16th ICCRTS
"Collective C2 in Multinational Civil-Military Operations"

Plan Failure Analysis and Plan Adaptation for Multi-Level Campaign Planning

Topic 2: Approaches and Organizations

Name of Author(s)
Jens Happe
MacDonald, Dettwiler and Associates (MDA)
13800 Commerce Parkway
Richmond, BC, Canada, V6V 2J3

Mohamad Allouche, Micheline Bélanger
R & D Defence Canada - Valcartier
2459 Pie XI North
Quebec, QC, Canada, G3J 1X5

Point of Contact
Jens Happe
MacDonald, Dettwiler and Associates (MDA)
13800 Commerce Parkway
Richmond, BC, Canada, V6V 2J3
e-mail: jhappe@mdacorporation.com

Abstract

The military campaign planning process involves multiple commanders at different levels of command and with different areas of responsibility. In such a highly interdependent and data-intensive activity, it becomes difficult to coordinate all plans and plan components to generate a joint plan free of inconsistencies, or even to track the source of an inconsistency, let alone repair it. This paper presents a plan management approach based on Hierarchical Task Networks. It demonstrates how this approach can be developed into a system for visual analysis of plans, plan validation and monitoring, and explains on a practical example how such a system can trace failures through different levels of command, so as to detect and repair inconsistencies between plans.

1. Introduction

The complexity of today's operations requires the involvement of a multidisciplinary team that can be distributed in time and space. For example, different military environments may be working in parallel on their respective part of a global plan. Then, considering that multiple plans may have to be executed in parallel, different operational commanders as well as tactical commanders will require to have real-time access to information about the execution of these different plans based on their responsibilities. Accordingly, to support the Canadian Forces in their planning and monitoring activities, this work addresses their need for an intuitive distributed, real-time multi-plan management system. The system must analyze dependencies and possible conflicts between plans (e.g. regarding resource use). It must adequately cater to commanders, operators and users responsible for operations in different locations and at different levels of abstraction. It must respond to the highly dynamic and uncertain situation in the operational theatre and allow plan adaptations and repairs. On the other hand, it should not require military personnel to become knowledge experts, but instead it should display

plans and plan elements in familiar, easy to absorb, graphical and text-based representations. The amount of information should be tailored to the respective operator's need and include elaborate details only if the operator specifically drills down for them. Finally, it is desirable to furnish a formal concept for validating the correctness and executability of plans and attainment of the mission goals.

This work borrows largely from the theory of Hierarchical Task Networks (HTN), a branch of Artificial Intelligence planning. It supports the concept of hierarchical plan decomposition and instantiation of plan templates according to the given situation, locations, and resource assignments. The pure HTN approach falls short of military needs, though, as it has no notion of goals or time (time points and durations). It also does not provide for executing plans and actions in parallel. Our main accomplishment is an adaptation of HTN to the needs of military plan execution, incorporating ideas from Hierarchical Goal Analysis (HGA) (Hendy et al., 2002) and Effects-based Reasoning (Farrell, 2007) to represent and reason with goals and effects, as well as ideas from Scheduling Theory to reason with time.

An integral part of this description is how these concepts are presented to military users in an intuitive user interface. Among the features shown are: 1. a plan and goal hierarchy representing the user's scope of responsibility and allowing drill-down into lower-level goals and plans, 2. a map view representing the spatial aspects of plans and/or goals, 3. a schedule view representing the temporal allocation of plans and executing resources, 4. a browser view allowing cross-hierarchy navigation and tracing of plan failures to the elements that caused them, and 5. user dialogues to facilitate plan adaptations and changes. A proof-of-concept implementation of these features is available.

Furthermore, the paper describes approaches and algorithms for validating plans, as well as how the user can interact with the system to monitor plans under execution, and investigate and repair failures.

2. The Military Campaign Planning Process

The Canadian Forces Operational Planning Process (CFOPP) (SJS 2008) is a coordinated and coherent process for determining the best method of accomplishing objectives or for planning possible future tasks. It is composed of 5 stages (Initiation, Orientation, COA Development, Plan Development and Plan Review) and can be used to develop campaign plans, operations orders, contingency plans and the detailed planning of each campaign phase, branch and sequel. The CFOPP can be employed for deliberate planning as well as for rapid response planning (or crisis action planning):

- Deliberate planning consists of initiating and developing plans in anticipation of a known or anticipated future events or circumstance. It is not subject to the immediate pressures of time or prevailing threats;
- Rapid response planning consists of initiating and developing plans in response to a current or developing crisis. It requires an expeditious co-ordination and approval.

Nowadays it is often very difficult to draw a firm distinction along classic staff lines between operations and plans. Operations in the 21st century have become analogous to continuous planning, and planning and operations have grown into a single integrated process (Hales et al. 2008).

Plans can be complex, involving many tasks, objectives and constraints. Managing the execution of plans in a dynamic environment is a difficult problem. Challenges related to the dynamic management of plans involve being able to allow different people with different responsibilities at different levels of the organization to work simultaneously on a set of plans. Based on the task that they have to do (some may want to consult plans, others to refine development of plans, and yet others to monitor plans, etc). These people may have to work on the same plan at different levels of abstraction in terms of level of detail related to the organizations, activities, timing, space and purpose. In a literature survey (Hunter et al., 2007), we examined existing plan management systems and identified requirements for plan management activities. We found that most existing software to support planning is focused on efficient

plan generation. In order to manage plans effectively, information tools should also support the development of situation awareness through current, past and future plan activities in the battlefield. In fact, making informed decisions involves being aware of the status of plans that are currently being executed, as well as past and future plans. If it appears that a plan is not going to meet some objective, it may be necessary to modify this plan while considering how these changes impact on other existing plans. Accordingly, plan management involves activities of plan representation, analysis, forecasting, and monitoring, which we examined in (Hunter et al., 2007).

2.1. Plan representation

A representation of a plan is a complete, unambiguous description of that plan at the appropriate level of abstraction. The amount of detail included in this representation depends on the individual communicating the plan, as well as the intended use of the plan. For example, representations of plans must be very precise to be useful for computer-assisted plan management at the tactical level. On the other hand, a rough sketch of units overlaying a map might provide sufficient detail for strategic planning at a very high level. Plan representation involves formulating conventions for specifying all aspects of a plan, including resources, assumptions, constraints, and objectives. It also includes approaches to plan visualization.

