AFRL-IF-RS-TR-2004-283
Final Technical Report
October 2004

PLAN CRITIQUING AND LOOK-AHEAD CONSTRAINT REASONING FOR ACTIVE TEMPLATES

Alphatech, Incorporated

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-283 has been reviewed and is approved for publication

APPROVED: /s/

               WAYNE A. BOSCO
               Project Engineer

FOR THE DIRECTOR: /s/

               JAMES A. COLLINS, Acting Chief
               Information Technology Division
               Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE OCTOBER 2004	3. REPORT TYPE AND DATES COVERED Final Mar 00 – Jun 04

4. TITLE AND SUBTITLE
PLAN CRITIQUING AND LOOK-AHEAD CONSTRAINT REASONING FOR ACTIVE TEMPLATES

5. FUNDING NUMBERS
C   - F30602-00-C-0040
PE - 63760E
PR - ATEM
TA - P0
WU - 05

6. AUTHOR(S)
Christopher M. White

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Alphatech, Incorporated
50 Mall Road
Burlington Massachusetts 01803

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency   AFRL/IFTB
3701 North Fairfax Drive   525 Brooks Road
Arlington Virginia 22203-1714   Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2004-283

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Wayne A. Bosco/IFTB/(315) 330-3528/ Wayne.Bosco@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
The primary objective of this program was to develop knowledge-based Active Templates technology to help Special Operations Forces (SOF) planning staff quickly build robust and agile plans, and then monitor their execution. To this end, we conducted research and development in four major areas. First, we developed a SOF knowledge representation language and example scenario for representing mission plans and constraints on those plans. Second, we developed a template-based natural language user interface to allow mission planners to input mission plan details in a format that is understandable to both the machine and human operator. Third, we designed a branch plan specification environment with a plan critiquer and look-ahead constraint reasoner, and demonstrated the feasibility of these designs by developing a temporal constraint reasoner. Finally, we developed and transitioned to operational use adaptive planning software that supports collaborative mission planning and orders production. We conclude that although successful, many areas for further research and development still exist in all four areas including further research in plan representations to support reasoning over branch plans, implementing the branch plan environment designs, and enhancing the adaptive planning software to interface with additional support tools including a knowledge base for reasoning over the plan.

14. SUBJECT TERMS
Templated-Based Natural Language Environment, SOF Knowledge Representations, Branch Planning, Collaborative Workflow Mission Planning

15. NUMBER OF PAGES
70

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

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

This technical report describes work performed on the Plan Critiquing and Look-Ahead Constraint Reasoning for Active Templates (contract number F30602-00-C-0040) project conducted for the Air Force Research Laboratory (AFRL)-Rome Research Site in Rome, NY and the Defense Advanced Research Projects Agency (DARPA) in Arlington, VA. The primary objective of this program was to develop knowledge-based Active Templates technology to help Special Operations Forces (SOF) planning staff quickly build robust and agile plans, and then monitor their execution.

Our initial approach was to transition to Active Templates reusable ontologies, knowledge bases, and technology for knowledge-based plan critiquing originally developed under DARPA's High Performance Knowledge Bases (HPKB) program to critique courses of action for ground forces operations. As part of this effort, we provided the original ontologies to the Active Templates community. We also adapted and extended these ontologies and knowledge bases to the SOF planning domain. These new ontologies were then used as a common knowledge representation for later development efforts in Active Templates.

Using the SOF ontologies as a common knowledge representation of the situation and the plan, we developed a suite of lightweight tools for natural language parsing for translating between natural language inputs and this common knowledge representation. As a demonstration of the potential applicability of these lightweight natural language parsing tools, we integrated a template-based natural language interface for specifying tactical tasks into the Active Templates Temporal Plan Editor (TPE). We also developed a standalone template-based natural language application for specifying a Course of Action.

Continuing our extension and use of our SOF ontologies, we designed a plan critiquer and look-ahead constraint reasoning system. These designs extend the Active Templates Temporal Plan Editor to support the generation and critiquing of branch plans and sequels. The resulting plans can be critiqued for feasibility and completeness using parametric, interval, disjunctive, and resource constraints. As an initial proof of concept, we developed a temporal constraint reasoner and integrated it with the TPE. The resulting system maintains and enforces temporal constraints between tasks defined in the temporal plan. Using these constraints, the system minimally modifies the plan in order to maintain constraint consistency or warns the user when consistency cannot be maintained.

For our final development effort, we designed and implemented a planning tool to guide an Army Special Forces Operational Detachment Alpha (ODA) team through the Military Decision Making Process (MDMP). This tool is designed to increase the planning tempo. It allows the ODA team to import commander's guidance and situation information into their planning documents, share information from between steps of the planning process, and generate doctrinally defined and formatted documents as intermediate and end products of the planning process. Development of this tool was performed under the guidance of the Army Special Operations Battle Lab (ARSOBL) and the Special Operations Mission Planning Office (SOMPO). The tool was successfully tested by several ODAs in controlled and experimental settings including Millennium Challenge 2002 and has been included as part of the Special Operations Mission Planning Environment (SOMPE) being provided to all ODAs.

1. INTRODUCTION

ALPHATECH's Active Templates project was based on two important principles. First, current methodologies and technologies allow rapid construction of an expressive, broad-coverage model of a military planning domain sufficient to support automated plan critiquing. Second, military planners are highly structured people – they express planning material using a circumscribed and well-defined doctrinal vocabulary. The primary objective of this program was to develop knowledge-based Active Templates technology to help Special Operations Forces (SOF) planning staff to quickly build robust and agile plans, and then monitor their execution.

The effort began by focusing on transitioning reusable ontologies, knowledge bases, and technology for knowledge-based plan critiquing originally developed under DARPA's High Performance Knowledge Bases (HPKB) program to Active Templates in order to critique courses of action for ground forces operations. The investigation included the following:

- adapting and extending the HPKB ontologies and knowledge bases to cover the SOF planning domain;

- providing a comprehensive suite of lightweight tools for natural language parsing for translating between natural language inputs and a common knowledge representation of the plan;

- providing a plan critiquer and look-ahead constraint reasoning capability to help ensure the feasibility of plans under multiple uncertain future contingencies;

- developing a planning tool to guide an Army Special Forces ODA team through the Military Decision Making Process (MDMP).

The scope of this investigation was limited to Direct Action operations and did not incorporate situation monitoring.

The next section describes the methods, assumptions, and procedures used in investigating each of the four areas of interest. Results and their discussion are then presented. This is then followed by conclusions drawn from these results. Recommendations for future action based on the results of the study are then provided. A list of references is provided as the last section. A list of symbols, abbreviations, and acronyms used in the report is provided in Appendix A.

2. METHODS, ASSUMPTIONS, AND PROCEDURES

2.1 SOF KNOWLEDGE REPRESENTATIONS

As part of our work on the Active Templates project, we transitioned and expanded knowledge representations developed under the DARPA HPKB Battlespace Challenge Problem. For this effort, we collected information from various sources pertaining to the DARPA HPKB Battlespace Challenge Problem (CP) and made this material available to the Active Templates program via the ALPHATECH HPKB website (http://www.alphatech.com/protected/hpkb/index.html; login: hpkb; password: hpkb411). The following information is available at this website:

1.   Documents describing the challenge problem;

2.   A selected list of relevant army sources (i.e., field manuals and student texts);

3.   An interactive knowledge document created by ALPHATECH, Inc. during the HPKB program for searching and browsing background challenge problem information;

4.   Documents describing Course of Action (COA) analysis;

5.   Scenario materials developed for the challenge problem; and

6.   COA ontology definitions present in CycL for use in Cycorp's Cyc knowledge base system.

Using this information as our base, we focused on extending the COA grammar and the COA Ontology to support Direct Action (DA) missions. Our initial efforts were focused on developing a plan representation sufficient to support inter-operation of a task editor with the Temporal Plan Editor. The plan representation needed to contain sufficient information to support editing of the plan through different views of the plan such as a COA sketch, COA statement, and synchronization matrix. It also had to allow for the development of plan patterns to use as templates for developing a complete plan, and it needed to facilitate critiquing of the plan after it had been tailored to a specific mission.

As part of our preparations for developing a plan representation, we attended a CycL knowledge representation training course and a knowledge acquisition workshop for plan patterns. The CycL course provided us with insights into the Cyc knowledge representations used for the HPKB program. These insights were beneficial in our development of a plan representation for Active Templates. The plan pattern workshop provided us with a deeper understanding of the details that plan-goal graphs in the Tool Interchange Manager (TIM) might contain. While working with the subject matter experts who attended the workshop and helping them learn about developing concept graphs and plan-goal graphs, we learned several domain issues about setting up and maintaining an Intermediate Staging Base (ISB). We also saw how many elements of a plan-goal graph for rescuing hostages could be reused in other direct action missions.

As another part of our preparations for developing a plan representation, we developed an airfield seizure scenario based on a scenario used during the Active Templates program kick-off. During this scenario an airport facility was captured from enemy forces in order to support a Noncombatant Evacuation Operation (NEO). Our scenario consists of the following five phases: alert, marshal, and deploy; forced entry operations; lodgment operations; battle handover; and

redeploy. A Subject Matter Export (SME) developed a draft OPLAN, and a COA statement and sketch for the second and third phases of this scenario.

Our first step in expanding the knowledge representations developed under the DARPA HPKB Battlespace Challenge Problem was to develop a knowledge representation language (KRL) to support plan representation and critiquing. This KRL was developed in collaboration with the Small Business Innovation Research (SBIR) Phase I team for An Instructable Agent for Rapid Knowledge Entry using Natural Language. The goal of the KRL design was to develop a language that is lightweight and efficient, yet expressive enough to enable the kind of representation and inference that we required.

We then collaborated with the Agile Commander Integrated Product Team (IPT) to refine the COA ontology plan representations for military operations. Key constraints for the plan representation were that:

1.   It must be easy to group activities in the COA by unit (e.g., tasks of the main effort) and area of operations (e.g., the tasks of the main attack);

2.   It must be easy to specify the plan incrementally and in almost any order. For example,

   a. Specify the purpose of the main effort.

   b. Specify purposes for the supporting efforts.

   c. Specify tasks for the main and supporting efforts.

3.   It must be easy to extend the representation to support multi-phased operations.

Finally, in collaboration with the Intermodal Lift SBIR Phase I team, we further refined the COA ontology plan schema to include Constraint Satisfaction Problem (CSP) variables for use by the plan critiquer, parameters to help simplify the constraint network, and extensions for defining parametric, interval, disjunctive, and resource constraints. These extensions were then used during our investigation into plan critiquing and look-ahead constraint reasoning.

2.2 TEMPLATE-BASED NATURAL LANGUAGE USER INTERFACE

As part of our overall goal of providing a unified planning and editing environment for military operations, we investigated providing a method that would allow users to interact with a Knowledge Base (KB) via natural language (words, phrases, and sentences). The goal of this research was to allow users to specify courses of action in as natural and realistic a manner as possible. Free text input is of course the most natural and realistic specification method; however, simple parsing techniques for free text have limitations in their ability to properly understand the user's intended meaning. Among these limitations is the difficulty in handling ambiguous statements where it is not clear "what goes with what." In the statement "**Forces conduct attack by fire and breach obstacle in order to deny Red unit the ability to maneuver in passage point,"** it is not clear whether the purpose (**deny Red unit the ability to maneuver in passage point**) goes with both of the actions (**conduct attack by fire and breach obstacle**) or with just the second action (**breach obstacle**). Ambiguity and other language constructs presented challenges for both the User Interface (UI) component and the Natural Language (NL) component of Active Templates. We were therefore faced with the task of developing a non-trivial Natural-Language

component that could handle the complexity of real Course Of Action (COA) statements as needed by our users.

To handle this complexity, we set out to develop an interface that would allow a user to selectively specify information in a textual fashion by building up sentence fragments via drilldown templates. These templates allow the user to specify information in any order, not necessarily in a left to right progression as is typical in applications using a shift reduce parser. The results of the user input appear as a complete sentence once all required information is entered. This allows the user to verify the input in a format that he is familiar with, natural language, and yet also guarantee that the input is in a format that the system can interpret and understand.

We also desired to make the system easy to rapidly tailor to new situations. Thus, we wanted a backend language for specifying the input templates, which could then be used to generate the user front-end input environment. We chose to use Prolog's Definite Clause Grammar (DCG) as the backend language for specifying the grammar. This grammar is then interpreted in order to generate the drilldown templates and input options provided to the user in the graphical front-end interface. Our basic assumption is that non-terminals in the grammar represent drilldown templates, and terminals in the grammar represent final input selection options.

In order to test the usefulness of the new natural language interface, we wanted to integrate it into existing military planning tools to form a multi-modal environment for military planning. In our vision of Active Templates, we viewed it as essential to develop a multi-modal environment for military planning. Users should be able to switch effortlessly between text-based and graphical-based planning tools with the changes made in one tool being automatically reflected in the other tools. We therefore chose to integrate this new technology into the SOFTools Temporal Plan Editor (TPE). The TPE allows users to graphically lay out an operation spatially and temporally, using a standard set of icons to represent locations, resources, and actions. We also developed a Course of Action Editor that could be combined with the TPE and Task Editor to form a multi-modal planning environment.

To facilitate development of a multi-modal planning environment, we developed a client/server architecture that would allow plan objects to be stored in a central knowledge base and accessed by each client application or tool in the planning environment. The client/server model treats each component (e.g., TPE, Task Editor, and COA Editor) as a client process with its own internal structure and rules. A server process communicates simultaneously with all of its clients, maintaining coherence as the user specifies a course of action. In this framework, the TPE is simply another client communicating graphically-specified information to the server, which in turn notifies the other clients to update their text contents to reflect this information. Minimally, we would expect the visually-specified temporal layout of activities in the TPE to be reflected by prepositions like "before," "after," and "during" in the text representation, and vice-versa.

With our client/server model in place, we were ready to create a multi-modal environment. First, the TPE and knowledge base were integrated into the environment. This made the information captured by the TPE available to other clients as they were integrated into the environment. Initially, the TPE only supported actions representing the movement of a resource from one location to another. Since military planning and COAs also specify tactical tasks that occur at a location, we extended the TPE to allow the user to specify actions occurring at a location over a given duration. The tactical tasks that we included are: bypass, clear, cover, guard, isolate, occupy, prepare, retain, river crossing operations, screen, secure, and seize.

Starting with the Course of Action (COA) grammar provided in Technical Memo-HPKB-COA-10 (TM-HPKB-COA-10, 1999), we extended and modified the grammar to support the description of tactical tasks including information about the agent, type of operation, task, and purpose associated with the action. We used this grammar to develop a template-based natural language Task Editor, which we then integrated into the TPE in order to allow users to specify detailed information when defining a location based tactical task. When editing a task in the TPE, the user can invoke a Task Editor for that task in order to provide a textual description of the task. The Task Editor and TPE operate together via the client/server model to act as a single integrated planning environment.

Using the template-based natural language technology, we next built a multiple panel Course of Action (COA) Editor that would allow a user to specify multiple-phrases for each part of a Course of Action including the Close Battle, Deep Battle, End State, Risk, Security, Fires, Obstacles, Reserves, and Rear statements as defined by Technical Memo-HPKB-COA-10 (TM-HPKB-COA-10, 1999). The COA Statement Editor provides planners with a text-based interface for specifying mission plans and complements graphical planning environments such as the TPE. As part of the independent Integrated Course of action Critiquing and Elaboration System Concept Experimentation Program (ICCES CEP) project, we integrated the COA Statement Editor with the nuSketch COA sketching tool during which it was used by end-users to develop COA statements.

2.3 PLAN CRITIQUER AND LOOK-AHEAD CONSTRAINT REASONER

As another part of our goal of providing a unified planning and editing environment for military operations, we investigated designs for a plan editor that allows the user to develop mission plans. We assumed that a useful mission plan editor would allow users to plan missions that contain branches and sequels. We also assumed that such a system would critique the plan for feasibility based on parametric, interval, disjunctive, and resource constraints. Finally, we assumed that the system would also need to assess the plan for completeness. In other words, the system would need to check whether alternative mission branches had been defined for all known situations.

To support the development of this Branch Plan Editor with plan critiquer and look-ahead constraint reasoner, we first developed a white paper outlining our concepts and system designs. Key elements of the design included a Plan Critiquer, a Decision Point Editor, a Situation Editor, a Task Editor, and branch plan development and visualization capabilities.

- The *Plan Critiquer* uses asset, time, and location information available in the Temporal Plan Editor (TPE) to alert the planning staff of potential problems in the temporal plan in regards to feasibility, completeness, and correctness. Initially, the Plan Critiquer based its constraints solely on information currently available from the TPE plus additional details about assets and locations, such as vehicle range and global coordinates. As our branch plan environment was designed, we extended the Plan Critiquer to consider additional information such as temporal constraints, specified world conditions, and dependencies between tasks and maneuvers.

- The *Decision Point Editor* allows the user to define a set of decision points. The user may specify multiple branch plans for each decision point and include with each branch plan the conditions for when a branch plan will be used. The conditions provide assistance to the planning staff and commander for deciding when to activate a branch plan. The conditions also provide needed information to the Look-Ahead Constraint Reasoner for determining which branch plans are valid given world conditions and previous decisions.

- The *Situation Editor* allows the user to define possible contingencies and world conditions, and how they interact. For example, the user may define a contingency for whether helicopter support is available and indicate that helicopter support is unavailable whenever the weather condition is bad. The user can use the conditions defined in the Situation Editor to define when a branch plan in the Decision Point Editor is valid.

- The *Task Editor* allows the user to specify detailed information about tasks and maneuvers. In addition to the asset information, which can already be specified in the Temporal Plan Editor for maneuvers, the Task Editor allows the user to specify dependencies between tasks and maneuvers. These dependencies include temporal constraints on the start, end, and duration of a task or maneuver; and information about the type of operation and purpose of the task or maneuver.

The white paper was reviewed by Active Templates subject matter experts and a prototype Temporal Constraint Reasoner was developed and integrated into BBN's Temporal Plan Editor. We then developed a prototype consisting of the basic infrastructure needed to support the several editors required for the Branch Plan Editor as well as the constraint engine needed for plan critiquing and look-ahead constraint reasoning.

