USSOCOM, 1996.

7. Joint Pub 6-0, *Doctrine for Command, Control, Communications, and Computer (C4) Systems Support to Joint Operations*, Joint Staff, 30 May 1995.

8. *USSOCOM C4I Strategy Into the 21st Century*, USSOCOM SOJ6-PI, March 1996.

9. Bradley, G.H. and Buss, A.H., "Dynamic, Distributed, Platform Independent OR/MS Applications — A Network Perspective," *INFORMS Journal on Computing*, v. 10, n. 4, Fall 1998.

10. *Special Operations Forces (SOF) Mission Planning, Analysis, Rehearsal, and Execution (MPARE): Theater Special Operations Command Requirements: SOF Mission Planning* [Draft], USSOCOM SOIO-C4I-MO, January 1999.

11. Bilyeu, A.L., *Concept for a Special Operations Planning and Analysis System*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1998.

12. Defense Information Systems Agency, "DISA Mission and Mandate." [http://www.disa.mil/missman.html]. 11 January 1999.

13. Defense Information Systems Agency, "Mission Needs Statement (MNS) for Defense Information System Network (DISN)." [http://www.disa.mil/DISN/docs/mns.html]. 1 October 1996.

14. Defense Information Systems Agency, "Defense Message System (DMS)." [http://www.disa.mil/D2/dms/index2.html]. 1 March 1999.

15. Defense Information Systems Agency, "C4I For The Warrior Brochure." [http://spider.osfl.disa.mil/fbsbook/fbsbook.html].

16. Defense Information Systems Agency, "GCSS Executive Summary." [http://www.disa.mil/gcss/execsum.htm]. 30 July 1998.

17. Memorandum, SOIO-C4I-MO, USSOCOM, Subject: SOCCENT/SOCACOM MPARE Requirements Inputs, 26 October 1998.

18. Memorandum, SOIO-C4I-MO, USSOCOM, Subject: SOCCENT Mission Planning Requirements, 28 October 1998.

19. Memorandum, SOIO-C4I-MO, USSOCOM, Subject: Trip Report, MPARE SOC Requirements Coordination with SOCACOM, 11 November 1998.

20. Memorandum, SOIO-C4I-MO, USSOCOM, Subject: Trip Report, MPARE SOC Requirements Coordination with SOCPAC, 30 November 1998.

21. Ravindran, A., Phillips, D.T., and Solberg, J.J., *Operations Research: Principles and Practice*, 2d ed., John Wiley & Sons, 1987.

22. Ahuja, R.K., Magnanti, T.L., and Orlin, J.B., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.

23. Morris, L.N., *Critical Path Construction and Analysis*, Pergamon Press, 1967.

24. Lockyer, K.G. and Gordon, J., *Critical Path Analysis and other Project Network Techniques*, 5th ed., Pitman Publishing, 1991.

25. Wiest, J.D. and Levy, F.K., *A Management Guide to PERT/CPM*, Prentice-Hall, 1969.

26. Kerzner, H, *Project Management: A Systems Approach to Planning, Scheduling and Controlling*, 3d ed., Van Nostrand Reinhold, 1989.

27. PC Webopaedia, "component." [http://webopaedia.internet.com/TERM/c/component.html]. 14 May 1998.

28. PC Webopaedia, "component software." [http://webopaedia.internet.com/TERM/c/component_software.html]. 18 September 1997.

29. Bradley, G.H., Buss, A.H., and Shaw, C.H., "An Architecture for Dynamic Planning and Execution Using Loosely Coupled Components," *NPS Research*, v. 8, n. 3, Naval Postgraduate School, Monterey, California, October 1998.

30. Loosely Coupled Components: Demonstration of MPARE Functionality [Presentation], 1999.

31. Java 2 [Computer programming language], Sun Microsystems, 1998.

32. Jackson, L.A., König [Computer software], 1999.

33. Jackson, L.A., *König User Guide (Beta 1.2)*, 26 March 1999.

34. Schrepf, N.J., Thistle [Computer software], 1999.

35. Schrepf, N.J., *Visual Planning Aid for Movement of Ground Forces in Operations Other Than War*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1999.

36. Wood, K., *OA4202, Network Flows and Graphs* [Class notes], Naval Postgraduate School, Monterey, California, Summer 1998.

37. Winston, W.L., *Operations Research: Applications and Algorithms*, 2d ed., PWS-Kent, 1991.

38. Joint Pub 5-0, *Doctrine for Planning Joint Operations*, Joint Staff, 13 April 1995.

39. Sun Microsystems, "Java Platform Ports." [http://java.sun.com/cgi-bin/java-ports.cgi].

40. Sun Microsystems, "Java 2 DataSheet." [http://java.sun.com/marketing/collateral/jdk1.2_ds.html].

41. Winston, P.H. and Narasimhan, S., *On to Java*, Addison-Wesley, 1996.

42. Horstmann, C.S. and Cornell, G., *Core Java 1.1*, v. 1-2, Sun Microsystems Press, 1997-1998.

43. Curtin, M., "Write Once, Run Anywhere: Why it Matters." [http://java.sun.com/features/1998/01/wora.html]. 19 May 1999.

44. Topley, K., *Core Java Foundation Classes*, Prentice-Hall, 1998.

45. PC Webopaedia, "thin client." [http://webopaedia.internet.com/TERM/t/thin_client.html]. 18 May 1998.

46. Landay, J., *CS160, User Interface Design, Prototyping, and Evaluation* [Class notes], University of California, Berkeley, California, Fall 1998. [http://bmrc.berkeley.edu/courseware/cs160/fall98/lectures/model-view-controller/index.htm].