2.2. Plan Analysis

Plan analysis refers to the process of assessing the quality of a plan. One aspect of plan analysis is to look for flaws in the plan that will make the plan fail to achieve the desired goal. It involves:

- validating the mission as to the executability of plans and attainment of the desired outcomes
- analyzing dependencies between plans; finding redundant plans and plans blocking each other
- deriving links/dependencies between plan elements at a higher level from dependencies between corresponding plan elements at a lower level
- identifying aspects that may have an impact (positive or negative) on a plan (e.g. geographical characteristics, infrastructure needed, possible threats from enemy courses of action).

For example, plan analysis may cover resource availability/status, what-if analyses, error detection, collision avoidance with other plans, and the impact of a plan on other plans.

2.3. Plan Forecasting

Plan forecasting refers to the capability to predict the outcome of a plan. A plan is typically formulated to achieve a specific goal, but changes in the environment and actions of the adversary may alter its outcome. Plan forecasting involves a method that predicts the most likely state of affairs at a specific future time point. Since plans often have uncertain outcomes, plan projection may include:

- Identifying expectations of future plan outcomes
- Predicting future situations
- Projecting the impact of unexpected events and outcomes
- Projecting the impact of plan updates
- Plan Status: Can the status of the plan be predicted at any point in time?
- Impacts of changes on future plans: If changes in the execution or outcome of a plan affect the future execution of another plan, can this dependence be identified and highlighted?

2.4. Plan Monitoring

Plan monitoring refers to the ability to observe the execution of a plan in real time. This requires physical resources such as sensors for detecting unit locations, and the ability to communicate with

active units. Plan monitoring involves obtaining this information and using it to determine any changes required to an active plan. Consequences of changes to an active plan may be significant and difficult to predict. Plan monitoring involves comparing the current state of the plan with the state that would have been predicted at the same stage prior to execution. Accordingly, plan monitoring covers:

- displaying status and performance of the plans currently running
- displaying the progress made towards attaining the active goals
- highlighting critical plans, i.e. those at risk of failing
- displaying the impact of a plan change on one level to its parent plans on a higher level
- analyzing plan failures and identifying possible causes of the failure.

3. How HTN can Support the Planning Process

HTN have existed for almost as long as STRIPS-based planners. They have been defined as early as 1975 (Tate 1975), (Sacerdoti 1975). However, the early approaches were framed more like heuristics; only in 1994 has a formal theoretical framework been published (Erol, Hendler et al. 1994), specifying syntax and semantics of HTN, and stating a sound and complete planning algorithm (Erol 1994). The goal of HTN planning is to treat planning problems as humans approach them: by decomposing them into a hierarchy of smaller and smaller subproblems, whereas the lowest-level problems have elementary solutions. By contrast, the Classical Planning approach constructs a flat sequence of actions to transform the initial state of the world into a desired goal state (Figure 1). Another goal was to bridge the gap between planning (which belongs to the realm of Artificial Intelligence) and scheduling (which belongs to the domain of Operations Research). HTN contribute to this, since a plan hierarchy specifies a partial order of tasks; a temporal interpretation is straightforward.

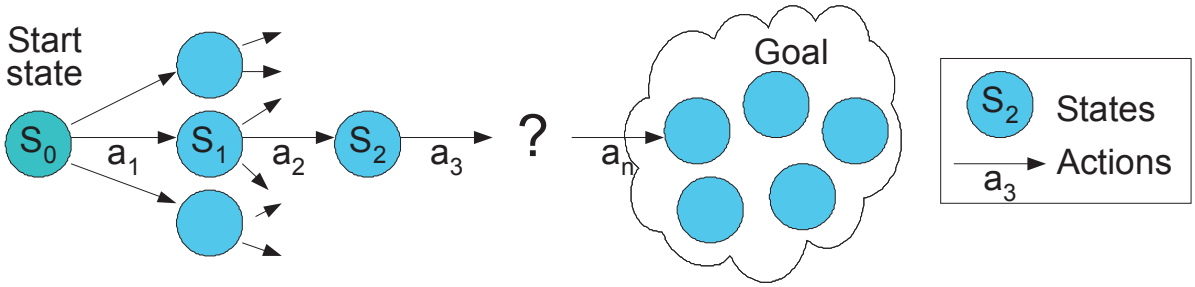

Figure 1 The Classical Planning approach

3.1. What are Hierarchical Task Networks?

This section briefly describes the theory of HTN. It largely follows (Erol, Hendler, & Nau, 1994) and (Nau, 2004), but simplifies a few aspects for the sake of brevity and clarity in this paper.

3.1.3. Tasks

The central concept in HTN is that of a *task*. Unlike its more restricted definition in the military, tasks comprise anything that needs to be accomplished, potentially from adjusting the wing angle on a plane to preserving world peace. However, the application domain will determine the highest and lowest level of abstraction. For instance, *move(resource, start, dest)* could be a task that sufficiently describes a resource moving along some unspecified trajectory from *start* to *dest*. In other contexts (e.g. in a tactical surveillance operation over an area of interest) this task may have to be subdivided along a number of waypoints into subtasks for moving along individual straight or curved segments.

As the example in the previous paragraph shows, tasks have parameters specifying the resource

executing the task, the target(s), locations, and any other relevant objects. (It is usually understood that each parameter has a specific domain defining what type(s) of objects it can stand for .) To allow general, reusable rules for task decomposition, variables can be used in place of these parameters. In the description below, we use $x_1, \ldots, x_n$ for variables and $r_1, \ldots, r_n$ for parameters that are either variables or instantiating objects. Hence, tasks are formally written as expressions of the form $t(r_1, \ldots, r_n)$.

Tasks at the lowest level of abstraction are called *primitive*. An *operator* is a rule that specifies when and how a primitive task can be executed, and how it changes the current state. An *action* is a ground instance of an operator, i.e. the execution of the primitive task using a particular resource in a particular location etc. Actions and operators constitute the atomic level of state transitions, in that they have no distinguishable internal structure.

By contrast, non-primitive tasks need to be decomposed. The rules that describe when a task can be decomposed, and into what subtasks in what order, are called (decomposition) *methods*. The expressive power of HTN lies in the way a non-primitive task can be broken into a hierarchy, down to primitive tasks, which can then be executed by instantiating actions. The remaining sections will define this formally.

3.1.1 States

A *state* is a set of ground atoms (predicates), describing all relevant static facts, and all dynamic facts holding at a certain time. This includes not just the environment, but also the location and status of own resources, information about the enemy, etc. HTN planning usually makes the closed-world assumption, that is, if a state does not contain the predicate p, then $\neg p$ (not p) is assumed to be true in that state.