2.4 SPECIAL FORCES MDMP ADAPTIVE PLANNING SOFTWARE

As the final effort under this contract, we developed the Adaptive Planning Software to support Army Special Forces doing mission planning. The Adaptive Planning Software (APS) allows the user to create a forms based workflow that shares information between different sections of the workflow. APS v1 is tailored to the needs of the Special Forces. Specifically, it targets the planning process of an Operational Detachment Alpha (ODA). A workflow template has been developed to guide the ODA through the MDMP (Military Decision-Making Process) as tailored and described in the Special Forces Planning Techniques (SFPT) Student Handout for the Special Forces Qualification Course dated April 2000. APS allows an ODA to tailor the workflow template to their specific needs. The ODA can then use this workflow template to guide them through the mission planning process. Information entered into the workflow during early stages of the planning process is presented again in later sections when needed for further mission planning, briefing, and document preparation.

Development of the APS was carried out in three main phases. Phase 1 was designed to develop a prototype system that demonstrates the key concepts of collaborative mission planning, supporting text, image, and spreadsheet data, developing a workflow template for the MDMP, providing a method to edit the workflow template, sharing information between fields in the templates, importing information from DC2S, including security marking with all data, generating mission planning documents and briefings, and developing a presentation viewer. Phase 2 focused on transitioning the prototype developed in Phase 1 to an operational system. During this phase, the system went through extensive internal and end-user testing. Phase 3 focused on transitioning the APS technology to a Special Operations Mission Planning Environment (SOMPE) contractor for transitioning into the joint community.

3. RESULTS AND DISCUSSION

3.1 SOF KNOWLEDGE REPRESENTATIONS

3.1.1 Airfield Seizure Scenario

The Phase II COA sketch and statement, Phase III COA statement, draft Operations Plan (OPLAN), partial Temporal Plan, and synchronization matrix for the Airfield Seizure (AFS) scenario are provided in Appendix B. The COA statements are broken into the following paragraphs: mission statement, close battle statement, reserves, security, deep operations, rear operations, fire support statement, obstacles, risk statement, and end state. Phase II focuses on the forced entry operations of the scenario. During this phase, the enemy air defenses are neutralized and friendly troops parachute onto the runway. From here, the friendly troops clear and secure the airfield facilities. Phase III focuses on the lodgment operations for supporting the evacuation of noncombatants including the securing of the ground evacuation route between the Freedomtown bridge and the airport.

Although this airfield seizure scenario and the products developed for it represent only one possible plan for one type of mission, it provided many insights into the complexities of representing the knowledge associated with a direction action mission. We have the interactions of multiple teams to achieve a common purpose, the dependencies between phases of the operation, decision points and potential branch plans, and different representations of a common plan such as a graphical sketch and a textual statement, which complement each other to provide a complete picture of the plan. The scenario materials developed here served as the basis for testing many of our theories and systems in later parts of the project.

3.1.2 Knowledge Representation Language

In collaboration with the SBIR Phase I team for An Instructable Agent for Rapid Knowledge Entry using Natural Language, we developed a knowledge representation language (KRL) to support plan representation and critiquing. Features of this KRL include:

- It supports the functionality of a frame system including *classes*, *instances*, *slots*, and *facets* (Minsky, 1975). Classes are data structures that represent a situation or object in the real world. Classes have a set of attributes, or slots, that describe the class. Each slot has attributes of its own, called facets, which describe the slot. Some typical facets of slots are *value*, which specifies the value of the slot, *cardinality*, which specifies the number of values the slot can take, and *type,* which specifies constraints on the type of the objects that fill the value facet. Instances are an extension of a *Class*. They inherit all of the slots of their parents, with the added restriction that they may not add new slots, and can only modify the value facets of the slots they inherit.

- It is designed to be consistent with the Open Knowledge Base Connectivity (OKBC) (Chaudhri, 1998) standard for knowledge base interoperability.

- It has a clear formal semantics via translation into a subset of first-order logic (FOL).

- It is designed to be extensible beyond a strictly frame-based paradigm to a larger subset of first-order logic. This is achieved by coding the representations in XSB (Sourceforge), a variant of Prolog, using the Prolog notation for their translation into FOL. The KRL can be extended simply by admitting a larger subset of Prolog into the language and extending "background" inference operations like property inheritance to support the more expressive language.

- It supports the concept of *contexts*. Contexts (Guha, 1991) are a method of dividing up a knowledge base into smaller self-consistent chunks, providing a form of modularity that can improve efficiency of inference by localizing reasoning to subsets of the knowledge base.

- It has a front-end parser that allows users to utilize a simple syntax when describing classes and instances. This syntax frees the user from having to remember to include a context argument to each predicate and allows for a more compact specification of slot type and cardinality constraints. It also shields the user from some XSB-specific details such as declaring slot names as Hilog terms.

Figure 1 shows example knowledge representations in which a context, two classes, and an instance are defined. A parser developed by the An Instructable Agent for Rapid Knowledge Entry using Natural Language Phase I SBIR team converts this information into prolog assertions, which can then be reasoned over. Each knowledge representation file is defined within a specific context, which is declared in the form:

```
Context contextName [: superContext]
```

where the specification of a super context is optional. Classes are declared in the form:

```
Class className [: superClass]
<Class slot definitions>
End
```

where the specification of a super class is optional and a class slot definition is declared in the form:

```
slotName : slotType (slotCardinality) [= slotValue]
```

Valid values for slot cardinality are, 0, 0+, 1, and 1+, where 0 and 1 indicate optional verses required and '+' indicates multiple possible values versus a single possible value. The specification of a slot value is optional. An instance is declared in the form:

```
Instance instanceName : superClass
<Instance slot definitions>
End
```

where an instance slot definition is declared in the form:

```
slotName = slotValue
```

Slot values must begin with a lowercase letter, a digit, or be enclosed in quotes (single or double). Slot values may contain whitespace as long as the entire value is enclosed in quotes. Context, class, instance, and slot names must follow the same rules as slot values. Finally, the '%' character indicates comments. Anything between a '%' and a newline is considered a comment.

```
% Define a context for processes
Context process_context

% A generic class for any step in a process
Class step
                name : string (1) = 'Step'
                predecessors : step (0+)
                successors : step (0+)
                inputs : product (1+)
                outputs : product (1+)
                duration : number (1)
End

% A linear step
Class linear_step : step
                name : string (1) = 'Linear Step'
                predecessors : step  (1)
                successors : step (0)
End

% A specific linear step
Instance linear_step_497 : linear_step
                name = 'Linear Step 497'
                predecessors = linear_step_496
                successors= linear_step_498
                inputs = product_15
                outputs = product_16
                duration = 10
End
```

Figure 1. Example knowledge representations.

3.1.3 Plan Representation

Using this knowledge language, we developed a knowledge representation of a military operation. The complete knowledge representation is provided in Appendix C. We distinguish operations from tasks in the following way; operations are "containers" for tasks and sub-operations while tasks include doctrinally specified *tactical tasks* and other needed activities. Operations have a wide-ranging applicability including being used to:

1. Represent a mission (task and purpose);

2. Provide a top-level object for structuring a COA (decomposing a mission into sub-missions);

3. Package up several tasks constituting a main effort or a supporting effort;

4. Package up several tasks constituting a main attack;

5. Represent a phased operation as a sequence of sub-operations;

6. Represent shifting of the main effort.

Figure 2 shows a segment of the plan representation for military operations. The *militaryOperation* class contains several slots including *agent*, *tasks*, *location*, and *purpose*. The slot names are followed by their types. For example, the *agent* slot is filled by an object of type *militaryForce*. The slot type is followed by the slot's cardinality. A 1 indicates that the slot must have one and only one value assigned to it for each instance of a *militaryOperation*. A 1+ indicates one or more values must be assigned to the slot. A 0+ indicates that the slot may have zero or more values assigned to it. As stated earlier, an operation is a container for tasks and sub-operations. Sub-operations are defined as *militaryOperations* and tasks are defined as members of the *task* class. There are several types of tasks including tactical tasks and movements. Tactical tasks (*tacticalTask*) are defined as a subclass or specialization of the *task* class, and specific tactical tasks such as clear and secure are subclasses of the *tacticalTask* class.

```
class militaryOperation
   agent : militaryForce                    (1)
   tasks : task                             (1+)
   location : location                      (1)
   purpose : string                         (1)
   opInterval : interval                    (1)
   endState : unitPosture                   (0+)
   subOperations : militaryOperation        (0+)
   supportingEfforts : militaryOperation    (1+)
   mainEffort : militaryOperation           (1)
end

class task
   agent : militaryForce                    (1)
   taskInterval : interval                  (1)
end

class tacticalTask : task
   typeOfOperation : operationType          (1)
   formOfManeuver : maneuverType            (1)
   purpose: string                          (1)
end
```

Figure 2: Segment of the military operation ontology

Figure 3 shows some example instances that could exist for part of the airfield seizure scenario. The first instance represents the third phase of the airfield seizure. During this phase, Task Force Charlie secures Route Blue in order to enable evacuation of US embassy personnel. The second instance specifies the activities for Team 4 during this part of the operation. The third and fourth instances further specify some of the tasks that Team 4 will execute.

```
instance airfieldSeizurePhase3 : militaryOperation
   agent = taskForceCharlie
   tasks = secureRouteBlue
   location = cordilereAirport
   purpose = "enable evacuation of US embassy personnel"
   opInterval = opInt1
   subOperations = team1op, team2op, team3op, team4op, firesOp
   supportingEfforts = team1op, team2op, team3op, team4op
   mainEffort = team1op
end

instance team4op : militaryOperation
   agent = team4
   tasks = moveToRouteBlue, clearRouteBlue, secureRouteBlue, moveToLodgment
   location = cordilereAO
   purpose = "enable movement of US embassy personnel to lodgment"
   opInterval = team4opInt
end

instance moveToRouteBlue : movement
   agent = team4
   taskInterval = moveInt1
   startLocation = cordilereAirport
   endLocation = routeBlue
   transport = hmmwv1, hmmwv2 , hmmwv3 , hmmwv4 , hmmwv5, hmmwv6
end

instance clearRouteBlue : clearTT
   agent = team4
   taskInterval = clearInt1
   typeOfOperation = attack
   location = routeBlue
   transport = hmmwv1, hmmwv2 , hmmwv3 , hmmwv4 , hmmwv5, hmmwv6
   purpose = "enable securing Route Blue"
end
```

Figure 3: Example instances of a military operation

3.1.4 Constraints

In addition to specifying a plan representation for military operations, we also defined four classes of constraints for critiquing a mission plan: temporal, interval, resource, and parametric. Temporal constraints are point-based, single interval constraints over a time domain. Interval constraints associate a pair of time intervals using temporal constraints on their endpoints. They are used to represent preconditions and effects of tasks, and compile into point constraints. Resource constraints encode resource usage (e.g. fuel consumption) as point-based, single-interval constraints over resource domains. Parametric constraints are arbitrary procedural constraints.

Figure 4 shows an example of interval constraints on the transport vehicles associated with an instance of the *movement* class. The first constraint states that the vehicles must be at the start location immediately prior to the start of the movement. The second constraint states that the vehicles will be at the end location immediately after the end of the movement. Similar constraints could also be written for the cargo associated with the movement.

```
class movement
    startLocation: location      (1)
    endLocation: location        (1)
    moveInterval: interval       (1)
    transport: vehicle           (1+)
    cargo: resource              (1+)

    % constraints on the transport vehicle
    constraints:
        forall (transport, startLocation, moveInterval)
            exists(preInterval: interval)
                [at( preInterval, transport, startLoc), meets(preInterval, moveInterval)],
        forall(transport, endLocation, moveInterval)
            exists(postInterval: interval)
                [at(postInterval, transport, endLoc), meets(moveInterval, postInterval)]

    % similar constraints on the cargo
    ...
end
```

Figure 4: Example class definition with interval constraints

3.2 TEMPLATE-BASED NATURAL LANGUAGE USER INTERFACE

The Template-Based Natural Language User Interface (TBNLUI) allows one to create free-flowing text that can be understood by a computer application. The idea behind TBNLUI, as explained in the Software User's Manual (ALPHATECH TBNL Applications UM, 2002), is to allow the user to generate free-flowing text similar to the natural text that one would include in a free-text comment box; however, in TBNLUI, the user is guided through a series of templates that allows the user to communicate understandable ideas to the computer. TBNLUI is a general interface that may be integrated into an existing application or tailored to collect specific information in a standalone application. Integration, as explained in the Software Developer's Guide (ALPHATECH TBNL Applications SDG, 2002), is accomplished by utilizing the TBNLUI Server. The server displays a TBNLUI by way of a NLGUI Client. This allows applications to access the abilities of the TBNLUI without needing to know anything about how the TBNLUI works or how to display it. Regardless of how the TBNLUI is accessed, the basic concept for composing free-flowing text remains the same. Once the text has been composed, the parse structure can be exported in text format so that other applications can access and modify the information. The results of the TBNLUI research were presented at the Active Templates Spring 2001 Principal Investigator Meeting.

3.2.1 Grammar Development

We modified the Course of Action (COA) grammar provided in Technical Memo HPKB-COA-10 (TM-HPKB-COA-10, 1999) to support the description of tactical tasks and to support the generation of COA statements. The resulting grammar was used in the Task Editor and COA Statement Editor applications. During the grammar development process, we found the following rules to be helpful in developing a grammar that generates a user-friendly interface:

1. Keep the displayed grammar shallow. A user has an easier time learning to navigate the drilldown templates if the grammar is shallow. Breadth allows the user to see at a glance what information needs to be provided.

2. Use automatic expansion of unambiguous grammar rules. A grammar can be written using generic rules that create more depth as long as unambiguous options are automatically expanded by the interface. This decreases the effort required by the user to reach a leaf node while allowing general purpose grammars to be written.

3. Use general rules where possible. Combine similar rules with minor difference using constraints to limit what needs to be shown to the user based on previous selections.

4. Use optional rules that reference a required rule. This allows users to see an optional template that when selected, automatically expands the required grammar rule. The following is an example of an optional rule (`optRule`) that expands to a required rule (`rule`).

   ```
   optRule --> rule; [].
   rule --> a, b, c.
   ```

5. Write a conjunction as an optional template the follows a required list item. This allows the user to enter the first item and then later decide to expand the list without changing the original drilldown decision. The following is an example conjunction of items. It is made up of a required item (`item`) followed by an optional list of items (`andItems`). The list of items (`andItems`) provides the user with the option of appending another item to the list.

   ```
   items --> item; andItems.
   andItems --> [and], items; [].
   ```

3.2.2 User Interface interpretation of grammar

The Template-Based Natural Language Applications are based on a Template class designed to work with a Prolog Definite Clause Grammar (DCG) (Bratko, 1990). As explained in the Software Developer's Guide (ALPHATECH TBNL Applications SDG, 2002), the Template class reads and interprets the grammar rules to generate the user interface. As the user selections, grammar rules are expanded in-line in order to give the look and feel of expanding and specifying phrases of a sentence. Grammar rules containing terminal symbols are grouped and used to display fixed-text and selection menus. Free variables in the grammar are displayed as free-text entry widgets and allow the user a greater degree of flexibility in specifying input including the specification of new terminals. Non-terminal symbols in grammar rules are displayed as choice points in the natural language sentence being developed in the user interface. If only a single non-terminal symbol occurs in the rule, then the rule automatically expands to display based on the rules associated with that non-terminal. Multi-level menus may be generated if a rule contains multiple unambiguous non-terminal options and/or terminal symbols. This allows broad category such as 'Type of Operation' to be broken into submenus for 'offensive', 'defensive', 'enabling', and 'operations other than war' can help guide the user without requiring extra component drilldown operation. If a rule includes a nil expansion option, then it is displayed as optional input in the user interface. Required (non-optional) non-terminals symbols are initially displayed with an in-lined rule name (<rule_name>) to indicate that required text needs to be filled in. This rule name is replaced by the user's input once the required information is provided.

In addition to the interpretation of the DCG into user interface components, the TBNLUI also supports several user interface options including component wrap-around and automatic component focus. Component wrap-around allows the system to use multiple lines to display text input components. This improves the readability of the text being developed by allowing the user to view more of the text at one time. Automatic component focus is an option that allows the system to automatically refocus the display to the component currently under consideration by the user. When the user selects a new input component, the system can expand this component to allow further specification by the user while re-collapsing other components that the user is no longer working with. This helps focus the user's attention to the details of the current text being developed.

3.2.3 Single and multiple sentence and phrase options

As described in the Software User's Manual (ALPHATECH TBNL Applications UM, 2002), the TBNLUI provides both a single sentence/single phrase environment option and a multiple sentence/multiple phrase environment option. The single sentence/single phrase environment is useful for integrating the template-based NL application with other applications that need structured text input for a specific field. The NL environment can be initiated each time the user wants to edit a specific field in the third party application. An example use of this environment is the Task Editor, which was integrated with the Temporal Plan Editor. This example is described in more detail below. The multiple sentence/multiple phrase environment is useful for developing an application for editing documents like a COA statement. A tabbed environment allows the user to switch between topics, such as fire support and close battle statement, and allows multiple sentences to be developed for each topic, for example, a task statement for each unit in the close battle statement. An example use of this environment is the COA Statement Editor, which is described in more detail below.

3.2.4 Standalone and client/server configuration options

The TBNLUI is designed to work both as a standalone application and in a client/server that allows it to be integrated with other applications. The server-based configuration of the TBNLUI works with and is called by other applications as a way for users to enter structured text that will be utilized by the controlling application. The user interface window is created when needed by the controlling application, which initializes the TBNLUI with the desired grammar and sentence to be modified. Configuration of the TBNLUI to operate in either standalone or client/server mode is described in the Software Developer's Guide (ALPHATECH TBNL Applications SDG, 2002). Example use of the standalone configuration is demonstrated in the COA Statement Editor and example use of the client/server configuration is demonstrated in the Task Editor, both described below.

3.2.5 TPE enhancements

To demonstrate the integration of the TBNLUI as a client/server application, we modified BBN's Temporal Plan Editor (TPE) to include tactical tasks associated with a fixed location over time on the synchronization matrix. As shown in Figure 5, we used standard military symbology to represent the tactical tasks. A dialog box (Figure 6) associated with the tactical task allows the user to change the type of task or its start and end times. A plan representation for tactical tasks was also created to support the integration of the TBNLUI. This representation is stored in a *plan*

server that mediates the exchange of information amongst different input components. For example, the TPE can send information about the type of a task or its location to the plan server, and this information is then available for retrieval by the natural language interface. Changes made to the plan in the TPE are posted to the plan server. The natural language interface can then update its display of the information when the description is edited.
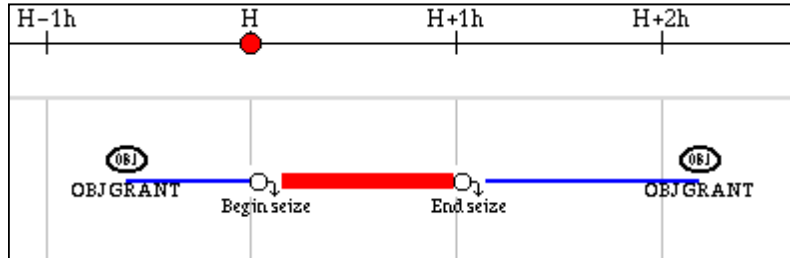
Figure 5: Tactical Tasks shown in the TPE.