BIBLIOGRAPHY

Air Force Doctrine Document (AFDD) 35, *Special Operations*, Headquarters, Department of the Air Force, 16 January 1995.

Arntzen, A., *Software Components for Air Defense Planning*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1998.

Armed Forces Staff College, *Joint Planning Orientation Course* [Course notes], National Defense University, 1998.

Army Field Manual No. (FM) 7-30, *The Infantry Brigade*, Headquarters, Department of the Army, 3 October 1995.

Army Field Manual No. (FM) 7-85, *Ranger Unit Operations*, Headquarters, Department of the Army, 9 June 1987.

Army Field Manual No. (FM) 31-20, *Doctrine for Special Forces Operations*, Headquarters, Department of the Army, 20 April 1990.

Army Field Manual No. (FM) 31-20-5, *Special Reconnaissance Tactics, Techniques, and Procedures for Special Forces*, Headquarters, Department of the Army, 23 March 1993.

Army Field Manual No. (FM) 101-5, *Staff Organization and Operations*, Headquarters, Department of the Army, 31 May 1997.

Bazaraa, M.S., Jarvis, J.J., and Sherali, H.D., *Linear Programming and Network Flows*, 2d ed., John Wiley & Sons, 1990.

Byous, J., "The Network Vehicle: A Smart Way to Go." [http://java.sun.com/features/1997/nov/javacar.html]. 6 April 1999.

Chairman of the Joint Chiefs of Staff Manual (CJCSM) 3500.05, *Joint Task Force Headquarters Master Training Guide*, Joint Staff, 15 April 1997.

Collins, J.M., *Special Operations Forces: An Assessment*, National Defense University Press, 1994.

Courtois, T., *Java Networking and Communications*, Prentice-Hall, 1998.

Commander's Battle Staff Handbook, U.S. Army Research Institute, Fort Benning, GA, 1 November 1992.

Federal Electric Corporation, *A Programmed Introduction to PERT*, John Wiley & Sons, 1963.

Folkes, S. and Stubenvoll, S., *Accelerated Systems Development*, Prentice-Hall, 1992.

Geary, D.M., *Graphic Java 2: Mastering the JFC*, v. 2, 3d ed., Sun Microsystems Press, 1999.

Ginzberg, M.J., Reitman, W., and Stohr, E.A., Eds., *Decision Support Systems*, North-Holland, 1982.

Joint Pub 1-02, *Department of Defense Dictionary of Military and Associated Terms*, Joint Staff, 23 March 1994 [as amended through 7 December 1998].

Joint Pub 3-0, *Doctrine for Joint Operations*, Joint Staff, 1 February 1995.

Joint Pub 3-05.3, *Joint Special Operations Operational Procedures*, Joint Staff, 25 August 1993.

Joint Pub 3-05.5, *Joint Special Operations Targeting and Mission Planning Procedures*, Joint Staff, 10 August 1993.

Joint Pub 5-00.2, *Joint Task Force Planning Guidance and Procedures*, Joint Staff, 13 January 1999.

Joint Pub 6-02, *Joint Doctrine for Employment of Operational/Tactical Command, Control, Communications, and Computer Systems*, Joint Staff, 1 October 1996.

Joint Special Operations Task Force (JSOTF) Headquarters Standing Operating Procedures (SOP), Headquarters, Special Operations Command, Atlantic Command (SOCACOM), 1 August 1998.

Linstone, H.A., *Multiple Perspectives for Decision Making: Bridging the Gap between Analysis and Action*, North-Holland, 1984.

Martino, R.L, *Project Management*, MDI Publications, 1968.

McRaven, W.H., *SPEC OPS — Case Studies in Special Operations Warfare: Theory and Practice*, Presidio Press, 1995.

Meredith, D.D., Wong, K.W., Woodhead, R.W., and Wortman, R.H., *Design & Planning of Engineering Systems*, 2d ed., Prentice-Hall, 1985.

Sanders, N., *Stop Wasting Time: Computer-Aided Planning and Control*, Prentice-Hall, 1991.

Special Operations Forces Reference Manual [CD-ROM], USSOCOM SOOP-P/T, January 1998.

Swanson, L.A. and Pazer, H.L., *PERTSIM: Text and Simulation*, International Textbook, 1969.

Tactical Standing Operating Procedures (TACSOP), Headquarters, 2d Battalion, 504th Parachute Infantry Regiment, 9 January 1994.

United States Special Operations Forces Posture Statement, Office of the Assistant Secretary of Defense (Special Operations/Low-Intensity Conflict), 1998.

USSOCOM Publication 1, USSOCOM SOJ3, 25 January 1996.

Wysocki, R.K., Beck, R., Jr., and Crane, D.B., *Effective Project Management*, John Wiley & Sons, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Gordon H. Bradley, Code OR/Bz 10
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

4. Professor Arnold H. Buss, Code OR/Bu 5
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

5. LTC Joel R. Parker, Code OR/Jp 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

6. MAJ Leroy A. Jackson 1
   TRADOC Analysis Center – Monterey
   Naval Postgraduate School, P.O. Box 8692
   Monterey, California 93943-0692

7. United States Special Operations Command 2
   Attn: SORR-SC
   7701 Tampa Point Blvd.
   MacDill AFB, Florida 33621

8. COL(P) Stanley A. McChrystal 1
   Commander, 75th Ranger Regiment
   Attn: AORG-RGR, Bldg. 2834
   Fort Benning, Georgia 31905

9. CPT Keith A. Hattes 1
   453 Fairview Ct.
   Hartland, Wisconsin 53029-1501