3.1.2. Operators and Actions

An *operator* is defined by:

- a unique name $\text{oper}(x_1, \ldots, x_n)$, whereas ($x_1$, ..., x_n), lists all the variables occurring as parameters in any of the other elements below, which need to be instantiated. For instance, *fly(resource, start, dest)* could be an operator that can be instantiated by a variety of resources flying from any given start location to any given destination.
- a primitive task $t(r_1, \ldots, r_m)$. The operator describes one possible way of executing this task.
- a set of *preconditions* $p(r'_1, \ldots, r'_l)$, which are predicates that must be true in order to apply this operator in the current state. For instance, *at(resource, start)* is one of the preconditions for *fly(resource, start, dest)*.
- A set of *effects* $q(r''_1, \ldots, r''_k)$, which are predicates that hold after the operator has been applied. For instance, *at(resource, dest)* is one of the effects of *fly(resource, start, dest)*. Similarly to Classical Planning, applying an operator can be regarded as a transition from a state containing all preconditions, among other predicates, to a state in which all the specified effects are attained but that is otherwise unchanged.

Actions are ground (fully instantiated) operators. As such, they correspond to actions in Classical Planning. Since deciding on resource assignments, locations, routes etc. is part of the planning process, ultimately all parameters are ground, and the end result, an executable plan, is a sequence of actions.

3.1.4. Constraints

A *constraint* in some way restricts the execution order of tasks. One can distinguish the following types of constraints (see (Nau, 2004) for a formal definition):

- *precedence constraint*: specifying that a task must be executed before another;

- *before-constraint*: specifying that a predicate must be true when a task starts executing;
- *after-constraint*: specifying that a predicate must be true when a task finishes executing;
- *between-constraint*: specifying that a predicate must be true between the end of a task and the start of another task.

3.1.5. Task Networks

A *task network* is a tuple (U,C) consisting of a set U of tasks and a set C of constraints. The network is called *primitive* if all its tasks are primitive. The root node (or highest level) of a planning problem is also a (usually non-primitive) task network; the objective of planning is to expand (or decompose) it into a primitive task network, and then find a plan (a partially ordered set of actions) that "satisfies" it.

3.1.6. Methods

Methods state how a task can be decomposed into lower-level (sub-)tasks. A method specifies:

- a non-primitive task $t(r_1, \ldots, r_n)$, which we call the *head task* of the method. For instance, *transfer(resource, payload, start, dest)* could be a task that can be instantiated by different resources carrying different types of loads (or even people) from a specific start location to a specific destination.
- a network of subtasks. For instance, a suitable task network breaking down *transfer(resource, payload, start, dest)* has three subtasks: *load(resource, payload)*, *fly(resource, start, dest)*, and *unload(resource, payload)*, and precedence constraints between the first and second, and between the second and third subtask, to ensure the correct temporal order of execution.

The same non-primitive task can occur in multiple methods. This introduces nondeterminism, as there may be more than one way to decompose a given instance of a task. To speed up the planning process, one can specify *preferences* between methods.

In the context of planning, each method defines a *reduction schema*, or a recipe for decomposing a given task network. The idea is as follows: Given a task network (U,C) containing a non-primitive task t and a method $m = (t,(U',C'))$:

- Remove t from U.
- Merge (U',C') into the given task network to obtain a new task network (U'',C''). This operation involves slightly more than taking the set union: namely, a unifying substitution must be applied to the variables, and the constraints in C referring to t must be propagated into suitable constraints for the tasks in U'.

If we apply this process until there are no more non-primitive tasks, we get a primitive task network, from which a plan can then be obtained.

Example. Assume the following tasks are defined:

- *rescue(personnel, loc)*— rescue some personnel stranded at location loc
- move(res, loc1, loc2) — move a resource res from loc1 to loc2
- attach(res1, res2) — attach (load) a resource res2 to a resource res1.

The latter two can be executed by elementary actions, whereas the first task must be decomposed. For instance, the following operator might be used to execute *move(res, loc1, loc2)*, in case *res* is an aerial resource:

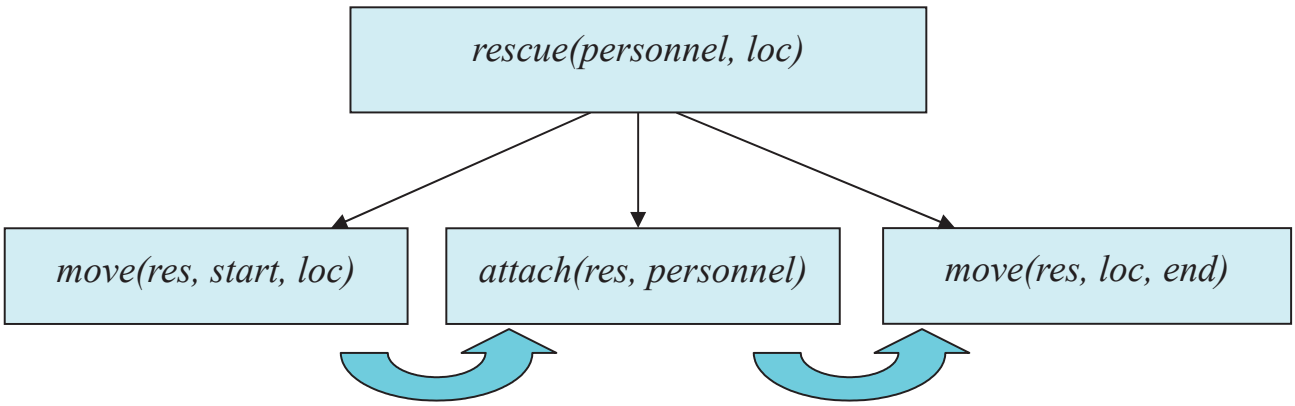

Figure 2 A Hierarchical Task Network Decomposing a Rescue Task

Operator: *fly(res, from, to):*

- Task: *move(res, from, to)*
- Precondition: *is_at(res, from)* at start of the executing action
- Effect: *is_at(res, to)*.

Likewise, *attach(res1, res2)* might be decomposed thus, provided the resource to be attached are people:

Operator: *pick_up(res, person, at):*

- Task: *attach(res, person)*
- Precondition: *is_at(res, at)* at start of the executing action
- Precondition: *is_at(person, at)* at start of the executing action
- Effect: *is_attached(res, person)*.

The following, method for *rescue(personnel, loc)* shows how tasks are decomposed into more elementary tasks:

Method: *airlift(res, personnel, start, loc, end)*:

- Head Task: *rescue(personnel, loc)*
- Subtasks:
  1. move(res, start, loc)
  2. attach(res, personnel)
  3. move(res, loc, end)
- Precedence constraints:
  - 1. ends before 2. starts
  - 2. ends before 3. starts

The task network defined by this method is shown schematically in Figure 2.