Figure 6: A dialog box showing details about the tactical task

3.2.6 Task Editor

In order to demonstrate the integration of the TBNLUI into a larger application, the Task Editor (Figure 7) was created and integrated into the TPE. The Task Editor allows the user to describe the tactical task statement, including the type of operation, purpose of the task, and temporal information, based on doctrinal requirements. This information is displayed as a sentence in the description field of the tactical task dialog box. When the user chooses to edit the sentence, the TPE invokes the Task Editor so that the sentence can be modified.

16
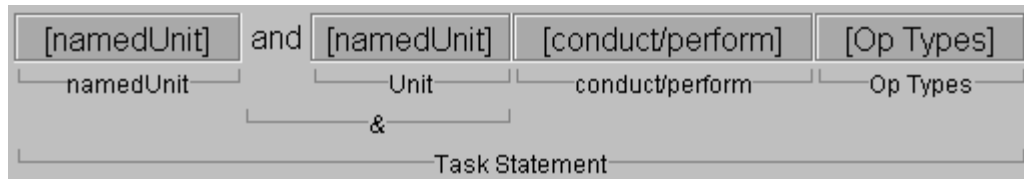
Figure 7: Example Task Editor

3.2.7 COA Statement Editor

To demonstrate the standalone configuration of the TBNLUI as well as the TBNLUI's multiple sentence/multiple phrase capability, we implemented a prototype COA Statement Editor. The COA Statement Editor provides planners with a text-based interface for specifying mission plans and complements other planning environments such as the Temporal Plan Editor. The initial prototype of the COA Statement Editor was integrated with a COA sketching tool (nuSketch) as part of a separate project (ICCES CEP) during which it was used by end-users to develop COA statements. This integration demonstrates the potential benefits of linking a text-based planning tool with a graphical planning tool. In addition, use of the COA Statement Editor in the ICCES CEP provided us with valuable end-user feedback on our tool. The interface used for this initial end-user test was based on a template drilldown paradigm as demonstrated in Figure 8. The user would choose a non-terminal to expand. Its expansion options were then displayed as several options presented below the previous sentence expansion. As the user chose further drilldown options, unselected options were hidden and the new drilldown option presented below the selected rule. This interface was found to be both cumbersome and confusing to the end-users. As a result, we modified the user interface to use the in-line sentence expansion approach with drilldown menus described in an earlier section. This resulted in the creation of a stand alone COA Editor as shown in Figure 9.
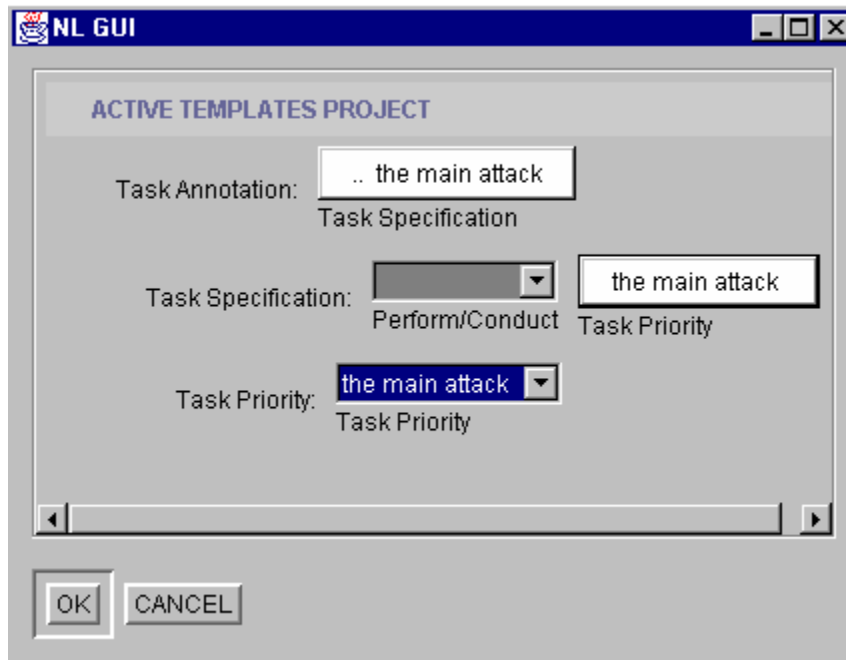
Figure 8: First statement editor layout
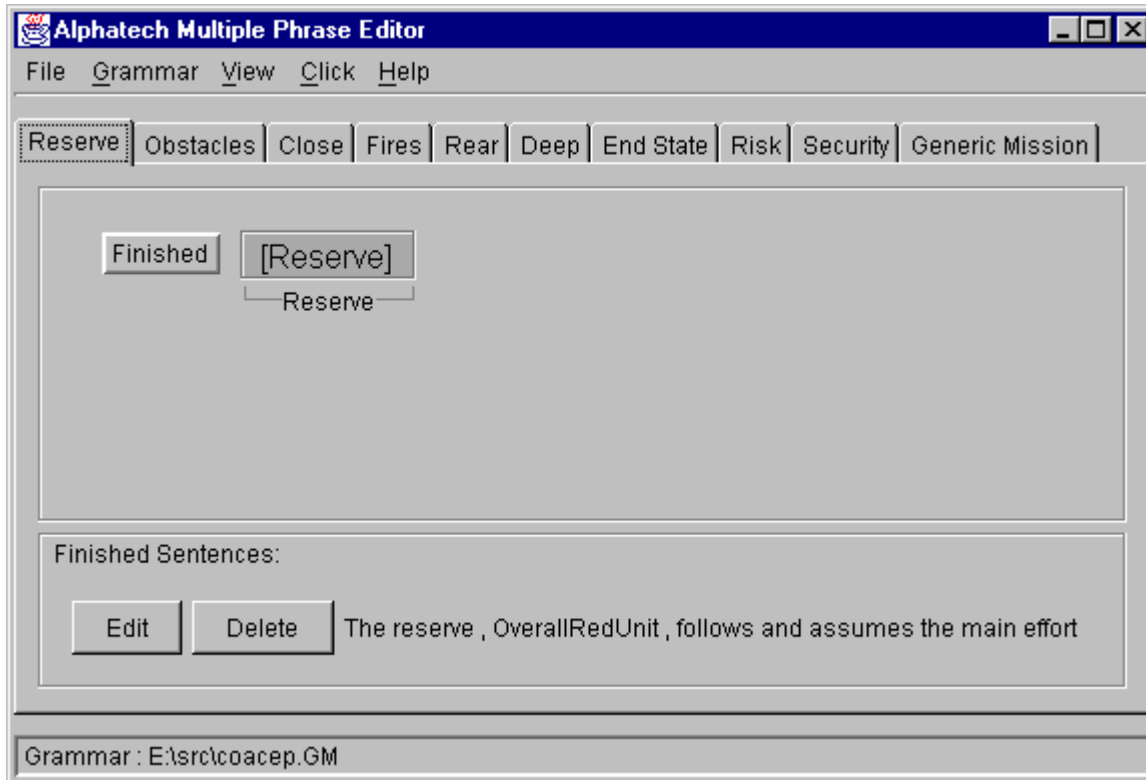
Figure 9: Example COA Statement Editor showing statement about reserves.

3.3 PLAN CRITIQUER AND LOOK-AHEAD CONSTRAINT REASONER

In our Plan Critiquer and Look-Ahead Constraint Reasoner design, we envision mission-planning staff using enhanced versions of the Temporal Plan Editor (TPE) and other graphical planning tools to develop branch plans for a mission. An enhanced graphical planning tool combined with a branch plan environment would allow the planning staff to specify decision points and indicate what effect a decision has on the plan. It would critique the various branch plans for feasibility. It would then allow the mission planning staff to view the effects of a combination of decisions by showing the resulting merged branch plans. Finally, it would support the analysis of what-if circumstances by allowing the planning staff to define a situation that the system then uses to determine if a feasible plan exists.

Figure 10 shows the concept of operations for the branch planning environment. The mission plan would be active in the plan server/critiquer. The planning staff would use the TPE to define individual branch plans. The planning staff could then use the Decision Point Editor (DPE) to define a set of branch plans to be merged and displayed in the TPE. Likewise, the planning staff could use the Situation Editor (SE) to define a combat situation for which a set of viable branch plans are to be merged and displayed in the TPE. A plan critiquer interacts with the plan server and notifies the planning staff of possible problems with the active plan.

Decision Point Information

General	Constraints	Relations	Branches

ID: 3   Name: Go-no-Go

	ID	Name	Conditions	
Edit	12	Primary	Helo support is available and fire support is available and either weather is good or weather is fair	▲

Decision Point Editor

	ID	Textname:	Description:	Branch Plan:	
	3	Go-no-Go	Check pre-mission requirements before issuing go-no-go command	Primary	▼
	4	Drop Primary DZ	Verify that air defense at airport are destroyed before commencing parachute drop at airport	Alternate	▼
	5	Begin airland ops	Assess runway conditions before commencing airland ops	Primary	▼

View   Reset   Close

Decision Point Editor

Temporal Plan Editor

Router

Plan Server / Critiquer

Situation Editor

Contingency:	ID	Value:	Dependency:	
Weather	1	Good		▲
	2	Fair		
	3	Poor		
	4	Bad		
Helicopter Support	5	Available		
	6	Unavailable	Weather is bad	
Enemy Force Size at Airfield	7	Motorized infantry platoon		
	8	Company		
	9	> Company		
Number of Hostages	10	< 200		
	11	200-300		
	12	300-400		
	13	> 400		

View   New   Delete   Can

Situation Editor

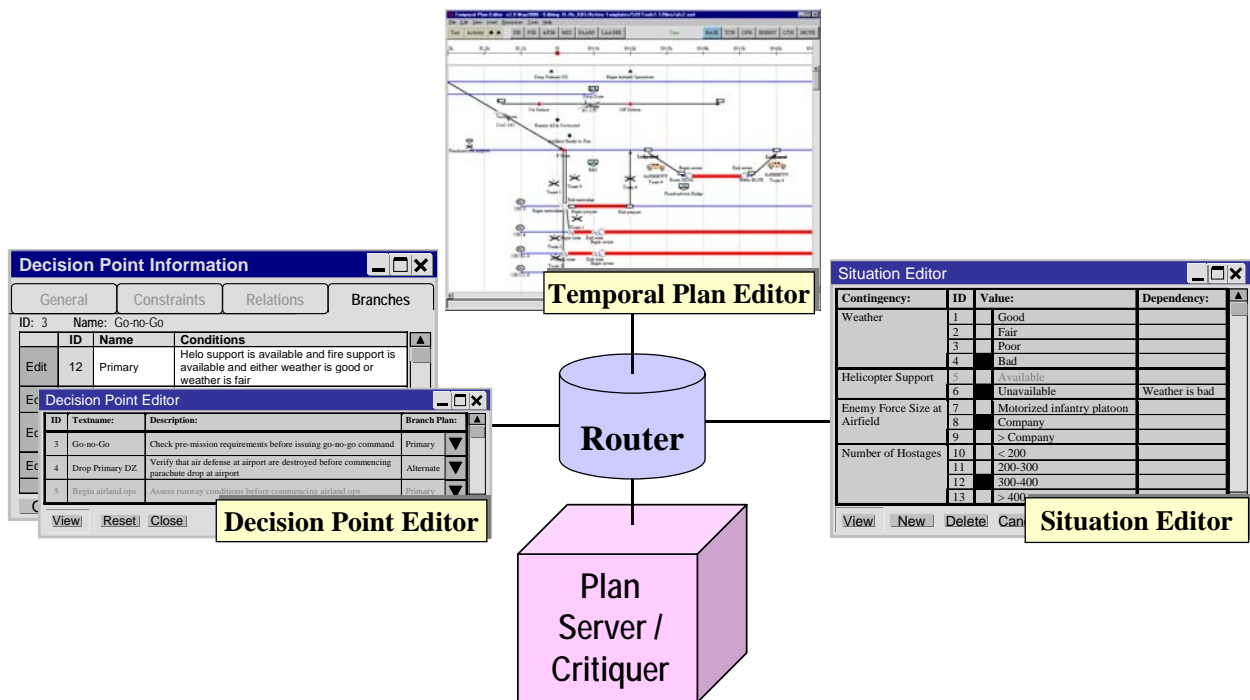**Figure 10: CONOPs for editing, viewing, and critiquing branch plans.**

Figure 11 shows a temporal plan for an Airfield Seizure (AFS) adapted from Active Templates kickoff meeting material. This temporal plan uses the capabilities of the Temporal Plan Editor (TPE) version 1.3.1 with our enhancements for displaying tactical tasks as described in 0. The TPE graphically captures temporal constraints such as the fact that TEAM D's task of clearing and securing Route Blue follows the preparing of the airfield for airland operations. Unfortunately, version 1.3.1 of the TPE does not contain a mechanism for checking the feasibility of a plan. In version 1.2d of the Synchronization Matrix Editor (Temporal Plan Editor), a simple constraint check is performed between an asset's speed and distance capabilities and the planned movement of that asset. These simple constraints can be useful in preventing the planning staff from inadvertently attempting to overextend an asset's capabilities. Section 0 describes how to expand the idea of using constraints to critique the plan and alert the planning staff of potential problems that they may want to further address.

Notice that the temporal plan displayed in Figure 11 describes only one possible sequence of events; however, situations may arise during the execution of the mission that require a different course of action or branch plan to be implemented in order to achieve the objective of the mission. Decision points indicate points at which a command decision must be made and a branch plan commenced if necessary. In the TPE, a decision point may be specified to occur at a specific time point and may be visually associated with an event point, start of an asset's movement, or end of an asset's movement.
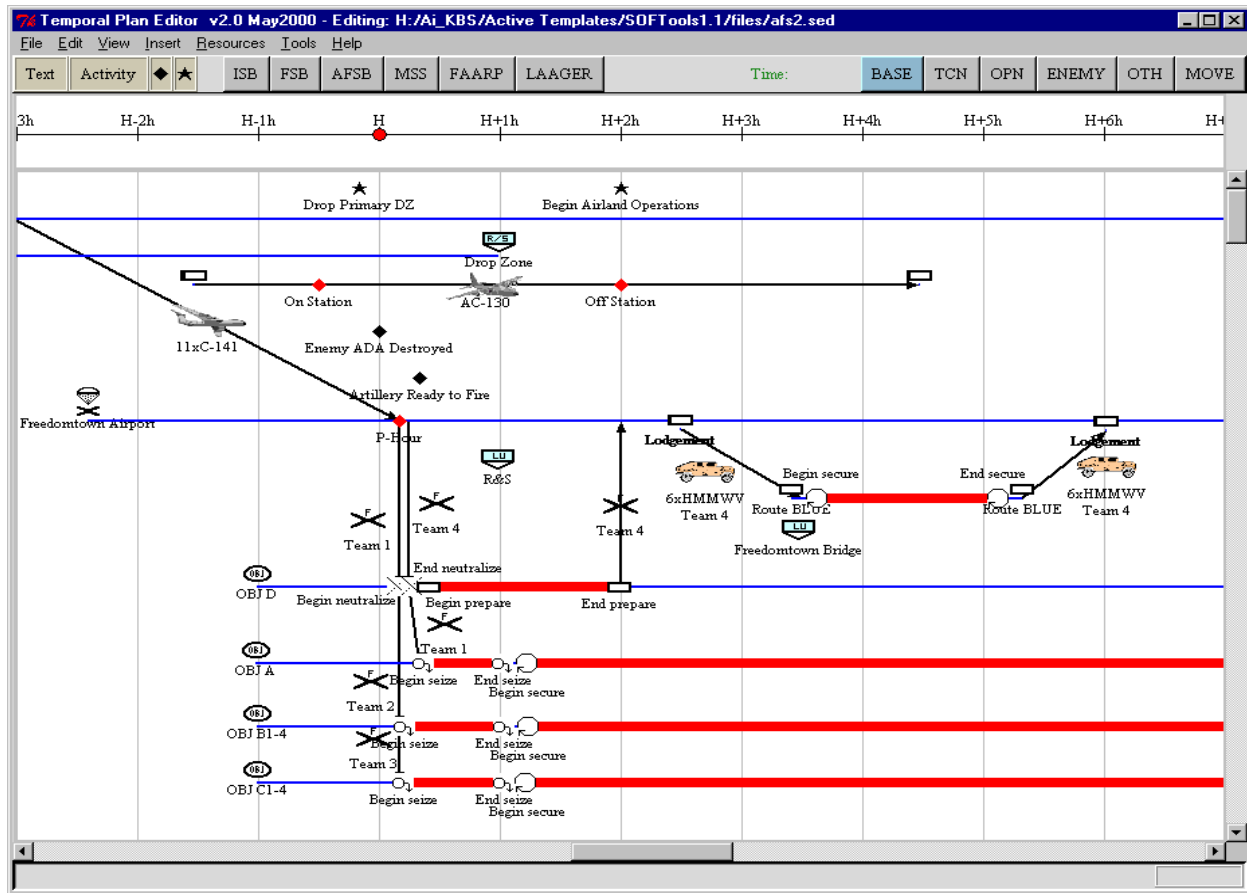
Figure 11. Temporal plan for an airfield seizure.

Although a decision point indicator may be included in a temporal plan, there is currently no mechanism in version 1.3.1 of the TPE for displaying branch plans resulting from a decision. Version 1.2d of the Synchronization Matrix Editor (Temporal Plan Editor) supports rudimentary branch plan visualization by allowing the user to switch to a new temporal plan at each decision point; however, the system only supports one independent alternate branch plan per decision point and does not provide a mechanism for directly viewing how the decision has affected the original plan. Section 0 describes how the Temporal Plan Editor could be extended to more fully support branch plan development and visualization. Sections 0 and 0 then describe tools for interacting with multiple decisions simultaneously.

3.3.1 A Plan Critiquer

Planning staff bring a wealth of experience and knowledge with them when developing a mission plan. This experience and knowledge helps them to rapidly develop plans that take advantage of the capabilities of the various assets that are available to them. Experience provides them with knowledge such as how many troops an MH-60 can transport, how much time a particular task will take to complete, and the level of force required to successfully complete a task. While an experienced planner can use this knowledge to develop successful mission plans, interactions between plan elements and checking the plan for viability and correctness may slow the planning process. Our design uses asset, time, and location information available in the Temporal Plan Editor (TPE) to alert the planning staff of potential problems in the temporal plan in

20

regards to feasibility, acceptability, completeness, and correctness. Based on information that is currently available from the TPE, possible user initiated critiques include:

1. *Does each asset have sufficient time to perform its maneuvers?*
2. *Does each asset have sufficient range to perform its maneuvers?*
3. *Does each unit have sufficient assets to perform its maneuvers?*
4. *Is information missing from the plan?*
5. *Is any unit responsible for a disproportionately large number of tasks?*
6. *Is each unit assigned a task?*
7. *Does each unit have a method of ingress?*
8. *Does each unit have a method of egress?*
9. *Has fire support been specified?*
10. *Have security forces been specified?*
11. *Have reserve forces been specified?*
12. *Are the units designated as the reserve uncommitted?*
13. *Does each unit arrive at its destination before its assigned task starts?*
14. *Is each task completed before the unit executing the task departs the location?*

When the planning staff has completed a temporal plan, they would initiate the plan critiquer through a menu option in the TPE. This would cause the information in the TPE to be passed to the plan critiquer, which would critique the plan based on a set of general constraints designed to test various aspects of the plan. The results of the critique would be reported to the planning staff in a separate window. For example, if no fire support has been specified under the 'Fire Support' separator in the TPE, then the plan critiquer would report a warning to the planning staff as shown in Figure 12.
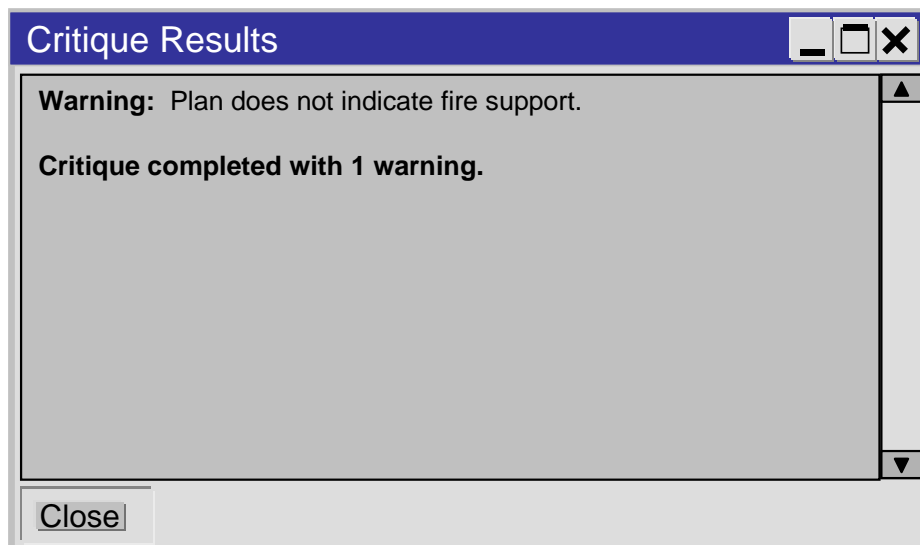
Figure 12: Plan critiquer results window with one warning message.

3.3.2 An Enhanced Temporal Plan Editor for Branch Plan Visualization

This section describes our concept of an enhanced Temporal Plan Editor for branch plan visualization. Figure 13 shows the primary temporal close battle plan for the airfield seizure scenario described earlier. At the top of the temporal plan in this figure are enhanced decision points. Each decision point has multiple branch options, which are indicated by the digits below the decision point. A highlighted digit indicates the branch that is currently being displayed for each decision point. In this case, the first (primary) branch plan is being displayed for each decision point, indicating that the plan being displayed is the base plan. The sequence of branch plans that are displayed can be changed by manipulating the branch plan selected at each decision point; by using the Decision Point Editor described in Section 0 to change multiple decisions at one time; or by specifying a situation using the Situation Editor described in Section 0.
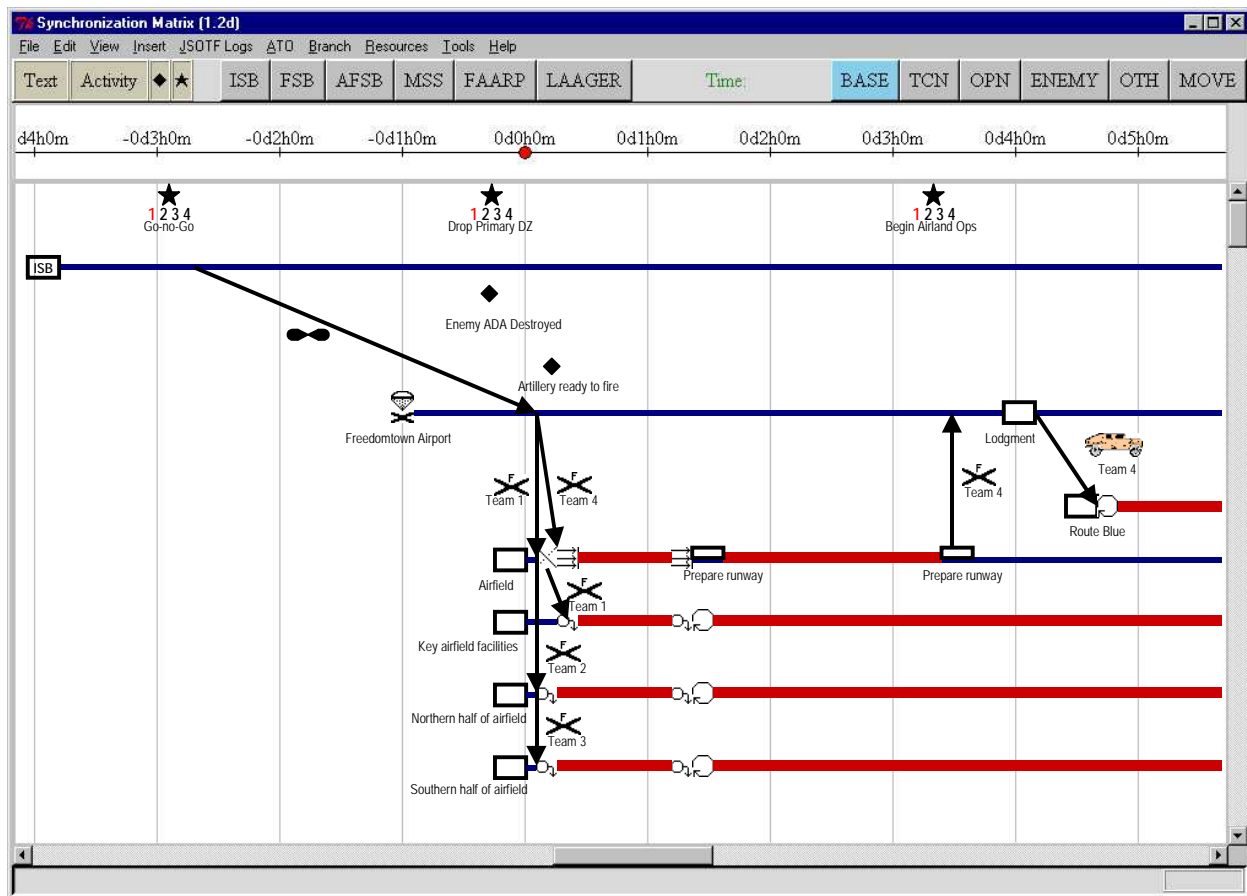
Figure 13: Primary plan for an airfield seizure.

Figure 14 shows the temporal plan when an alternate branch plan is displayed for the 'Drop Primary Drop Zone (DZ)' decision point, on the assumption that additional enemy air defense assets are detected at the primary DZ prior to P-hour. The alternate plan consists of having the troop air transports orbiting near the airport while additional fire support is used to destroy the enemy's air defenses. An additional decision point on whether to drop at the primary DZ is added after the additional fire support has been used. This branch plan results in a 15-minute delay in subsequent actions. Decision points at which a change in branch plans has occurred are highlighted. In Figure 14, the red decision point for 'Drop Primary DZ' indicates that the displayed

plan has changed as a result of adopting a new branch plan. In this case, the red 2 indicates that alternate branch plan 2 is being displayed, and the shaded 1 indicates that the primary plan was previously being displayed.

Figure 14: An alternate branch plan for an airfield seizure.

Options in the Show submenu of the Temporal Plan Editor's View menu control how changes to the branch plan sequence affect the current plan. All new plan elements or plan elements that are affected by the recent change can be highlighted (shown as purple in the examples). All plan elements that no longer exist in the new branch plan sequence can be overlaid and highlighted (shown in gray in the examples). In Figure 14, the new decision point, and the new ingress flight path are completely highlighted. In addition, elements that are delayed have received a highlighted time update. If start and end times had been displayed (another option in the View menu), then these times would have been updated and highlighted to reflect the time delay. Figure 15 shows the temporal plan for when a different alternative branch plan is used for the 'Drop Primary DZ' decision point with the previous branch plan sequence overlay displayed in gray. In the new branch plan sequence, the operation is relocated to an alternate site, which causes many of the previous plan elements to be eliminated and replaced with new plan elements.
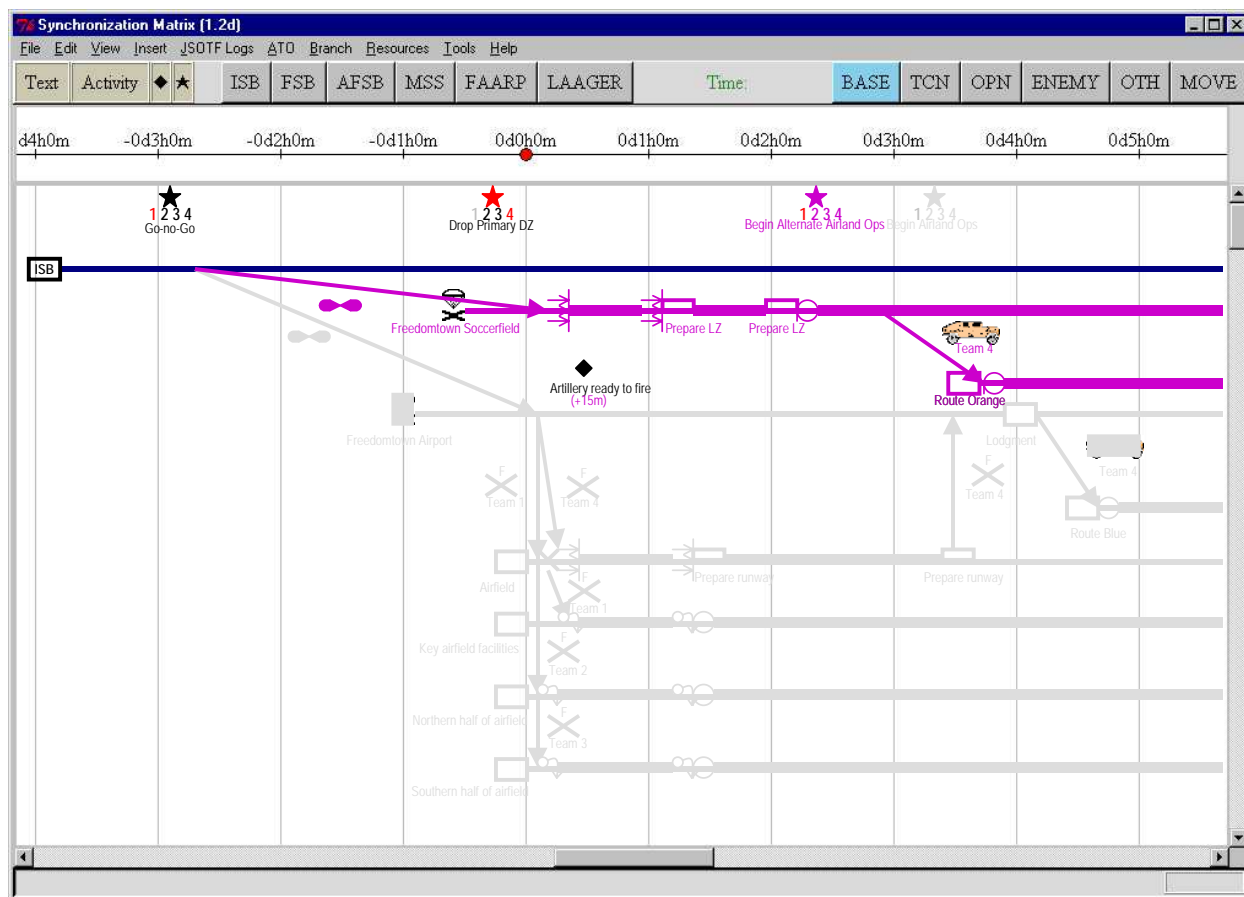
Synchronization Matrix (1.2d)

File Edit View Insert JSOTF Logs ATO Branch Resources Tools Help

Text   Activity ◆ ★   ISB FSB AFSB MSS FAARP LAAGER   Time: BASE TCN OPN ENEMY OTH MOVE

d4h0m   -0d3h0m   -0d2h0m   -0d1h0m   0d0h0m   0d1h0m   0d2h0m   0d3h0m   0d4h0m   0d5h0m

1 2 3 4
Go-no-Go

Drop Primary DZ

Begin Alternate Airland Ops Begin Airland Ops

ISB

Freedomtown Soccerfield

Prepare LZ   Prepare LZ

Team 4

Artillery ready to fire
(+15m)

Route Orange

Freedomtown Airport

Lodgment

Team 4

F
Team 1   F
Team 4

F
Team 4

Route Blue

Airfield

Prepare runway

Prepare runway

F
Team 1

Key airfield facilities

F
Team 2

Northern half of airfield

F
Team 3

Southern half of airfield

Figure 15: An alternate branch plan with an overlay of the previous branch plan.

3.3.3 A Decision Point Editor

The Decision Point Editor (DPE) allows the planning staff to view and edit details associated with the decision points in a plan. Figure 16 shows the main window of the Decision Point Editor. The DPE lists the name, description, and associated branch plan for each decision point. Each branch plan currently selected for display in the TPE is indicated and can be changed by the user. For each decision point, the user may activate a menu that provides commands specific to the decision point selected. The user can choose to edit the details of the decision point, add the decision point to the plan in the TPE, remove the decision point from the TP, create a new decision point in the DPE, or delete the decision point from the DPE. The DPE also allows the user to specify a sequence of branch plans by selecting a specific branch plan for each decision point or by allowing the system to generate the branch plan sequence. The resulting branch plan sequence can then be critiqued using the plan critiquer discussed in Section 0 or viewed using the TPE as discussed in Section 0.
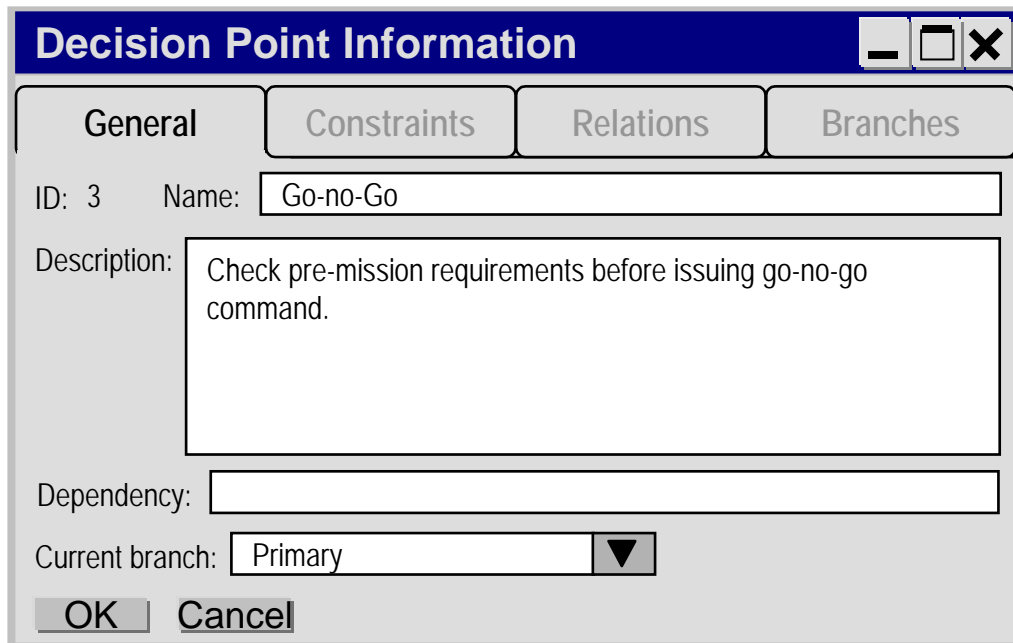
Figure 16: Decision Point Editor for an airfield seizure.

If the user chooses to edit a decision point, the system presents the Decision Point Information (DPI) dialog box for that decision point. Figure 17 show the four tabs of information available for a decision point.

- Figure 17 shows the general information collected about a decision point. It includes editable fields for the name of the decision point, a description of the decision made at the decision point, dependencies restricting when the decision point is valid, and the branch plan currently selected for display in the TPE in the DPI.

- Figure 18 shows the temporal constraints on the decision point interval including the start time, end time, and absolute or minimum and maximum duration. It would be possible to specify that a decision point interval can start or end at a specific time or as soon or late as possible. In addition, it would be possible to specify specific start and end times with respect to H-hour or to provide relative times such as sunrise and moonrise.

- Figure 19 shows the relation information for the decision point. The Relation tab allows the planning staff to indicate dependencies between the decision point and various plan elements such as tasks and maneuvers. The planning staff can list plan elements that precede or follow the current decision point; the type of interval relationship between the plan element and the decision point; and the amount of time lag that occurs between the related end points. When the planning staff creates a plan in the TPE, the system interprets the plan in order to pre-populate the relation information. The planning staff can then modify this information as necessary in the Relation tab. A view option in the TPE would allow the system to display relation information via connecting lines. In addition, an enhanced version of the TPE could be used to pre-specify the relation task and type information. The basic interval relations that can be used to describe the relationships between decision points and plan elements are graphically shown in Figure 21. There are six basic relations (precedes, meets, overlaps, contains, starts, and finishes), their inverses (follows, met-by, overlapped-by, during, started-by, and finished-by), and the equivalency relation.