3.2. Why are HTN superior to Classical Planning?

HTN models an iterative planning process, which breaks down a high-level planning task with incremental level of detail, with the ultimate output being a primitive plan network, whose lowest level has elementary, executable actions. Classical reasoning with tasks and actions separated planning (decomposing goals, creating actions and tasks) and scheduling (allocating tasks to resources and time slots, obeying all constraints). HTN-based planning combines planning and scheduling: it selects tasks and actions, and verifies constraints as they occur. It has been shown to solve planning problems much more efficiently than classical planners (Nau, 2005).

In HTN, specifying the decomposition methods to break down tasks becomes a conceptually separate activity from actually breaking down tasks into a plan hierarchy. In fact, it requires the skills of a knowledge expert, and great care must be taken to take into account as much as possible all

probable situations and contexts. However, this initial investment pays off as the same task is encountered again, possibly in slightly different situations with different parameter instances, as the same methods can be re-used.

In the planning process, branch plans and alternative courses of action can be modelled and stored as part of the same plan. Different environment conditions simply "activate" different branches of the plan. Provided that several alternative decomposition methods have been developed for a task, it is also easy to locally repair a plan if one of its currently executing tasks is likely to fail: simply decompose the task using a different method.

The HTN planning approach is scalable (Ambite 2003): an additional level of detail can be easily added. By contrast, classical planning suffers serious performance drops as the level of detail increases.

HTN can be extended to handle the following critical requirements for dynamic plan management in a military setting:

- Modelling and reasoning about resources
- Representing and reasoning with time
- Planning at different levels of abstraction
- Conditional outcomes of actions
- Uncertain outcomes of actions
- Exogenous events
- Incremental plan development
- Dynamic real-time replanning (Muñoz-Avila 1996)
- Dynamic execution monitoring.

3.3. Hierarchical Goal Analysis

Hierarchical Goal Analysis (HGA) (Hendy et al. 2002) is a relatively new approach for mission planning. It has been used in such diverse applications as multiple UAV control (Kobierski 2004), piloting a decision-making architecture on a modernized Halifax-class frigate (Chow 2006), autonomous distributed computing for a network of microprocessors onboard a spacecraft (Hartmann 2004), and operational plan execution management for the 2010 Olympics (Hunter et al., 2008). Instead of a task decomposition structure, it proposes a *goal decomposition structure*, in which goals form a multi-level hierarchy. This introduces the desirable aspect of goals as something one can systematically work towards attaining. At the lowest level, goals can be handed over to a human operator or to a machine, who will execute a suitable plan to accomplish the given subgoal.

The advantage of goal-oriented mission planning over task-oriented planning in a military setting is that it forces the operators to think in terms of reaching desired outcomes, rather than fulfilling a scheduled set of chores. Secondly, the mission decomposition structure tends to be leaner when using goals, rather than tasks. Finally, mission planners can not only traverse the goal hierarchy top-down (by decomposing a higher-level goal). But they can also analyze it bottom-up (by propagating the desired outcomes represented by the subgoals into higher-level goals); in experiments, this enabled them to find gaps in the goal decomposition process that would have critically affected mission success.

3.4. Proposed Approach

A major contribution of our work is that it combines goal decomposition and HTN-based task decomposition into a new approach that combines plans and goals into *one* hierarchy. Every node in the hierarchy decomposes either a plan or a goal. Hence both goals and plans can appear at any level; however, (Kobierski 2004) shows that higher levels of abstraction in a real mission tend to have—and decompose—goals, and lower levels tend to predominantly feature plans.)

3.4.1. Development of a Plan Hierarchy from a Task Network

Given a task that has been decomposed into a task network using HTN decomposition, we derive a *plan hierarchy*, or *plan decomposition structure*. We simply map each task into a plan consisting of the partially ordered subset of all actions accomplishing any of its primitive tasks. By extension, we say that the plan *accomplishes* the task that is mapped to it. The hierarchy results from the plan-subplan relationship, which directly corresponds to the task-subtask relationship in the task networks. Since a subplan can have more than one parent plan, the Plan Decomposition Structure is a Directed Acyclic Graph (DAG). The level of a plan in this hierarchy defines its *level of abstraction*. The leaf nodes in this hierarchy include the plans corresponding to the primitive tasks, which are singleton sets consisting of the actions accomplishing these tasks. But unless the task network is primitive, it will contain tasks that have not been decomposed; we map these to what we call *unspecified plans*. This is desirable in a military setting with an incremental, distributed planning process. A commander can plan a mission down to a certain level (e.g. operational), without specifying all the details on a lower (e.g. tactical) level. At that level, the unspecified plans are "actions at the operational level of abstraction". Without any further planning, these plans will fail to execute, since they correspond to tasks for which no decomposition method has been chosen yet. Later on, as the mission execution draws nearer and more knowledge about the actual situation becomes known, commanders at a lower level can decompose these tasks further and create a hierarchy of subplans and lower-level (tactical) actions, so as to make the operational-level plans actually executable. In fact, planning at the lowest levels (e.g. trajectory planning, collision avoidance) could be delegated to autonomous planners or controllers. This low-level plan development is completely transparent to the high-level operators who are only concerned with a "view" of the plan hierarchy down to their own level of abstraction, and that this plan will succeed.

3.4.2. Scheduling, Plan Execution and Monitoring

A plan can only be scheduled, decomposed and executed if:

- it has been assigned a decomposition method (see Section 3.1.6);
- all variable parameters used in the method are instantiated;
- it has at least one subplan implementing each of the subtasks mentioned in the method.

Once a plan is assigned a decomposition method, then a subplan must be specified, instantiating each of the method's subtasks. A subtask can be instantiated by more than one subplan, which provides contingencies or redundancies a commander may deem necessary. A user monitoring a plan can also cancel or unschedule a plan (that has not finished executing) when needed.

A plan can have the following *status* values:

NOT_STARTED	not yet started
RUNNING	started but not yet completed, and a subplan is running or waiting
WAITING	running but currently no subplan is running or waiting
BLOCKED	cannot (continue to) execute because a constraint is violated
CANCELLED	would be running but prematurely cancelled
UNSCHEDULED	would be running but has not been scheduled, or unscheduled before start
SUCCEEDED	finished successfully
FAILED	past scheduled end time; did not finish or did not produce desired effect(s)

The interval defining the start and end time of an *action* is explicitly specified. For a composite plan, however, a default interval can be determined as the smallest interval containing all *scheduled* subplans'

time intervals. But the operator could assign an earli er start time and/or a later end time, e.g. to allow for briefing and debriefing tim es at the beginning and end of a pl an. For unspecified plans, an approximate start and end time must be given.

Example. Continuing the exam ple in Section 3.1.6, the following provide s an executable plan for the ground instantiation *rescue(stranded_crew, crash_site)* of the *rescue* task, using the instance *airlift(rescue_heli, stranded_crew, start, loc, end)* of the *airlift* method. It obviously satisfies the precedence constraints and is execu table, provided that reso urce *heli* is located in *Nitric* at 11:15 and that the people to be rescued are at the crash site:

Plan: RescuePlan

Start time 11:15, End time 12:45

Subplans:

- *fly(heli, nitric, crash_site)*
  Start time 11:15, End time 12:00
- *pick_up(heli, stranded_crew, crash_site)*
  Start time 12:00, End time 12:15
- *fly(heli, crash_site, wahhabe)*
  Start time 12:15, End time 12:45

An execution schedule for this plan is shown in Figure 2.

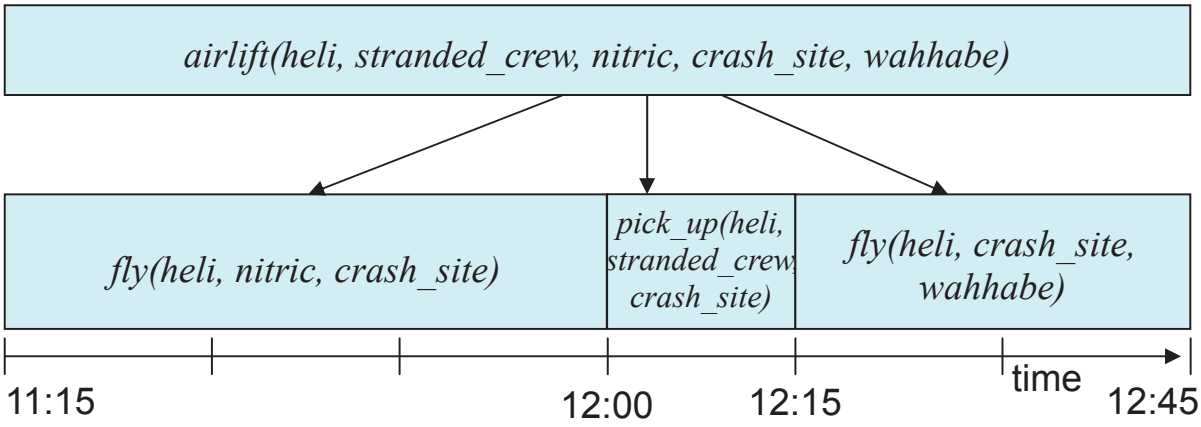

Figure 3 Example of a decomposed Plan scheduled for Execution

3.4.3. Adding goals to the plan hierarchy

In classical planning, goals constitute an explicit set of desirable states of the world. HTN problems do not specify goals explicitly, but we can equivalently define goals as predicates that should hold in the end state brought about by the successful execution of plans. Hence we only need to check whether the given goals are achieved or not, and if not, define tasks that will accomplish the goals and create plans for them using the HTN technique as before.

As we have seen, the HGA approach proposes the idea of a Goal Decomposition Structure, in which each goal can be d ecomposed into several sub goals. Again, a goal can be us ed more than on ce as a subgoal, so the goal decom position hierarchy is also a d irected acyclic graph. Goals specify a level of abstraction corresponding to the level at which the world is viewed. For instance, a goal of maintaining the stability of national governm ents is very high-leve l, whereas a goal of safely flying an a ircraft through fog is low-level. Just as in the plan deco mposition structure, the level of abstraction must strictly decrease from goals to subgoals.

Alternatively, goals can also be assigned a plan that will be executed to attain these goals. This provides a link between goal decomposition and plan decomposition. We require that a goal can only be assigned one plan, but that a plan can be assigned to satisfy multiple goals.