- Figure 20 shows the branch plan information for the decision point. Each branch is assigned a priority, name, set of defining conditions, and a base plan for modification. The conditions are based on the contingencies defined using the Situation Editor described in Section 0. These conditions are used to critique the plan and to guide the user in the selection of branch plans that are valid given the current situation. In addition to specifying contingencies for branch

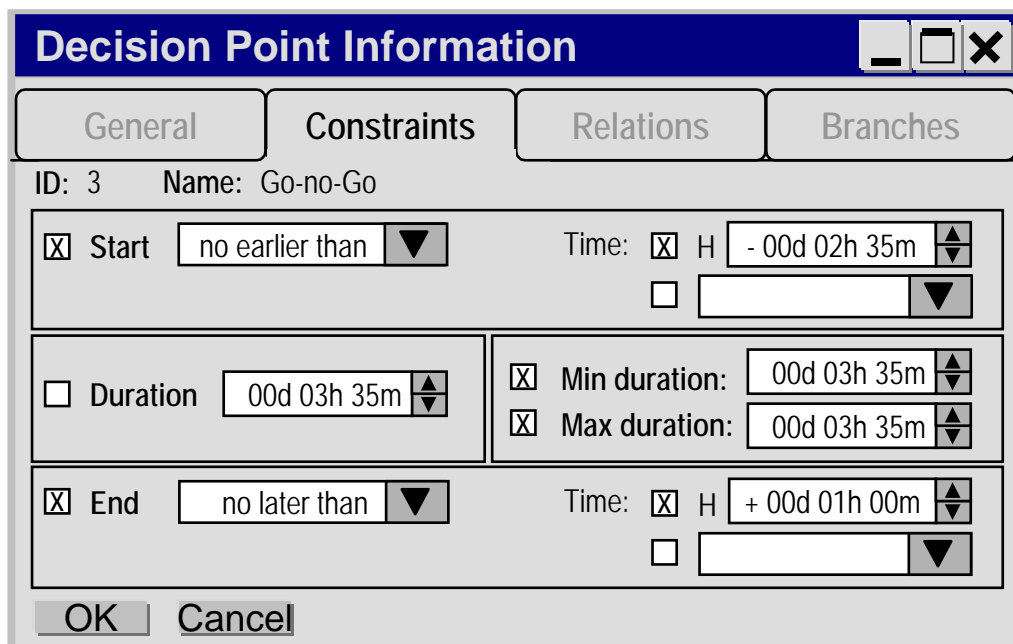plans, each branch plan can be edited using the TPE. Initially, the base plan specified by the 'Modification of' field is presented in the TPE. As the user modifies the base plan to create the branch plan, the system highlights the changes as described in Section 0.

Figure 17: General information in the Decision Point Information dialog box.

Figure 18: Constraint information in the Decision Point Information dialog box.

Decision Point Information

	General	Constraints	**Relations**	Branches

ID: 3   Name: Go-no-Go

	ID	Element Description	Relation	Lag	▲
	4	Move Team 1, Team2, Team 3, and Team 4 on Fixed Wing 1 from ISB to Freedomtown Airport	DP A ▼ follows DP	00h15m00s	
					▼

OK   Cancel

Figure 19: Relation information in the Decision Point Information dialog box.

Decision Point Information

	General	Constraints	Relations	**Branches**

ID: 3   Name: Go-no-Go

	ID	Name	Conditions	Modification of	▲
Edit	12	Primary	Helo support is available and fire support is available and either weather is good or weather is fair	Base	
Edit	13	Alternative	Helo support is available and fire support is available and weather is poor	Base	
Edit	14	Contingency	Helo support is unavailable and fire support is available and either weather is good or weather is fair	Base	
Edit	15	Emergency	Either helo support unavailable and fire support is unavailable or weather is bad	Base	▼

OK   Cancel

Figure 20: Branch information in the Decision Point Information dialog box.
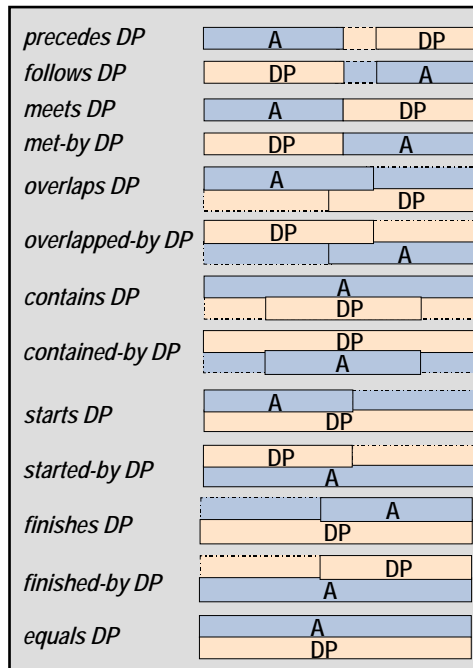
Figure 21: Graphical depiction of the interval constraints

3.3.4 A Situation Editor

The Situation Editor (SE) provides an alternate method for accessing branch plan sequences. In Section 0, the Condition field specified a set of conditions under which a particular branch plan is recommended. There may be multiple components to the condition for a branch plan. For example, a decision to use an alternate DZ may depend on weather conditions and the strength of the enemy force at the primary and alternate DZs. These contingencies may also affect several different branches or decision points. For example, the weather condition may also be used in deciding which mode of transportation to use from the DZ to the primary objective. The Situation Editor allows the planning staff to specify a situation and then ask the system to find a viable branch plan sequence (if one exists) for that situation. The user can choose to edit the details of a contingency, create a new contingency, delete a contingency, add or delete possible values for a contingency, or specify dependencies for a contingency value. The SE can also generate a situation from the defined contingencies, which the system can then use to locate a viable branch plan sequence.

Figure 22 shows an example of the Situation Editor being used to specify a situation that includes information about the weather, helicopter support availability, the enemy force size at the airfield, and the number of hostages. The values for each contingency are mutually exclusive, e.g., helicopter support can be either available or unavailable but not both. In addition, some values may be dependent upon or implied by other states. For example, helicopter support is unavailable according to the Situation Editor shown in Figure 22 if the weather is bad. These dependencies will be used to help guide the user in selecting a situation by shading values that are not valid based on previous selections and automatically selecting values that are implied by previous selections.

It is important to note that much of the information used by the Situation Editor is not available in the TPE other than as part of the DPE. Thus, in order for the Situation Editor to search for valid

branch plan sequences, it is necessary to formalize the condition information in the DPE and provide the planning staff with a mechanism for specifying this information.

Situation Editor				
Contingency:	**ID**	**Value:**		**Dependency:**
Weather	1		Good	
	2		Fair	
	3		Poor	
	4	■	Bad	
Helicopter Support	5		Available	
	6	■	Unavailable	Weather is bad
Enemy Force Size at Airfield	7		Motorized infantry platoon	
	8	■	Company	
	9		> Company	
Number of Hostages	10		< 200	
	11		200-300	
	12	■	300-400	
	13		> 400	

View   New   Delete   Cancel   Choose Random Situation

Figure 22: Situation Editor for an airfield seizure.

3.3.5 A Look-ahead Constraint Reasoner

As more and more branch plans and sequels are developed for a mission, it can become increasingly difficult for the planning staff to reason about the many possible interactions between branches. As a result, comparatively few branch plans and sequels are constructed into a plan. Furthermore, during plan execution, it can be difficult to accurately track all of the implications of selecting a particular branch plan in terms of downstream asset requirements, schedules, and other branch plans and sequels. To address these problems, we describe a look-ahead constraint reasoner designed to support more sophisticated plan feasibility critiques than could be handled by the plan critiquer described in Section 0. The critiquer identifies potential asset shortfalls and schedule violations associated with different combinations of branch plans and sequels.

In order to support look-ahead constraint reasoning, it is necessary to further extend the plan representation for tasks and maneuvers in the TPE. Figure 23-Figure 26 demonstrate our extensions. These four figures show examples of the four tabs of information associated with an individual task.
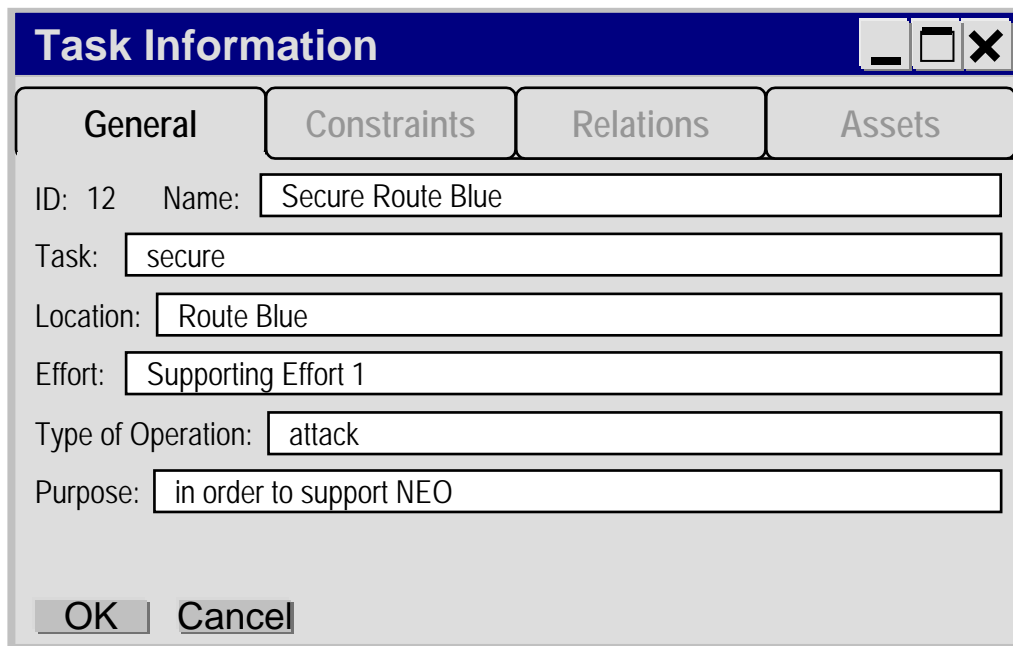
- Figure 23 shows the general information collected about a task. It includes editable text fields for the name of the task, the type of task, the location at which the task is performed, which effort (Main Effort, Supporting Effort) the task represents in the overall mission, the type of operation being performed, and the purpose.

- Figure 24 shows the temporal constraints on the task's interval including the task's start time, end time, and duration. These are the same types of temporal constraints that are available for decision points as discussed in Section 0.

- Figure 25 shows the relation information for the task. Like the Relation tab for decision points discussed in Section 0, this tab allows the planning staff to indicate dependencies between various tasks and other plan elements. The planning staff can list plan elements that precede or follow the current task; the type of interval relationship between these two items such as 'a task ends sometime before the current task starts', 'a task immediately precedes the current task', or 'both tasks start at the same time'; and the amount of time lag that occurs between the related end points. As in the case for Decision Points, the user could use the TPE to pre-specify task relations and type information and to graphically view the dependencies.

- Figure 26 shows the list of assets associated with the task. It includes information about the Asset Description, Op Element, and the Assets used.

For each branch plan sequence, the system would be able to test critiques such as:

1. *Is the Main Effort identified?*
2. *Does each unit have sufficient time to accomplish its mission?*
3. *Does each asset have sufficient fuel to complete all of its assigned tasks?*

This additional information will also help with branch plan visualization by providing information in regards to how changes caused by one branch plan affect other parts of the plan. Finally, collection of this additional task information will support look-ahead constraint reasoning by providing the necessary information to determine if a change caused by a branch plan affects the ability of a unit to complete its assigned tasks.

Figure 23: General information in the Task Information dialog box.

Task Information ▬ ❐ ✕

| General | Constraints | Relations | Branches |

ID: 12   Name: Secure Route Blue

☒ Start [at ▼]   Time: ☒ H [+ 00d 01h 15m ⬍]
☐ [▼]

☐ Duration [00d 01h 30m ⬍]   ☒ Min duration: [00d 01h 00m ⬍]
☒ Max duration: [00d 05h 00m ⬍]

☒ End [no later than ▼]   Time: ☐ H [+ 00d 00h 00m ⬍]
☒ [sunrise ▼]

[OK] Cancel

Figure 24: Constraint information in the Task Information dialog box.

Task Information ▬ ❐ ✕

| General | Constraints | Relations | Assets |

ID: 12   Name: Secure Route Blue

	ID	Element Description	Relation	Lag	▲
	5	prepare runway	A ... Task precedes task ▼		
					▼

[OK] Cancel

Figure 25: Relations information in the Task Information dialog box.

Task Information `_` `□` `✕`

General	Constraints	Relations	Assets

ID: 12   Name: Secure Route Blue

	Asset Description	Op Element	Assets	▲
	Team 4	TF Charlie	6XHMMWV	
				▼

  OK | Cancel

Figure 26: Asset information in the Task Information dialog box.

3.3.6 Discussion

The branch plan representation, visualization, and critiquing environment described above provides several key benefits including:

1. *Parametric constraints.* Even absent any branch plan representation, parametric constraints help users develop doctrinally correct plans while avoiding inconsistencies. The plan critiquer checks the plan and alerts the user of potential problems related to the use of assets, time, locations, tasks, and maneuvers when developing a plan in the TPE.

2. *Decision point development.* During plan development and war-gaming, points are identified in the plan where decisions as to which of several alternate branch plans will be executed. The points at which important decisions are made during war-gaming are noted in the plan and branch plans are developed for different contingencies. The Decision Point Editor can help the planning staff capture conditions of the decision point and associate branch plans for different contingencies with that point.

3. *Branch plan visualization.* Although branch plans can be developed separately from the main mission plan, it can be difficult to rapidly identify what the differences are between various branches. Sometimes the differences between branch plans are quite substantial, while at other times the differences are minor. In both cases, it is beneficial to be able to see what has changed between branch plan alternatives. This can help point out the effect a decision will have on the plan as well as identify those units that are affected by the decision. By wrapping the branch plan visualization in with the main plan, the differences between branch plans can be highlighted.

4. *'What-if' scenario testing.* Branch plans are developed to handle specific contingencies that have been identified for a mission plan. During subsequent plan development, briefing, and

war-gaming, it is useful to know what the plan would look like if certain conditions were to hold during mission execution. For example, someone may ask what would happen if helicopter support were lost and twice as many hostages needed to be extracted from the primary objective. Using the Situation Editor in conjunction with formalized conditions associated with the decision points, the planning staff can query the system for whether a viable plan has been developed for this contingency and then have the system display a possible course of action for this situation. The system could also use this information to suggest areas in the plan that may need further development.

5. *Temporal constraint reasoning.* In addition to verifying that various contingencies have been planned for, it is also important that branch plans do not cause violations in task dependencies and timing constraints. For example, if a branch plan calls for using an alternate DZ that leads to a longer infiltration time, it is important to know that the delay to time on target does not jeopardize the completion of the primary mission. If the plan requires two hours to complete the primary task, and this task must be completed before a specific time, the delay in reaching the target may make it impossible to complete the task. The temporal constraints defined for each task can be used to identify potential conflicts before they become real problems.

3.3.7 Plan Critiquer Prototype

As our first step in developing a plan critiquer and look-ahead constraint reasoner, we developed an XML message router in order to connect the Temporal Plan Editor to the Decision Point Editor, Situation Editor, and Plan Server/Critiquer (Figure 10). We then modified the Temporal Plan Editor to interface with the XML message router and developed a basic Plan Server/Critiquer with a plan representation sufficient for conducting temporal constraint reasoning on tasks and movements defined within the Temporal Plan Editor.

Message Router

To handle communications between the various user interfaces and the Prolog knowledge engine, we developed an XML message router. Upon connection to the message router, the various clients register their capabilities. When the router receives a request for services from a client, it forwards this request to all clients who have registered a capability for handling this request. For example, if a client makes a request to display a value, the message router passes this information to all clients capable of displaying that type of value.

Temporal Constraint Editor

To support development of the Temporal Constraint Reasoner, we created the Temporal Constraint Editor to support the generation of user-specified temporal constraints between two events (i.e., task and movement end points) and between an event and H-Hour. Figure 27 shows an example of the constraint editor as part of the Event Timings listing. Each constraint that has been created is listed and can be modified using the constraint editor in the lower portion of the window. The temporal constraints are specified by selecting two events and specifying the type of temporal constraint that exists between the two items. The user can specify that one item must occur before or up to the occurrence of the second item, that the two items must occur at the same time, or that there is a constant difference between the occurrences of the two items. The resulting constraints are then displayed in the TPE as shown in Figure 28.
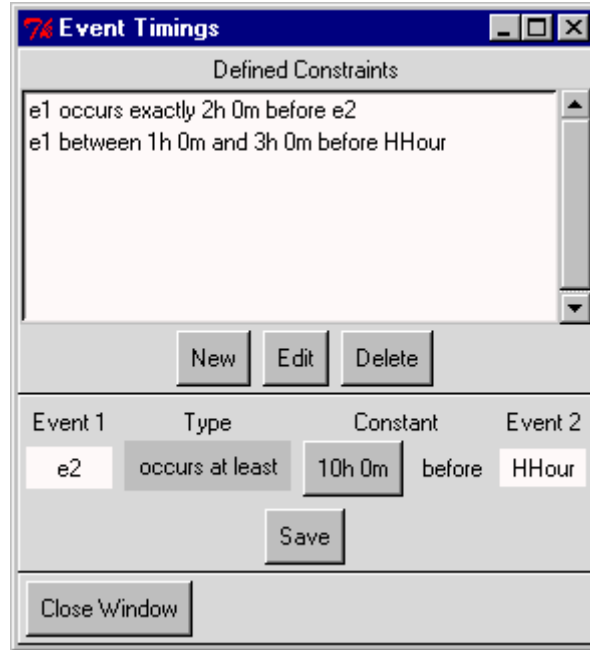
Figure 27: Temporal Constraint Editor with example constraints.

Temporal Constraint Reasoner

The point-based, single-interval constraints specified in the Temporal Constraint Editor can be represented mathematically as $e2 - e1 \in [a, b]$ and were used to implement a variety of qualitative and quantitative temporal relations as follows:

e1 occurs at the same time as *e2* :	$e2 - e1 \in [0, 0]$
e1 occurs before *e2* :	$e2 - e1 \in (0, \infty)$
e1 occurs at least *T* before *e2* :	$e2 - e1 \in [T, \infty)$
e1 occurs exactly *T* before *e2* :	$e2 - e1 \in [T, T]$
e1 occurs NET *T* before *e2* :	$e2 - e1 \in (-\infty, T]$
e1 occurs between *T1* and *T2* before *e2* :	$e2 - e1 \in [T1, T2]$

These constraints are used by a path consistency algorithm whenever a user adds, deletes, or modifies a constraint between events to check for network consistency and compute the minimal constraint network and domain for each event. Whenever the user attempts to temporally reposition an event, these constraints are used to notify and constrain the user to the limits of this repositioning given the current constraints.

Temporal Plan Editor Enhancements

To complete our initial prototype, we enhanced the Temporal Plan Editor (TPE) to work with the Temporal Constraint Reasoner. This integration allows the user to assign constraints between events including the start and end points of movements, the start and end points of tasks, planned events, and decision points. The user can specify that one event must occur sometime before another event, at the same time as another event, exactly X hours before another event, no earlier than X hours before another event, or between X and Y hours before another event. The TPE interface was enhanced to display the resulting constraints as curved blue arrows. A '+', '-', or '='

associated with the arrow indicates how the events associated with the endpoints of the arrow are constrained temporally to each other. A constant time, if specified with the constraint, is also displayed with the arrow. Figure 28 shows an example plan where the ingress to the objective is constrained to happen before the ambush task, which must occur before the egress. The start and end times for the movements and the task are also constrained to capture the concept that one can not arrive at a location before they have left their previous location.

The constraints are used by the system to notify the user of potential conflicts and allow the system to automatically update the temporal plan when the user changes the time at which an action occurs. When the user attempts to reposition an event, he is shown the domain of allowable times for this event based on the constraints and is prevented from extending an event beyond this domain. Once the event is repositioned, the constraints are propagated along arcs of the original constraint graph to select a satisfying temporal assignment that is minimally different from the one currently displayed. For example, if the end point of a task is extended by 30 minutes, all subsequent tasks that have been constrained to start after this task ends will be delayed so that no constraints are violated.
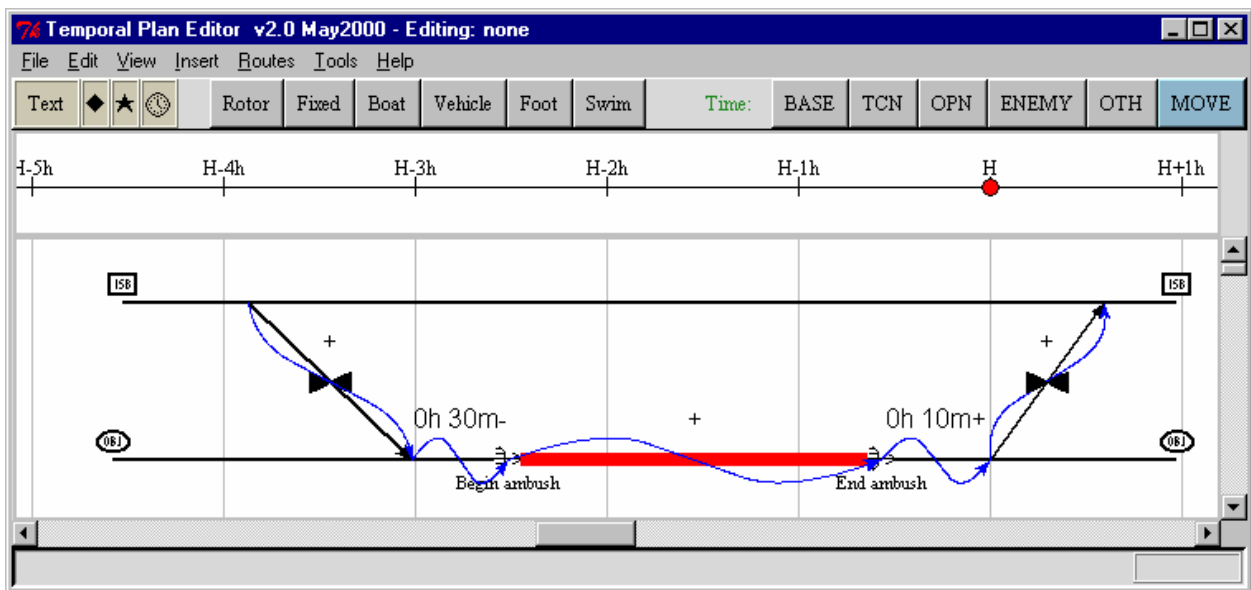
Figure 28: Example plan with temporal constraints.

3.4 SPECIAL FORCES MDMP ADAPTIVE PLANNING SOFTWARE

During this effort, we successfully developed and transitioned the Adaptive Planning Software (APS) version 1 to the Army Special Forces, specifically to support an Operational Detachment Alpha (ODA) in their application of the Military Decision Making Process (MDMP) during mission planning. Specifics about the software requirements, designs, test reports, and version descriptions were captured in the following documents, which were developed and delivered as part of this contract:

TR-1249 APS Software Requirements Specification

TP-382 APS Software Design Description

TR-1250 APS Software Test Report

TR-1251 APS Software Version Description

TR-1112 APS Software User's Manual

To initiate the development of the APS, we reviewed the Mission Planning Module (MPM), which represented the results of two previous attempts to convert the Navy SEALs' SWAMPS (Special Warfare Automated Mission Planning System) for the Army Special Forces domain. We also reviewed design documents supplied by the Army Special Operations Battle Lab (ARSOBL) that described the desired features of a new MPM. Following our review of the existing system and the functional requirements, we met with the ARSOBL in order to further discuss the initial designs for a new system.

The MPM was only partly functional. It consists of a web interface to a Microsoft Access database that stores information for various fields. The interface includes pointers to sources of relevant mission planning material such as geographic maps and satellite imagery. The interface also includes links to various local applications such as NetMeeting and TurboPrep. The interface guides an ODA through the seven steps of the MDMP to develop plans for a military operation. Each MDMP step requires the team to generate descriptions for many topics related to the mission, such as avenues of approach, cloud cover, socioeconomic factors, key facilities, and courses of action. Many of these topics are examined in multiple steps of the MDMP. To save the planning team time during mission planning, the system carries forward all previously entered information into subsequent steps of the MDMP. The system also allows the team to compare various courses of action that have been developed for the mission and automatically generates a Briefback PowerPoint presentation and an Operation Order (OPORD) Word document from the entered information.

In addition to a long list of known bugs in the existing system, end users of the system had identified four major deficiencies. First, the system was designed for the old tactical decision making process instead of the new MDMP. Due to the ever-changing nature of military doctrine, it is important that the new system provide an easy mechanism for updating the system to meet changes in doctrine. Second, the system lacked flexibility. New missions are likely to have their own unique problems and needs that get added to the standard information captured during the MDMP. A useful system needs to allow the planning team to create an unlimited number of new fields. Third, the system did not allow multiple users to work simultaneously. The planning team needs to develop the mission plan using multiple laptops connected together via a local area network that provides them with access to all of the data. Fourth, the system was very inefficient, with some basic operations taking 15-20 elapsed minutes to perform. In addition, users wanted the tool to be able to incorporate graphical data into the mission plan imported from the C2PC GIS environment. Graphical overlays are important for explaining key elements of a plan.

Through a rapid spiral prototyping process, we developed the new APS to address these issues and additional requirements identified jointly with the ARSOBL. The resulting APS v1 is designed on a client-server architecture as shown in Figure 29. The plan is stored in XML format in a MySQL database on a machine designated as the server. Client applications access this XML data and store it in a local cache where it can be rendered to an graphical user interface. Changes to the data are passed back to the server, which also controls data locks so only one user is modifying any piece of data at one time and reporting data changes when polled periodically by the clients.
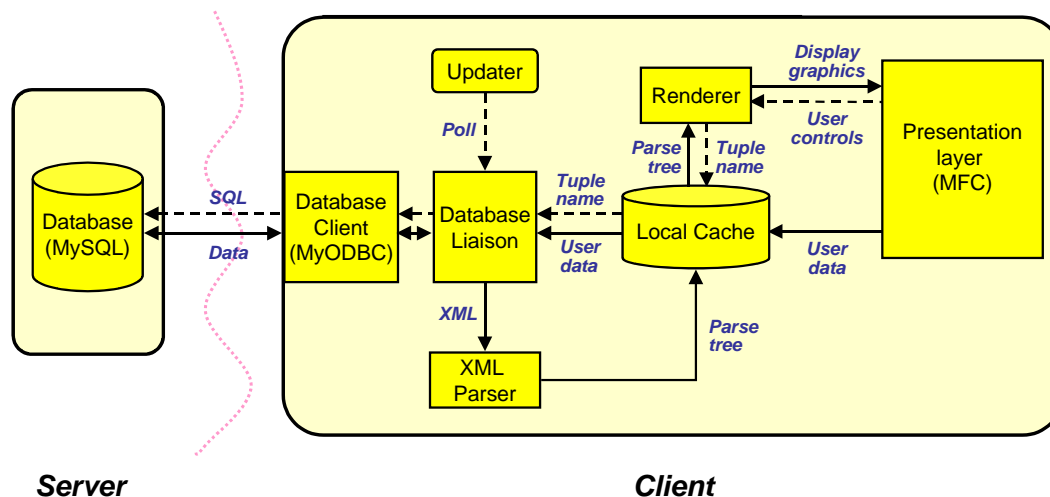
Figure 29: The MPM employs a client-server architecture

The resulting tool is based on DII-COE graphical user interface guidelines and provides the following capabilities:

1. *Page Administration:* allows the user to create a page hierarchy and associate text data fields with the pages in the hierarchy. The user may add and delete pages; modify page names; move pages around in the hierarchy; copy, paste, and delete fields; modify field names; and save the modified page hierarchy.

2. *Field Data Sharing*: allows data in one field to be shared with data in another field. Data sharing is specified the user by copying a field to be shared and then pasting it as a shared field. When the user types into a shared data field, this information is instantly shared to all local copies of that field.

3. *Data history:* allows the user to view the previous values of the fields in the mission folder. The user may select a date-time group for which to view the historical data. A copy of the mission folder is then displayed with the fields populated with the values available at the specified point in time.

4. *Example Mission Folder:* contains the page hierarchy and text fields for the 7 steps of the Military Decision Making Process (MDMP) used by an ODA.

5. *New File*: allows the user to create a new document from scratch or based on a previously developed document template.

6. *Save As File*: allows the user to save a document using a new file name. The user may save the document with all of its data or as a document template.

7. *Export to XML:* allows users to transfer information between APS databases. This capability could be used to support the transfer of information from one echelon to another via XML files.

8. *Import from XML*: allows the user to import XML files generated either by the APS or by the BBN DC2S TRACKER.

9. *Update from Imported*: allows the user to update the MDMP steps with data from the imported JSOTF products based on a predefined mapping. The system maps data imported

for Mission Initialization, Initial Mission Analysis, Warning Order, Mission Analysis prep for FRAG Order, and FRAG Order reports from the BBN DC2S TRACKER.

10.   *Spreadsheet Component*: allows the user to develop spreadsheets for things like lists of specified and implied tasks, COA criteria and scoring tables, decision matrices, and time schedules. Row and columns can be inserted and deleted; individual cells can be defined as read-only; and individual cells can be defined to share data with other cells in the current matrix or in other matrices defined within the document. Data sharing between spreadsheet cells is provided both bi-directionally (edits in any shared cell are applied to all of its shared cells) and uni-directionally (edits in one cell are shared to a second cell, but edits in the second cell are not shared back to the first cell). Entire spreadsheets can also be copied and pasted. The pasted spreadsheet may use just the template of the copied spreadsheet filled with copies of the existing data or it may bi-directionally share its data with all corresponding cells in the copied spreadsheet.

11.   *Image Component*: allows the user to load and edit graphics in the Special Forces (SF)-MDMP. Like text fields and spreadsheets, image fields can be added, deleted, renamed, printed (directory location of the image for text documents), copied, and pasted (with or without shared data) by using the context sensitive menu in the document tree view. A right click in the image field in the document page view provides further options, such as loading an image, editing an image, saving the field, and reverting the field. A request to edit the image causes the image to be brought up in an external application for editing purposes.

12.   *Selected COA Plan Generation*: allows the user to create a selected Coarse of Action (COA) Plan by selecting phases previously developed for different COAs. During COA development, a planning cell may develop several different COAs. During the course of analyzing and comparing these COAs, the team may decide to use the infiltration plan from one COA and the exfiltration plan from another COA. The APS provides a mechanism that allows the user to indicate which infiltration plan and other phases to incorporate into the final selected COA. The system then maps the data associated with the selected phases into the final selected COA.

13.   *Shared data reporting*: allows the user to verify that data fields are shared to all of the correct locations. Shared data reporting is most useful for process verification by those developing or modifying the layout of a document displayed in the SF-MDMP module.

14.   *Multi-user collaboration*: allows simultaneous development of a mission plan by multiple users on multiple machines connected through a LAN. Individual data fields are locked when a user begins modifying the data. Other users are notified that the data field is being modified and may continue working on other data fields. Once the user has completed his modifications, he may release the data lock and share the data updates with the other users.

15.   *Presentation Viewer*: allows the user to present/brief information in the SF-MDMP. The user may select a folder or field in the Process Tree view to present. The presentation viewer allows the user to cycle through the leaves and sub-folders in the Process Tree, presenting the contents of a sub-folder before continuing the presentation to the items following the sub-folder. The user may use a set previous and next buttons to cycle through the presentation or select a field in the process tree to change the currently presented field. The user may also increase and decrease the font size of text displayed in the display area.

16. *File Export to Microsoft Office® products*: supports the generation of PowerPoint® presentations and Word® documents. Sample document templates, which can be modified to meet specific user requirements, are included with the APS v1. The document templates are used by the SF-MDMP File Export capability to format the exported information. The MS Word® document template is based on Army Regulation 25-50 recommendations for Preparing and Managing Correspondence. The File Export capability also provides the user with the option of including or excluding fields containing no data.

17. *Data Security Classification:* associates data security classification information directly with each data object in the database. This ensures that the classification is shared along with the data to all locations that this data appears.

18. *Database Data Lock Identification:* used to identify data objects and components that are currently locked by the user or another client. The APS uses dark green for components and read-only grid cells locked by the user, light green for writeable grid cells locked by the user, dark red for components and read-only grid cells locked by other users, and light red for writeable grid cells locked by other users.

19. *InstallShield® Installation Packaging Scripts*: allows the system administrator to rapidly install and uninstall the application from NT based machines.

Throughout development, the APS was tested internally using a detailed software test plan and by various ODA end-user groups. ODA end-user testing occurred at ARSOBL facilities and at the Millennium Challenge 02 (MC02) warfighter experiment at Tonapah, NV. These tests provided valuable feedback on application usability and stability, and provided additional system requirements that were worked into subsequent development spirals.

Following the completion of the APS v1, we transitioned the technology to Brandes Associates, Inc., (BAI), for deployment, software maintenance, and enhancement. We provided BAI with a copy of the APS v1 and trained them on the use of the system. We also provided support in their requirements analysis and design phases for the APS v2, which is intended to be a joint-level collaborative planning tool. Our main focus during this support was to compare the proposed APS v2 system against the existing functionality in APS v1. During these reviews, we found potential deficiencies against several key APS v1 requirements and reported these back to BAI for their consideration.

4. CONCLUDING REMARKS

During this project, we investigated four major areas: SOF knowledge representations, template-based natural language user interfaces, plan critiquing and look-ahead constraint reasoning, and Special Forces Military Decision Making Process adaptive planning software. The following provides concluding remarks on each of these topics.

4.1 SOF KNOWLEDGE REPRESENTATIONS

As part of this investigation, we successfully developed an Air Field Seizure scenario that was used to test concepts being developed through the project. In addition, we developed a robust knowledge representation language that could serve as the backbone for our plan critiquer and look-ahead constraint reasoner. Using this language, we developed a knowledge representation of a military operation with constraints for critiquing a mission plan based on temporal, interval, resource, and parametric constraints.

4.2 TEMPLATE-BASED NATURAL LANGUAGE USER INTERFACE

During this phase of our project, we successfully developed a template-based natural language (TBNL) environment for building user interfaces. The environment allows a developer to specify a natural language grammar, which is then translated by the system into a graphical user interface. Using this environment, we developed two example uses of the technology. The first use merged the template-based natural language user interface with the Temporal Plan Editor for specifying task statements. The second use developed a standalone application for developing COA Statements. This second use employed the multiple sentence, multiple phase capability of the TBNL environment in order to guide the user in the development of COA statements that include statements about the reserve, obstacles, close support, fires support, rear support, deep support, end state, risk, security, and mission statement. By capturing these statements using a restricted natural language interface, the inputs are generated in a form that is natural to the user while also being easily interpreted by the system and usable for populating a knowledge base. In the case of a COA, this knowledge could be used to critique the COA.

4.3 PLAN CRITIQUER AND LOOK-AHEAD CONSTRAINT REASONER

For this phase of the project, we developed designs for a plan critiquer and look-ahead constraint reasoner. The designs included a decision point editor for defining branch plans, a situation editor for defining combat situations in which viable branch plans are to be merged, a plan critiquer for critiquing plans and reasoning over future possible alternatives, an enhanced temporal plan editor, and a message router to support multimodal plan development using these various editors. We also developed models of plan representations focusing on two key issues.

First, we investigated general methods of capturing and using branch plan information. During our investigation into modeling branch plan information, we found that one of the most important requirements for the system is a mechanism for merging branch plans. We identified that the act of merging may result in the deletion of facts that are true in the base plan. In addition, facts asserted

in two branch plans might contradict each other when merged. For example, the base plan may state that three tanks will be used for a particular task; a branch plan may replace the tanks with HMMWVs because a bridge has been weakened and can no longer support the weight of the tanks; another branch plan may replace the tanks with trucks in order to pick up hostages that have been found at the objective. We would like our system to have some simple mechanisms for handling these types of conflicts that the user could quickly learn and understand. The system will also support a mechanism that allows the user to override the automatically merged plan in order to correct mistakes introduced by these simple merging rules.

As a first demonstration of this design, we developed a demonstration prototype of a temporal constraint reasoner that combined an enhanced Temporal Plan Editor with the message router and a knowledge base with a plan critiquer in order to reason over temporal constraints defined in the Temporal Plan Editor. The resulting system used these constraints to limit the users ability to modify when events occur to those times that are valid given the current constraints and to automatically update the plan to maintain consistency.