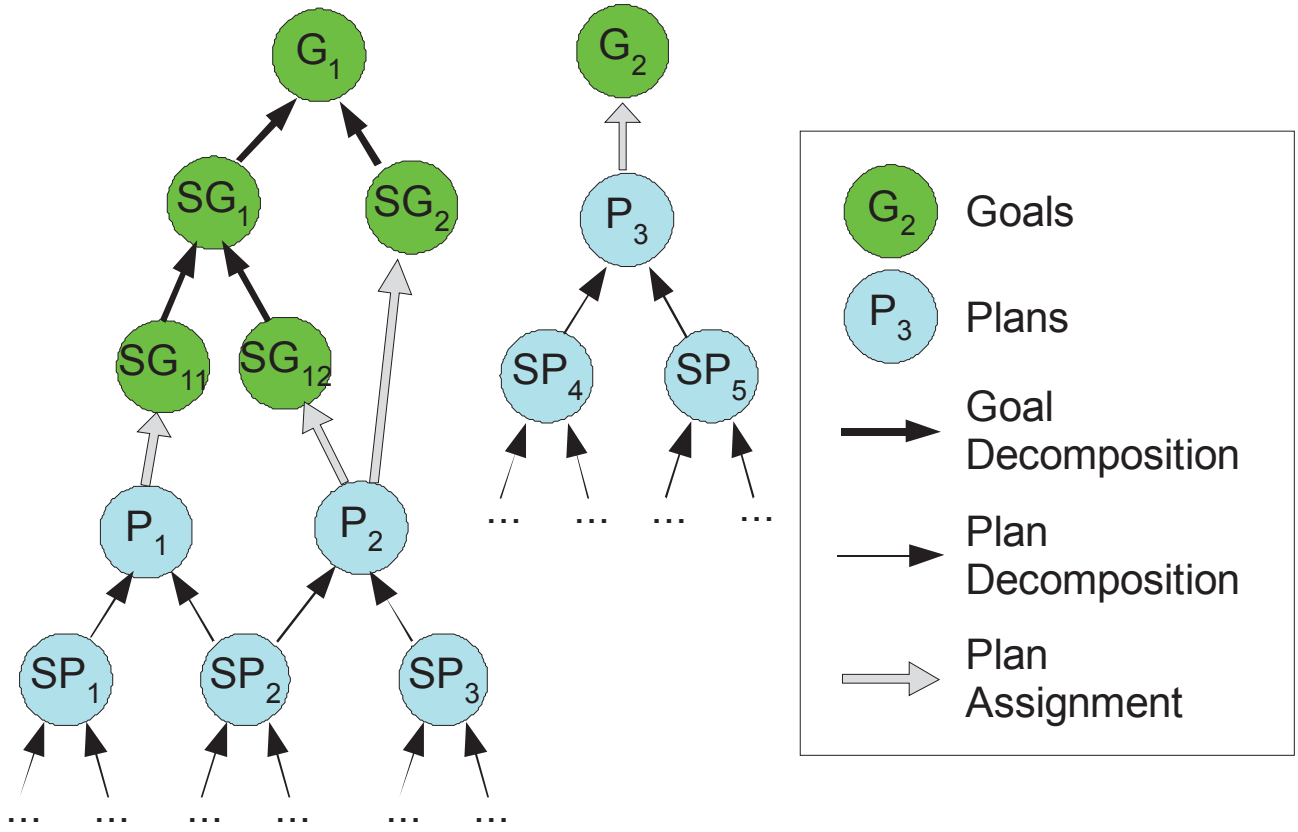

Figure 4 Example of the mixed Goal/Plan Decomposition Hierarchy

The start and end time of a goal are not explicitly set. Instead, the "execution interval" of a goal is defined as the smallest interval containing the execution intervals of all plans assigned to the goal or to any of the subgoals in the hierarchy below it. The execution interval of a goal can be used to define a timeout to monitor the accomplishment of this goal.

The *status* of a goal can take the following values:

NOT_STARTED	not yet started
RUNNING	started but not yet accomplished, and a subgoal or the assigned plan is executing
WAITING	started but not yet accomplished, and no subgoal nor the assigned plan is executing
ACCOMPLISHED	Goal condition has been attained
FAILED	Goal is past its end time, and condition has not (yet) been attained

There are many ways to define what constitutes accomplishment of a goal. One could require that the subgoals have to be accomplished simultaneously, or each one just at some point during the execution interval. Also, one must specify whether a goal can be accomplished after its end time. The approach chosen here is as follows:

A goal is monitored beginning at its start time. If at any time the goal condition is satisfied on its own (e.g. through an unrelated plan satisfying it as a side effect), the goal is considered accomplished

at that time. This holds also for unspecified goals. To keep the goal hierarchy coherent, once a *specified* goal is accomplished, it remains so. If a plan is assigned to an unaccomplished goal, it must be evaluated at the plan's end time. If the goal is still not accomplished, it is considered failed then. However, it can be accomplished later on, due to a change in the goal condition or because all its subgoals have been attained. This situation is very useful, as it will require no actions to be executed.

4. Prototype System

4.1. Architecture

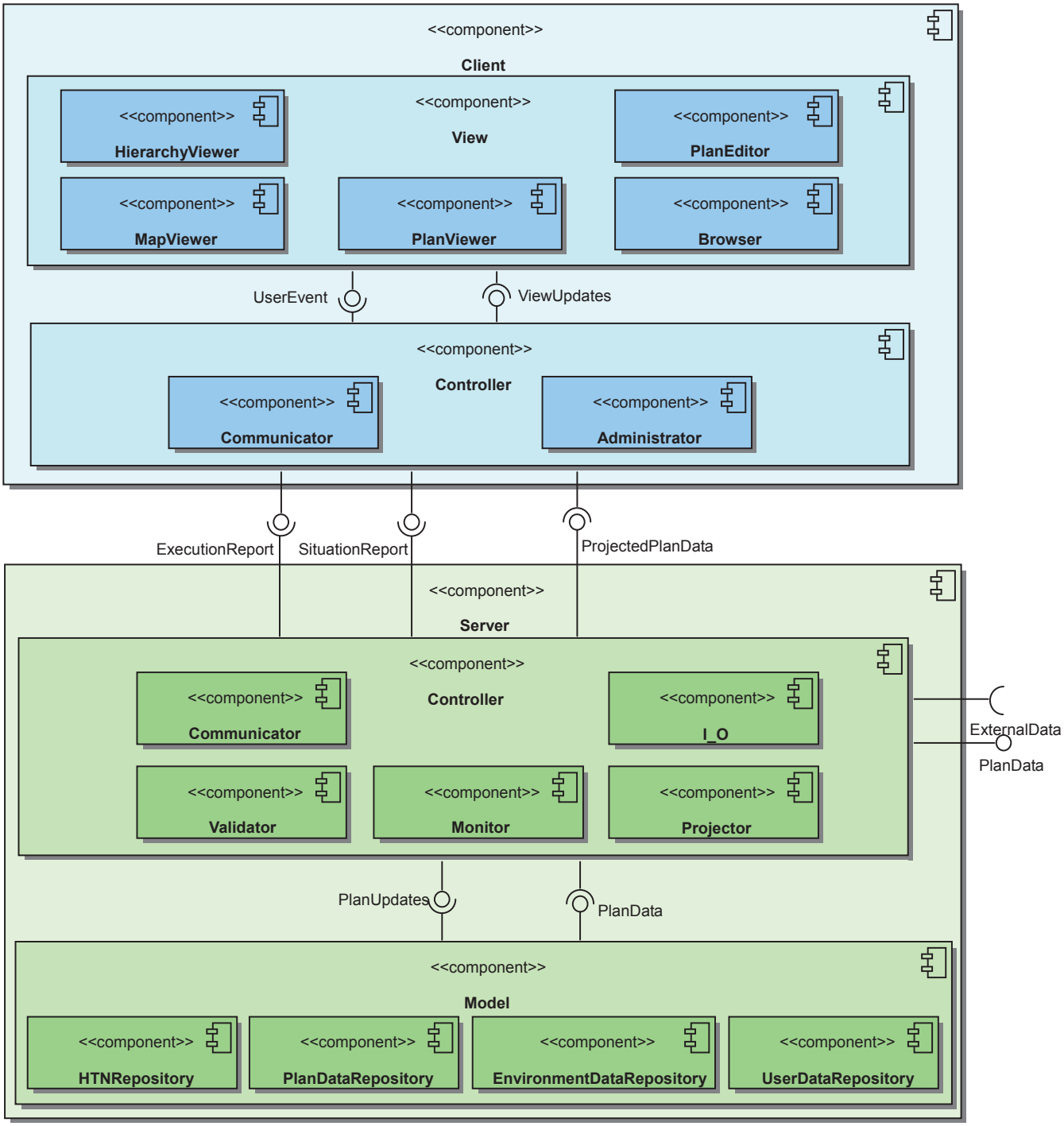

Figure 5 A UML Component Diagram of the Multi-plan Management System

The architecture of our prototype system, shown in Figure 5, resembles that of other distributed plan management tools, such as COPlanS (COPlanS 2007) However, as we will see, it uniquely accommodates the HTN-based plan representation and plan management capabilities described earlier. It is structured along a client-server design and implements a 3-tier Model/View/Controller paradigm. The client side provides a user front end with its views of plans and plan elements. The server side is responsible for storing and updating the plan structure (model). Controller functionality is found on both server and client; hence the middle tier is split in two. Among other things, it provides the communication between clients and server. In detail, the tiers comprise the following components:

View (client-side):

- *Hierarchy viewer, Map viewer, Chart viewer, Browser, Editors*: as described below.