4.4 SPECIAL FORCES MDMP ADAPTIVE PLANNING SOFTWARE

During this final phase of our project, we successfully developed and transitioned the Adaptive Planning Software to military use. This software provides a collaborative environment in which Special Forces ODAs can plan their missions and prepare their COA briefs and Operations Orders. The tool can be tailored to meet current and future operational needs. It maintains security classification on all data. This data can be shared to fields throughout the applications as well as being exported to an XML file for sharing between applications and echelons. An historical plan recall facility allows the users to view the state of the plan at any moment in history, and connection to Microsoft Word and Powerpoint allow the users to generate briefings and documents based on their inputs.

At the end of this contract, the APS v1 had received approval for inclusion in the SOMPE suite of tools and had been deployed to at least three Special Forces groups on approximate 40 machines. No major problems have been reported with the software. In addition, the FBI Hostage Rescue Team at Quantico is currently evaluating the software for their possible use.

5. RECOMMENDATIONS

Based on the outcomes of this project, we recommend further research and development in all four areas studied.

SOF Knowledge Representations: These concepts developed for SOF knowledge representations have already been transitioned to other ALPAHTECH projects including Agile Commander, which deals with military planning, and Intermodal Lift, including lift planning for special operations, and we would recommend further refinement and transition of these representations.

Template-Based Natural Language User Interface: The template-based natural language environment has demonstrated that it is possible to guide users in the development of human and machine understandable inputs. Future research includes further development of feature-based grammars that allow choices made in one part of the sentence to constrain the options available in other parts of the sentence. The difficult part of this research is developing a mechanism to provide an undo selection option. Since the feature constraint engine propagates the feature selection throughout a sentence, it becomes difficult to identify what is based on user selected inputs and what is based on constraint propagation. Several possible solutions exist to get around this, but an intuitive user interface must also be provided so that the user can properly control the undo mechanism. Another area for future research is supporting the guided entry of typed text. As the user types in text, the system would automatically use this text to fill in the natural language templates and provide warnings when illegal or ambiguous text is typed. This would enhance the user interface to allow users to communicate in a more natural method and would also open up the ability to interface the TBNL environment with a voice front end.

Plan Critiquer and Look-ahead Constraint Reasoner: During this project, we designed a multimodal branch plan editor with plan critiquer and look-ahead constraint reasoner. Due to time constraints, we were only able to develop a prototype temporal constraint reasoner to demonstrate some of the basic concepts. Our designs and the prototype temporal constraint system have laid the foundation for development of a full branch plan environment. In addition, there is further research that is needed for learning how to best represent branch plans so that they can be merged and reasoned over in order to build a full look-ahead constraint reasoner.

Adaptive Planning Software: Although the APS v1 has been successfully transitioned into operational use, there are many requirements that have been identified during end-user testing that could be added to the existing system. Among these are: tighter integration with existing Microsoft Office tools, development of more complex components to support specific mission needs, integrating additional mission planning support tools such as the Temporal Plan Editor, developing a component editor to allow users to create their own complex components, and integration with a knowledge base in order to support machine reasoning over the information entered by the user.

REFERENCES

Allen J. F. "Maintaining knowledge about temporal intervals," Communications of the ACM, volume 26, number 11, 1983, pp. 832-843.

ALPHATECH Template Based Natural Language Applications Software User's Manual, v.1.0, ALPHATECH, Inc., January 2002.

ALPHATECH Template Based Natural Language Applications Software Developer's Guide, v.1.0, ALPHATECH, Inc., January 2002.

Bratko, Ivan. "Prolog Programming for Artificial Intelligence." Addison-Wesley Publishing Company, Reading, Massachusetts, 1991.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., & Rice, J. P. (1998). OKBC: A Programmatic Foundation for Knowledge Base Interoperability. Proceedings of the Fifteenth National Conference on Artificial Intelligence. Madison, Wisconsin.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., & Rice, J. P. (1998). Open Knowledge Base Connectivity 2.0.3 (Proposed). http://www.ksl.stanford.edu/software/OKBC/.

Guha, R. V. (1991) Contexts: A Formalization and some Applications, Technical Report ACT-CYC-423-91, MCC, Austin, TX

The HPKB Course Of Action Challenge Problem Specification (version 1.1), available in .pdf, .doc, or .ps formats at http://www.alphatech.com/protected/hpkb/documents, userid: hpkb, password: hpkb411, 1998.

Martin R. Frank and Pedro Szekely. Adaptive Forms: An interaction paradigm for entering structured data. In Proceedings of the ACM International Conference on Intelligent User Interfaces, pages 153-160, (San Francisco, California, January 6-9) 1998.

Minsky, M. (1975) "A framework for representing knowledge" in P. Winston ed., The Psychology of Computer Vision, McGraw-Hill, New York, pp. 211-280

Technical Memo HPKB-COA-7 Additional Scenarios for the COA Challenge Problem, available at http://www.alphatech.com/protected/hpkb/documents/tm-hpkb-coa-7-v2.1.5.doc, userid: hpkb, password: hpkb411, 2000.

Technical Memo HPKB-COA-8 Knowledge Fragments for the COA Challenge Problem, available in .pdf, .doc, or .ps formats at http://www.alphatech.com/protected/hpkb/documents, userid: hpkb, password: hpkb411, 1999.

Technical Memo HPKB-COA-9 Performance Evaluation for the COA Challenge Problem, available in .pdf, .doc, or .ps formats at http://www.alphatech.com/protected/hpkb/documents, userid: hpkb, password: hpkb411, 1999.

Technical Memo HPKB-COA-10 Revised Grammar for COA Statements and the Products of Mission Analysis, available in .pdf, .doc, or .ps formats at

http://www.alphatech.com/protected/hpkb/documents, userid: hpkb, password: hpkb411, 1999.

Technical Memo HPKB-COA-11 Decision Point Case Descriptions, available in .pdf, .doc, or .ps formats at http://www.alphatech.com/protected/hpkb/documents, userid: hpkb, password: hpkb411, 1999.

Technical Presentation 382 Adaptive Planning Software v1.0 Software Design Description, ALPHATECH, Inc., December, 2002.

Technical Report 1112 Software User's Manual for Adaptive Planning Software v1.0, ALPHATECH, Inc., December, 2002.

Technical Report 1249 Software Requirement Specification for Adaptive Planning Software v1.1, ALPHATECH, Inc., September, 2003.

Technical Report 1250 Software Test Report for Adaptive Planning Software v1.1.1, ALPHATECH, Inc., December, 2003.

Technical Report 1251 Software Version Description for Adaptive Planning Software v1.1.1, ALPHATECH, Inc., December, 2003.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AcT	Active Templates
AFRL	Air Force Research Lab
AFS	Airfield Seizure
APS	Adaptive Planning Software
ARSOBL	Army Special Operations Battle Lab
COA	Course of Action
CSP	Constraint Satisfaction Problem
DA	Direct Action
DARPA	Defense Advanced Research Projects Agency
DCG	Definite Clause Grammar
DZ	Drop Zone
HPKB	High Performance Knowledge Bases
ICCES CEP	Integrated Course of action Critiquing and Elaboration System Concept Experimentation Program
IPT	Integrated Product Team
KB	Knowledge Base
KRL	Knowledge Representation Language
MC02	Millennium Challenge 2002
MDMP	Military Decision Making Process
MPM	Mission Planning Module
NEO	Noncombatant Evacuation Operation
ODA	Operational Detachment Alpha
OPLAN	Operations Plan
OPORD	Operation Order
SBIR	Small Business Innovation Research
SEAL	Sea, Air, Land
SF	Special Forces
SME	Subject Matter Expert

SOF   Special Operations Forces

SWAMPS   Special Warfare Automated Mission Planning System

TBNLUI   Template-Based Natural Language User Interface

TCR   Temporal Constraint Reasoner

TPE   Temporal Plan Editor

USSOCOM   United States Special Operations Command

XML   eXtreme Markup Language

APPENDIX A: AIRFIELD SEIZURE SCENARIO MATERIALS

1. COA SKETCH AND STATEMENT FOR ACT AFS SCENARIO, PHASE II
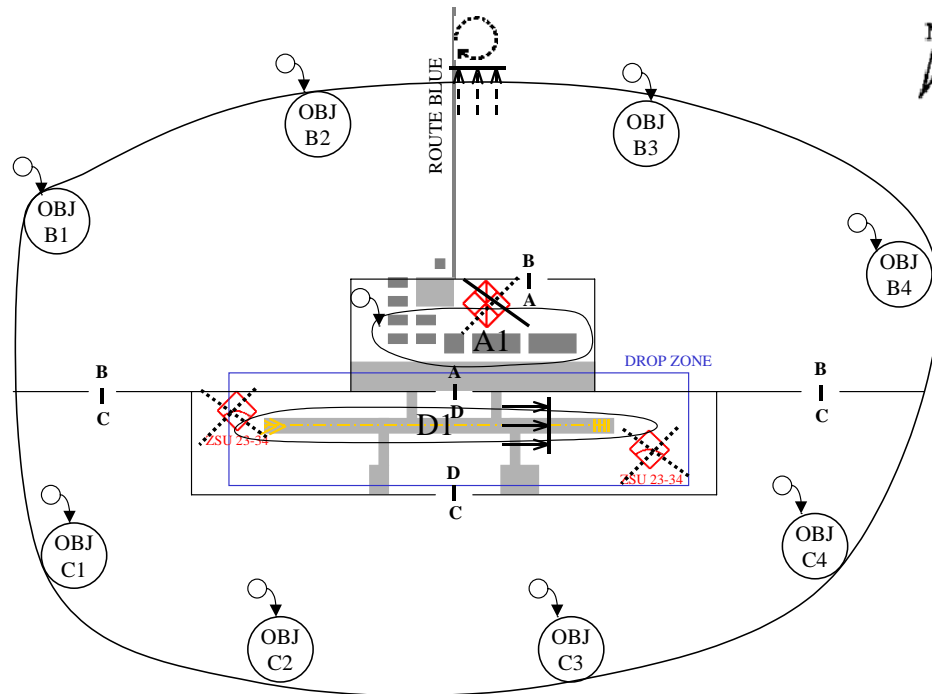
Figure 30. COA Sketch for AcT AFS Scenario, Phase II

Mission. On order, TF CHARLIE, an airborne infantry battalion, conducts airborne assault to seize Freedomtown Airport in order to establish a lodgment and conduct airland operations.

Close: An infantry company (TEAM 1), the initial main effort, attacks to clear OBJ D1 in order to prevent enemy forces from interfering with preparation of the airfield, and then attacks to seize key facilities and neutralize any threat in the immediate vicinity of the airfield. An antitank company (-) augmented with engineers (TEAM 4), assumes the main effort and clears OBJ D1 in order to ensure that the runway is free of enemy forces, foreign objects and debris (FOD), and damage that would prevent the use of the runway for airland operations. One infantry company (TEAM 2) seizes assault objectives B1, B2, B3, and B4 and secures the northern half of the airfield. Another infantry company (TEAM 3) seizes assault objectives C1, C2, C3, and C4 and secures the southern half of the airfield. On order TEAM 4 clears ROUTE BLUE and secures ROUTE BLUE in support of non-combatant evacuation operations.

Reserve: None.

Security: An R&S team inserted 24 hours prior to P-hour provides intelligence updates directly to the airborne commander prior to P-hour.

Deep: In deep operations, an AC-130 gunship destroys enemy air defense assets in the vicinity of Freedomtown Airport at P-:05.

Rear: Rear operations remain in the ISB.

47

Fires: Fire support is initially provided by AC-130 gunship. The task force will also heavy-drop one battery of direct support field artillery and all organic mortars. Fires will suppress enemy counterattacks against the lodgment.

Obstacles:

Risk:

End State: At the conclusion of this operation, the airhead is secure and the airfield is prepared for airland operations. TEAM 1 secures the northern half of the airhead. TEAM 2 secures the southern half of the airhead. TEAM 3 secures the airfield and its key facilities. TEAM 4 secures ROUTE BLUE.

1.1   COA STATEMENT FOR ACT AFS SCENARIO, PHASE III

Mission. TF CHARLIE secures the airhead and, when directed, facilitates airland operations by providing army airspace command and control (A2C2), marshalling, and airport operations. On order, TF CHARLIE will facilitate the evacuation of non-combatants through the lodgment by way of aircraft.

Close: An infantry company (TEAM 1) secures the airfield and its key facilities. An antitank company (-) (TEAM 4) secures ROUTE BLUE between the airport and the Freedomtown bridge. One infantry company (TEAM 2) secures the northern half of the airhead. Another infantry company (TEAM 3) secures the southern half of the airhead.

Reserve: An antitank platoon is the battalion reserve.

Security: Scout platoon screens forward of the airhead line.

Deep:

Rear: Rear operations remain in the ISB.

Fires: Fires will neutralize enemy mounted forces and air defense assets.

Obstacles:

Risk:

End State: At the conclusion of this phase, the non-combatant evacuation is complete, airland operations are complete, and the task force is prepared to hand over operation of the airfield to local authorities. TEAM 1 secures the northern half of the airhead. TEAM 2 secures the southern half of the airhead. TEAM 3 secures the airfield and its key facilities. TEAM 4 secures ROUTE BLUE.

1.2 DRAFT OPLAN FOR ACT AFS SCENARIO

<div align="right">
Copy __ of __ copies
<Headquarters>
<Location>
<DTG>
</div>

Reference:
<MapSheet>

Time zone used throughout the plan: ZULU

Task Organization: Annex A (Task Organization)

1. SITUATION

    a. Enemy Forces. Annex B (Intelligence)

    b. Friendly Forces.

        (1) JSOTF

           (a) JSOTF mission statement. When directed, JSOTF conducts non-combatant evacuation operations (NEO) to recover American citizens (AMCITS) in Freedomtown, Cordilerre and return them to US soverignty.

           (b) JSOTF intent.

               (1) Purpose. To provide assistance to the US Ambassador at the US Embassy in Freedomtown: secure the compound; evacuate the AMCITs to US authority located on a US naval vessel TBD.

               (2) Method. JSOTF forces will alert, assemble, and deploy to West Africa area of operations (AOR). This must be a flexible tailored force capable of securing and evacuating up to 800 AMCITs from Freedomtown. The advance force will focus efforts on priority marshalling areas and ground lines of communication (LOCs)/routes to the evacuation control center to be selected and established. We must maintain a high level of tactical discipline while processing non-combatants; interfacing with the media and host nation personnel/government officials; maintaining security and protecting the force. The international perception of this mission should project a well-organized, small, and efficient operation. We must prevent collateral damage and noncombatant casualties. JSOTF will establish the capability to conduct an emergency assault in the event of imminent loss of life.

               (3) Endstate. In the end, all personnel desiring evacuation and JSOTF forces are safely recovered/redeployed.

        (2) TF ALPHA (Ground component)

(a) TF ALPHA mission statement. On order, TF ALPHA conducts military operations to seize Freedomtown Airport and to evacuate non-combatant AMCITs located in Freedomtown, Cordilerre to a US Naval vessel (TBD) in order to safeguard American Citizens in Cordilere and return them to U.S. authority.

(b) TF ALPHA intent.

(1) Purpose. To safeguard AMCITS in Cordilere; secure the embassy compound; evacuate the AMCITs to US authority located on a US naval vessel TBD.

(2) Method. TF ALPHA will alert, assemble, and deploy with JSOTF to West Africa area of operations (AOR). Special forces teams will conduct special reconnaissance to confirm target status and facilitate direct action. The evacuation will be conducted by five special forces teams, each at a separate marshalling area. AMCITS will be removed from marshalling areas, directly to the safe haven (afloat) via helicopter. Minimize the amount of time spent at marshalling areas, and remove evacuees from harm as quickly as possible. An airborne battalion will simultaneously conduct an airfield seizure to secure a logdment for airland operations in order to return the airport to Cordilere authorities and provide relief and support to the country's government. If evacuation of AMCITS directly to ship by helicopter is frustrated, we will secure AMCITS at the airport and evacuate them via the first available return aircraft. Maintain a high level of tactical discipline and operational security (OPSEC) while interfacing with the media and host nation personnel/government officials. Protect the force, and prevent collateral damage and noncombatant casualties.

(3) Endstate. In the end, all personnel desiring evacuation and JSOTF forces are safely recovered/redeployed, Freedomtown airport is secured and returned to the nation of Cordilere.

(3) JSOTF advance parties will establish an intermediate staging base (ISB).

(4) TF BRAVO (aaritime component) provides maritime support to operations.

(5) JSOACC (air component) provides air support to operations.

c. Attachments and Detachments. Annex A (Task Organization). Effective on order.

d. Assumptions.

(1) US Navy fleet will provide an appropriate vessel to support the entire evacuation, to include basing heliborne assets.

(2) USSOCOM as well as US Air Force AMC and ACC will provide the appropriate forces to support recommended COAs presented by JSOTF to Theater CINC and JCS.

(3) Basing rights stipulated in "Status of Forces" agreement with Friendlyland will be adhered to.

2. MISSION

On order, TF CHARLIE conducts airborne assault to seize Freedomtown Airport in order to establish a lodgment and conduct airland operations.

3. EXECUTION

Intent:

(1) Purpose. Establish a lodgment and conduct airland operations, provide an alternate method for evacuation of non-combatants.

(2) Method. TF CHARLIE forces will alert, assemble, and deploy to West Africa area of operations (AOR) with JSOTF and TF ALPHA. Conduct airborne assault to seize Freedomtown airport. Key to this operation is the destruction of enemy air defense systems, rapid seizure of initial objectives, and strict tactical discipline on the drop zone. Assemble minimum essential force as quickly as possible to rapidly seize key facilities and secure the airhead. Clear and certify the runway and be prepared for airland operations, and establish A2C2 NLT P+2. Maintain a secure airhead throughout the operation, and be prepared to secure evacuees and evacuate to the ISB.

(3) Endstate. Freedomtown airport secure and returned to the control of the government of Cordilere. Airland operations complete, with no loss of aircraft or non-combatant life. TF CHARLIE redeployed to ISB.

a. Concept of operations.

Phase I (Alert, marshal, and deploy) – When directed, TF CHARLIE prepares for airborne forced entry operations. The task force Alerts from home station per SOP, and moves to an intermediate staging base (ISB) in Friendlyland for marshaling operations. The task force deploys from the ISB. Phase I ends at P-hour (when the first jumper exits the aircraft at the drop zone).