Controller (client-side):

- *Administrator module*: allows user login and overall session management
- *Communicator module*: transmits individual users' updates as situation and plan execution reports, and receives collective updates and projections from the server.

Controller (server-side):

- *Validator:* validates new and updated plan elements for consistency.
- *Projector*: projects the status of plans and changes in the environment to forecast plan outcomes and goal attainment
- *Monitor*: periodically monitors the situation and its predicates, as well as changes in plan status at the current time
- *Communicator*: receives situation and execution reports from clients, and transmits collective updates and plan projections back to the clients
- *Server I/O*: reads plan and situation updates from external systems, databases and services.

Model (server-side):

- *HTN Repository*: all predicates, rules, tasks and decomposition methods
- *Plan Repository*: all plan and goal data
- *Environment Data Repository*: all defined predicates and their histories over time
- *User Data Repository*: data related to the users of the system, their login information, access privileges, and responsibilities (level of abstraction).

4.2. Graphical User Interface

The main view (Figure 6) provides a concise but informative picture of the current situation, thus facilitating real-time monitoring of plans as they unfold. It is shown at the initialization of the program (after the user has logged in). In accordance with its purpose, it initially shows only the highest-level information appropriate for the operator's level of abstraction. For instance, a strategic commander would see a world map with all strategic plans, an operational commander would see their theatre of operation, and a squadron leader might see the squadron members and the tactical plans assigned to them. In general, a higher-level view is displayed as a summary of the union of the information from the lower levels. Various capabilities for drill-down access to underlying levels exist, as listed below.

The goal and plan hierarchy is shown on the left-hand side of the screen. It forms the starting point for navigating through goals, plans and plan elements. Therefore, it is always visible, and all other

views are aligned with the goals and plans shown in this view. The user can elect to display goals only, or plans only, or both. In the following description, we will usually refer to the elements in the tree as "goals", but mean to include plans also.

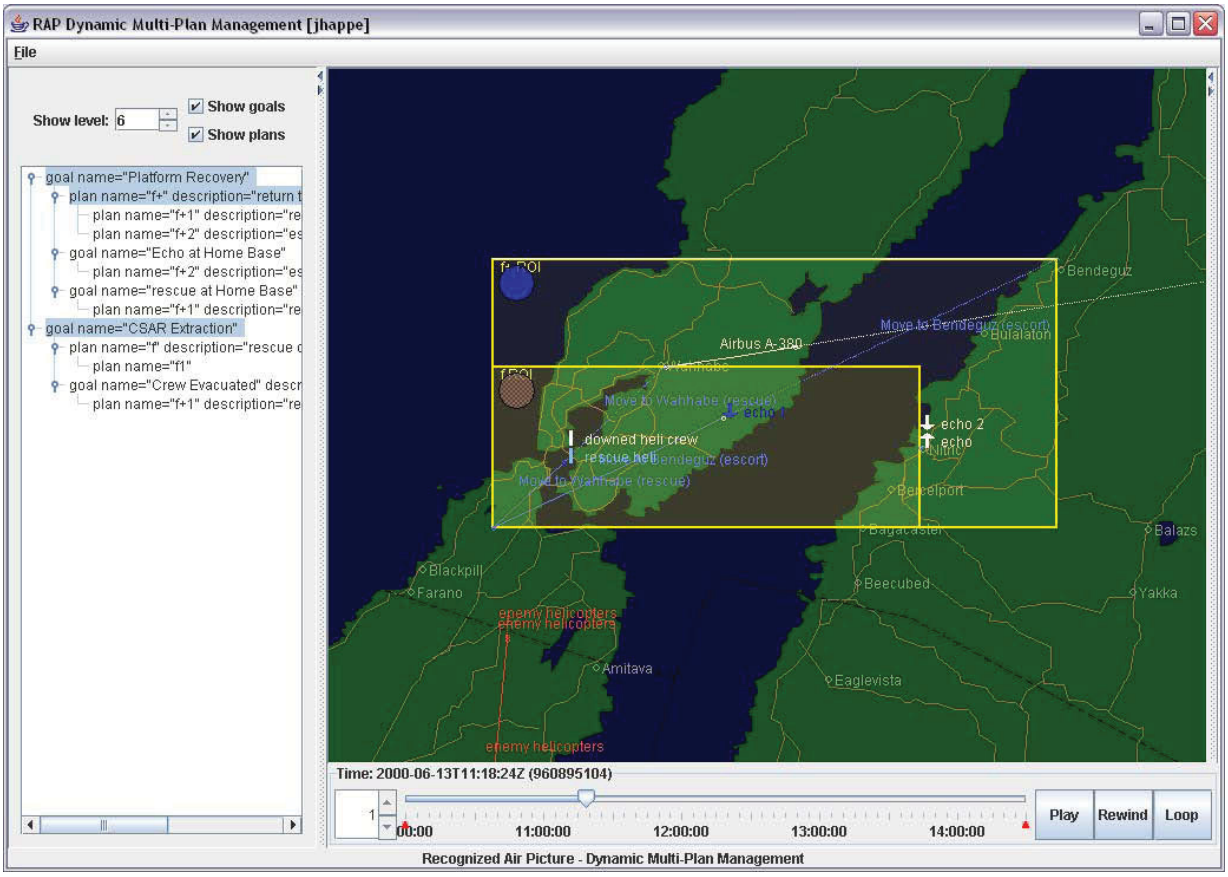

Figure 6 Initial Main View and Goal Map View

By selecting a goal per mouse click, the user can show related information in the current view on the Right-Hand Side (RHS) of the screen. It is possible to select multiple goals and show comparative or aggregate information. This allows comparing goals or plans, or analyzing overlaps and dependencies between plans. Any combination of goals, including goals at different levels of abstraction, and goals with their subgoals, can be selected. There is no limit on the number of goals that can be selected.