Phase II (Forced entry operations) – At P-hour, D-Day, TF CHARLIE conducts airborne assault to seize Freedomtown Airport. In deep operations, An AC-130 gunship destroys enemy air defense assets in the vicinity of Freedomtown Airport at P-:05. In security operations, an R&S team inserted 24 hours prior to P-hour provides intelligence updates directly to the airborne commander prior to P-hour. In close operations, the task force conducts parachute assault at P-hour. An infantry company, the initial main effort, attacks to seize key facilities and neutralize any threat in the immediate vicinity of the airfield. An antitank company (-) augmented with engineers (on order main effort), clears the runway in order to ensure that the runway is free of enemy forces, foreign objects and debris (FOD), and damage that would prevent the use of the runway for airland operations. One infantry company seizes assault objective in the northern half of the airhead line. Another infantry company seizes assault objective in the southern half of the airhead line. Fire support is initially provided by AC-130 gunship. The task force will also heavy-drop one battery of direct support field artillery and all organic mortars. Rear operations remain in the ISB until airland operations commence. Phase II ends when the airhead is secure and the airfield is prepared for airland operations.

Phase III (Lodgment operations) – TF CHARLIE secures the airhead and, when directed, facilitates airland operations by providing army airspace command and control (A2C2), marshalling, and airport operations. In security operations, two infantry companies continue to

secure the airhead line, and adjust it as necessary. In close operations, an infantry company provides security to the airfield and key facilities. An antitank company (-) secures key routes between the airport and the Freedomtown bridge. Fires will neutralize enemy mounted forces and air defense assets. Key rear operations will relocate with the airland aircraft to the lodgment. On order, TF CHARLIE will facilitate the evacuation of non-combatants through the lodgment by way of aircraft. Phase III ends on order.

Phase IV (Battle handover) – When directed, TF CHARLIE will turn the Freedomtown Airport over to the local authorities. Phase IV ends when the security of the Freedomtown airport is returned to legitimate military and/or police forces and TF CHARLIE has returned to the ISB.

Phase V (Redeploy) – On order the task force redeploys from the ISB to home station. Phase V ends when all of TF CHARLIE has returned to home station.

(1) Maneuver.

(a) Phase I (omitted)

(b) Phase II

TEAM 1 (A Company) attacks to seize OBJ A1 in order to gain control of the airport tower and key facilities necessary to land aircraft at Freedomtown airport.

TEAM 2 (B Company) attacks to seize OBJ B1, OBJ B2, OBJ B3, and OBJ B4 in order to prevent the enemy from observing the airfield or engaging it with direct fires.

TEAM 3 (C Company) attacks to seize OBJ C1, OBJ C2, OBJ C3, and OBJ C4 in order to prevent the enemy from observing the airfield or engaging it with direct fires.

TEAM 4 (D Company) clears OBJ D1 in order to ensure that the runway is free of enemy forces, foreign objects and debris (FOD), and damage that would prevent the use of the runway for airland operations.

(2) Fires. (Annex D). AC-130 gunship will be on station at P-30 minutes. AC-130 provides preparatory fires in Phase I, and direct fire support to TF CHARLIE during Phase II. The task force will heavy-drop organic mortars and one battery of field artillery. Mortars and artillery will be prepared to fire NLT P+:20.

(3) Reconnaissance and surveillance. An R&S team will be in place to provide intelligence updates and target acquisition during Phase I of the operation.

(4) Intelligence.

(5) Information operations.

b. Tasks to maneuver units.

(1) TEAM 1 (A Company)

Phase II:
        (a) Initial main effort.

        (b) Neutralize enemy forces in OBJ D1.

        (c) Seize OBJ A1.

        Phase III:

        (d) Provide security to the airfield and key facilities.

(2) TEAM 2 (B Company)

        (a) Seize OBJs B1, B2, B3, B4 NLT P+1.

        (b) Neutralize all enemy forces within the airhead in sector.

        (c) Coordinate between OBJs B1 and C1, and between B4 and C4.

        Phase III:

        (d) Patrol the airhead line in sector.

(3) TEAM 3 (C Company)

        (a) Seize OBJs C1, C2, C3, C4 NLT P+1.

        (b) Neutralize all enemy forces within the airhead in sector.

        (c) Coordinate between OBJs B1 and C1, and between B4 and C4.

        Phase III:

        (d) Patrol the airhead line in sector.

(4) TEAM 4 (D Company)

        (a) On order (~P+:20) main effort.

        (b) Prepare heavy-dropped HMMWVs and weapon systems for operation
NLT P+:20.

(c) After TEAM 1 neutralizes enemy forces in OBJ D1, clear remaining enemy forces from OBJ D1.

(d) Conduct an initial assessment of the runway NLT P+:30 to determine its suitability for airland operations.

(e) Clear the runway of foreign objects and debris (FOD).

(f) Repair damage to runway as necessary to certify it for assault landing. Use engineer support to repair surface damage.

(g) Coordinate with CCT element to certify the condition of the runway and all runway repairs.

Phase III:

(g) Clear Route Blue in order to prepare it for the movement of evacuation vehicles in support of NEO.

(h) Establish link-up with SF team at Freedomtown bridge, at the north end of Route Blue.

(i) Provide route security and guidance during NEO.

c. Tasks to combat support units.

d. Coordinating instructions.
(1) This plan is effective for planning and implementation on receipt. D-Day, H-hour is on order.
(2) Commander's critical information requirements (CCIR)
(a) Priority intelligence requirements (PIR).
(b) Essential elements of friendly information (EEFI).
(c) Friendly force information requirements (FFIR).
(3) Rules of Engagement: Annex E.

(4) Force protection. Air defense warning status YELLOW; air defense weapon control status WEAPONS TIGHT.

(5) All parachutes in the vicinity of the airstrip must be bagged or removed from the vicinity of the runway as soon as possible, and NLT P+1.

(6) Air movement will be conducted IAW air movement appendix to Annex C (operations).

4. SERVICE SUPPORT (omitted).

5 COMMAND AND SIGNAL

a. Command.

 (1) Phase I: Once airborne, the airborne commander provides C2 via in-flight communications.

 (2) Phase II: Assault CP will assemble at the lead edge of the runway immediately after P-hour.

 (3) Phase III: Assault CP will move to the control tower and establish TAC CP.

b. Signal.

 (1) Phase I: In-flight communications provided by air component.

 (2) Phase II-V: Handheld radios are initial means of communication on the drop zone. Primary tactical communications are via FM. FM nets are to be operational NLT P+:05. TACSAT provides communications to ISB, higher headquarters, and adjacent units.

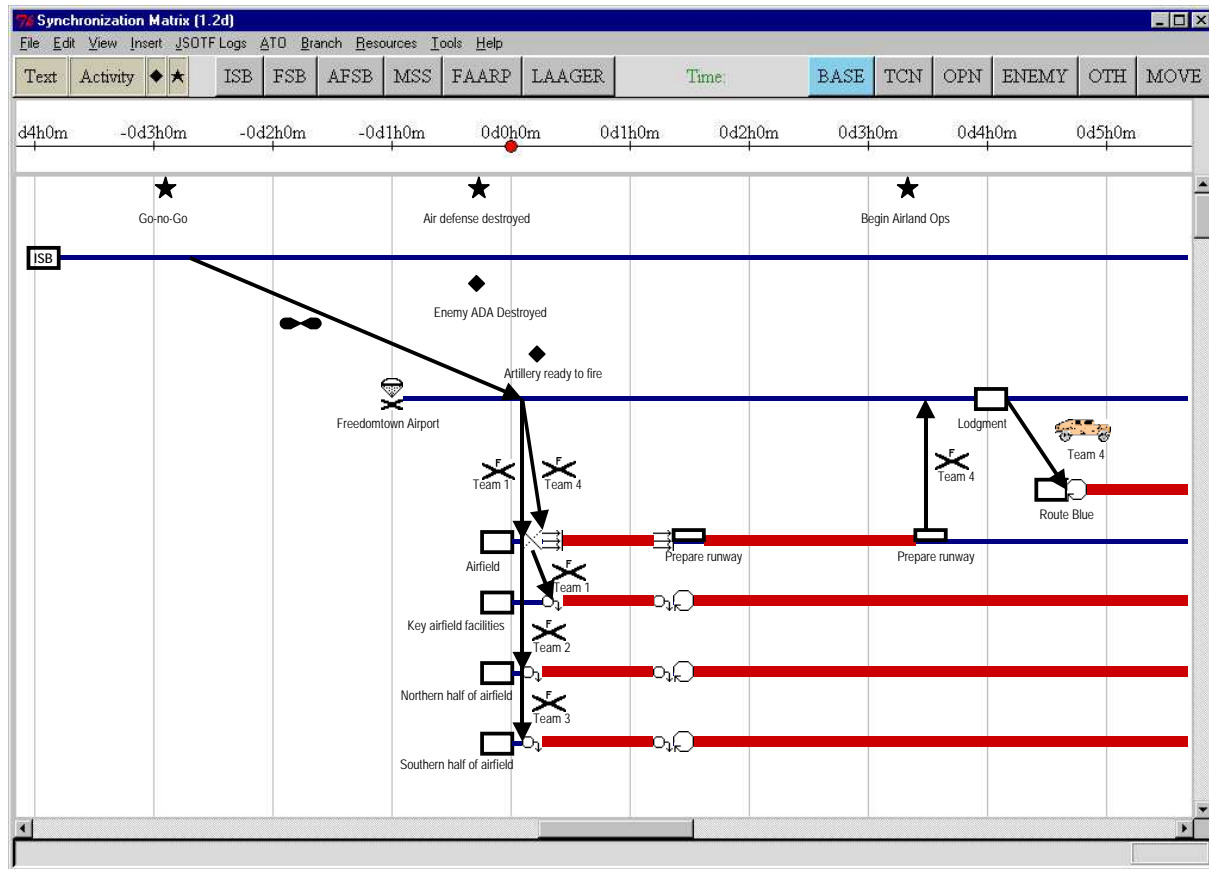ACKNOWLEDGE:

        JONES
        LTC, Commanding

OFFICIAL:

/s/Smith
SMITH
S3

ANNEXES: A—Task Organization
    B—Intelligence
    C—Operations Overlay
    D-Fire Support
    E-Rules of engagement

1.3   TEMPORAL PLAN FOR ACT AFS SCENARIO

APPENDIX B: MILITARY OPERATION KNOWLEDGE REPRESENTATION

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright(c) ALPHATECH, Inc. Burlington, MA 01803 ALL RIGHTS RESERVED
%
% CM Header:
%
% Module Name: militaryOps
%
% Software Class: II
%
% Date Created: 2/28/2001
%
% Author: Chris White
%
% Module Description:
%
%   Purpose: Knowledge Representation of a Military Operation.
%       Military operation is the basis of a mission.  It is a container for
%       tasks and sub-operations.  It can be used to represent a mission,
%       provide a top-level object for structuring a COA (decomposition of a
%       mission into sub-missions, packaging several tasks constituting a main
%       or supporting effort, package several tasks constituting a main attack,
%       and represent shifting of the main effort
%
%   References:
%
%   Module History:
%
% |==========+==============+==============================================+
% |Date      |Programmer    | Description of Modification                   |
% |==========+==============+==============================================+
% |2/28/2001 |Chris White   | Initial version                              |
% |==========+==============+==============================================+
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NOTE: (1) and (0) are equivalent to a cardinality of functional
%       (1+) and (0+) are equivalent to a cardinality of multivalued

context mission

class militaryOperation
      % The Mission
      agent : militaryForce   (1)    % Who
      tasks : task        (1+)  % What
      location : location     (1)    % Where
      opInterval : interval   (1)    % When
      purpose : string  (1)    % Why

      endState : unitPosture  (0+)

      % Decomposition into sub-missions
      subOperations : militaryOperation   (0+)  % How
      supportingEfforts : militaryOperation      (0+)
```

```
        mainEffort : militaryOperation       (0)

        constraints:
                % all tasks occur during the interval of the operation
                forall(tasks, opInterval)
                        exists(taskInterval : interval)
                                [interval(tasks, taskInterval)
                                 contains(opInterval,taskInterval)]
                % all supporting efforts are also suboperations
                forall(supportingEfforts)
                        exists(subOp : militaryOperation)
                                [subOperations(subOp, supportingEfforts)]
                % the main effort is also a suboperation
                forall(mainEffort)
                        exists(subOp : militaryOperation)
                                [subOperations(subOp, mainEffort)]
end

class unitPosture
        posture : string  (1)
end

%
% -------------------------------------------------------------------------------
%
% Tasks include doctrinally specified tactical tasks, movements, and other
% needed tasks.
%
% -------------------------------------------------------------------------------
%

class task
        agent : militaryForce    (1)    % Who
        taskInterval : interval (1)      % When
end

%
% -------------------------------------------------------------------------------
%
% Tactical tasks are a doctrinally specified subclass of tasks.
%
% -------------------------------------------------------------------------------
%

class tacticalTask : task
        typeOfOperation : operationType       (1)
        formOfManeuver : maneuverType (1)
        purpose : string  (1)    % Why
end

class operationType
        type : string       (1)

        constraints:
                forall(type)
                        [member(type, [attack, defend])]
end

class maneuverType
        type : string       (1)
```

58

```
end

% Tactical tasks affecting an enemy

class ambushTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class attackByFireTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class blockTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class bypassTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class breachTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class canalizeTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class containTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class demonstrateTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class destroyTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class defeatTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class disruptTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class exploitTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class feintTT : tacticalTask
     objectActedOn : militaryForce (1)
end

class fixTT : tacticalTask
     objectActedOn : militaryForce (1)
end
```

```
class interdictTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class neutralizeTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class penetrateTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class reconnoiterForceTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class ruptureTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class supportByFireTT : tacticalTask
      objectActedOn : militaryForce (1)
end

% Tactical tasks affecting terrain

class clearTT : tacticalTask
      objectActedOn : location      (1)
end

class controlTT : tacticalTask
      objectActedOn : location      (1)
end

class occupyTT : tacticalTask
      objectActedOn : location      (1)
end

class reconnoiterLocTT : tacticalTask
      objectActedOn : location      (1)
end

class retainTT : tacticalTask
      objectActedOn : location      (1)
end

class secureTT : tacticalTask
      objectActedOn : location      (1)
end

class screenTT : tacticalTask
      objectActedOn : location      (1)
end

class secureTT : tacticalTask
      objectActedOn : location      (1)
end

class seizeTT : tacticalTask
      objectActedOn : location      (1)
```

```
end

% tactical tasks affecting friendly forces

% Breach is already defined for military force under enemy

class disengageTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class displaceTT : tacticalTask
      objectActedOn : militaryForce (1)
end

class exfiltrateTT : tacticalTask
      objectActedOn : militaryForce (1)
end

%
% ------------------------------------------------------------------------------
%
% Movements are another form of task.  It has interval constraints on the
% pre and post location of the transport vehicle and the cargo.
%
% ------------------------------------------------------------------------------
%

class movement : task
      startLocation : location      (1)
      endLocation : location  (1)
      transport : vehicle     (1+)
      cargo : asset      (1+)

      constraints:
            % constraints on the transport vehicle
            forall(transport, startLocation, taskInterval)
                  exists(preInterval : interval)
                        [at(preInterval, transport, startLocation),
                         meets(preInterval, taskInterval)]
            forall(transport, endLocation, taskInterval)
                  exists(postInterval : interval)
                        [at(postInterval, tranport, endLocation),
                         meets(taskInterval, postInterval)]
            % constraints on the cargo
            forall(cargo, startLocation, taskInterval)
                  exists(preInterval : interval)
                        [at(preInterval, cargo, startLocation),
                         meets(preInterval, taskInterval)]
            forall(cargo, endLocation, taskInterval)
                  exists(postInterval : interval)
                        [at(postInterval, cargo, endLocation),
                         meets(taskInterval, postInterval)]
end

%
% ------------------------------------------------------------------------------
%
% An interval is made up a start point and end point, which are events.
% and maybe a duration, and type of measurement.
%
```

```
% ------------------------------------------------------------------------------
%

class interval
      startPoint : event      (1)
      endPoint : event  (1)

      constraints:
            % The start point must always occur before the end point
            forall(startPoint, endPoint)
                  [startPoint =< endPoint]
end

class event
      name : string      (1)
      timePoint : timePoint (0)
end

class timePoint
      time : number      (1)
end

%
% ------------------------------------------------------------------------------
%
% Location
%
% ------------------------------------------------------------------------------
%

class location
      name : string      (1)
      coordinates : coordinates      (0)   % centroid of region
      subLocations : location (0+)
      terrain : terrainType   (0)
end

class coordinates
      latitude : number (1)   % in seconds of arc
      longitude : number      (1)   % in seconds of arc
end

class terrainType
      type : string      (1)

      constraints:
            forall(type)
                  [member(type, [unpaved, paved, water])]
end

%
% ------------------------------------------------------------------------------
%
% Assets: units and equipment
%
% ------------------------------------------------------------------------------
%

class asset
      name : string      (1)
```

62

```
end

class militaryForce : asset
end

class unit : militaryForce
      affiliation : unitAffiliation (0)
      type : sting        (0)
      echelon : unitEchelon   (0)
      unitOfAssignment : unit (0)
      attachedToUnit : unit   (0)
      opConByUnit : unit      (0)
      assignedUnits : unit    (0+)
      attachedUnits : unit    (0+)
      opConUnits : unit (0+)
end

class unitAffilication
      type : string     (1)

      constraints:
            forall(type)
                  [member(type,
                        [friend, assumedFriend, unknown, pending,
                        neutral, enemy, suspected, joker, faker])]
end

class unitEchelon
      type : string     (1)

      constraints:
            forall(type)
                  [member(type,
                        [installation, team, crew, squad,
                        section, platoon, detachment, company,
                        battery, troop, battalion, squadron, regiment,
                        group, brigade, division corps, army,
                        armyGroup, front, region,
                        platform, soldier, meusoc, mef, theater])]
end

class equipment : asset
      mass : measurement       (0)
      length : measurement     (0)
      width : measurement      (0)
      height : measurement     (0)
end

class vehicle : equipment
      numberCrewMembers : number    (0)
      range : measurement      (0)
      maxSpeed : measurement   (0)
      maxPax : number    (0)
      maxPayload : measurement      (0)
      cargoBayLength : measurement  (0)
      cargoBayWidth : measurement   (0)
      cargoBayHeight : measurement  (0)
      trafficableTerrain : terraintype (0+)
end
```

```
class measurement
      type : string      (1)
      amount : number    (1)
end
```