Time Slider

A common element in many of the views is the time slider at the bottom of the right-hand side panel. It encompasses the mission horizon; the slider knob indicates the current time. By dragging the knob back in time, the state of all plans and goals in the past can be shown. Conversely, dragging the knob into the future shows the projected state of the plans and goals, based on current assumptions and estimates. The red markers on the time slider indicate the periods of interest (POI). To the right of the slider, there are buttons for playing/pausing, rewinding, and looping through an animation.

Using colours to indicate plan and goal status

Many of the views use colours to visualize the execution status of plans and goals. The different goal status values are as follows:

- Green, for a goal that has finished successfully, a current goal expected to finish successfully, or a

goal that is projected to execute successfully at its scheduled time in the future;

- Red for a finished goal that has failed, for a current goal expected to fail, or for a goal planned to be executed in the future that cannot be executed successfully without plan repair;
- Yellow for a goal that is still predicted to finish successfully, but for which there is a change of tendency, which renders it possible that the goal may fail without intervention.
- Orange for a goal that is predicted to fail, (in missions being executed and those planned to be executed in the future) but for which there is a change of tendency, which renders it possible that the goal may succeed even without intervention.

Goal Map View

The system displays a map view at the top level of visualization (Figure 6). Initially, this view just shows the background GIS, environment information (neutral, allied and enemy tracks), and a time slider. The region and time interval shown correspond to the specific operator's ROI and POI.

As the user clicks on goals and plans in the Plan and Goal hierarchy, each goal and plan is shown on the map view by the region of interest covered, each with a coloured circle or disc corresponding to its current or forecast execution status. Furthermore, each goal's ROI is shown as a rectangle with a solid border and a light semi-transparent fill colour.

Right-clicking on a goal's region or disc will bring up a context menu similar to that in the goal and plan hierarchy view, with the possibility to expand this goal (show subgoals and -plans), edit the goal, delete it, or switch the view.

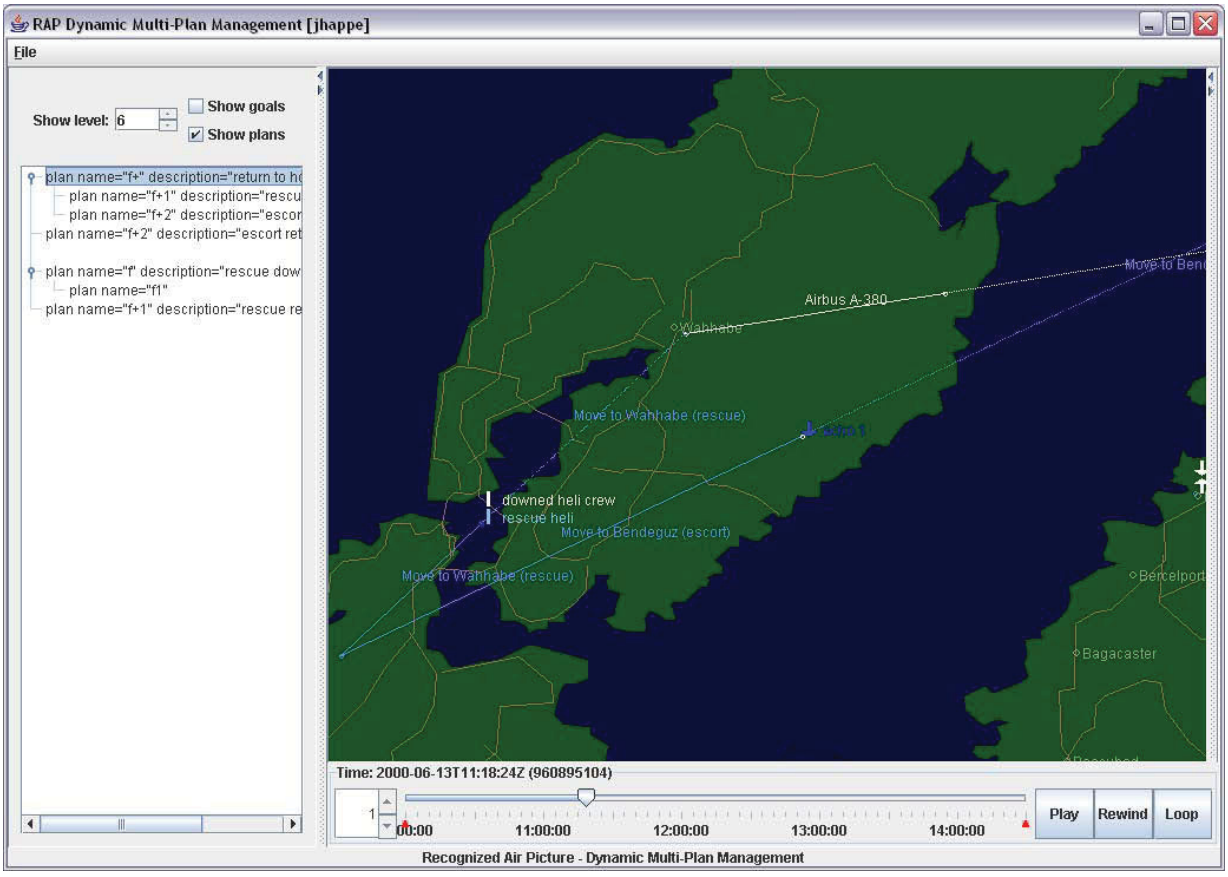

Figure 7 The Resource Map View shows Resources and Trajectories

Resource Map View

The resource map view is really the same as the Goal Map view: it shares the same functionality, and it also displays a GIS map with environmental information. However, the foreground is different, showing not goals and plans but rather the resources allocated to them, along with their trajectories (see Figure 7). The resources are shown at the level of abstraction that best corresponds to the level of abstraction of the plan. For instance, if an operational plan allocates a fleet of vessels, the fleet is shown as one resource. The context menu allows the user to expand this resource and show lower-level resources, e.g. the individual vessels. (Alternatively, the user can select subplans in the Plan and Goal Hierarchy, of course.) Upon dragging the time slider or playing an animation, the display shows the position of the resources as they move along their trajectories.

Schedule View

The schedule view displays a Gantt chart of tasks and resource allocations. The left-hand side of this chart shows the selected goal(s) along with its allocated resources, performance measures and subgoals/subplans. Each of these elements takes one row. For instance, Figure 8 shows the schedule view shown upon selecting the action *b-2*. If this was the only plan selected in the goal and plan hierarchy, only *b-2* is shown in the Schedule View as well. For the action *b-2*, one resource, *echo 2*, is allocated and the measure of performance, *cost*, is used. Being an action, *b-2* has no subplans.

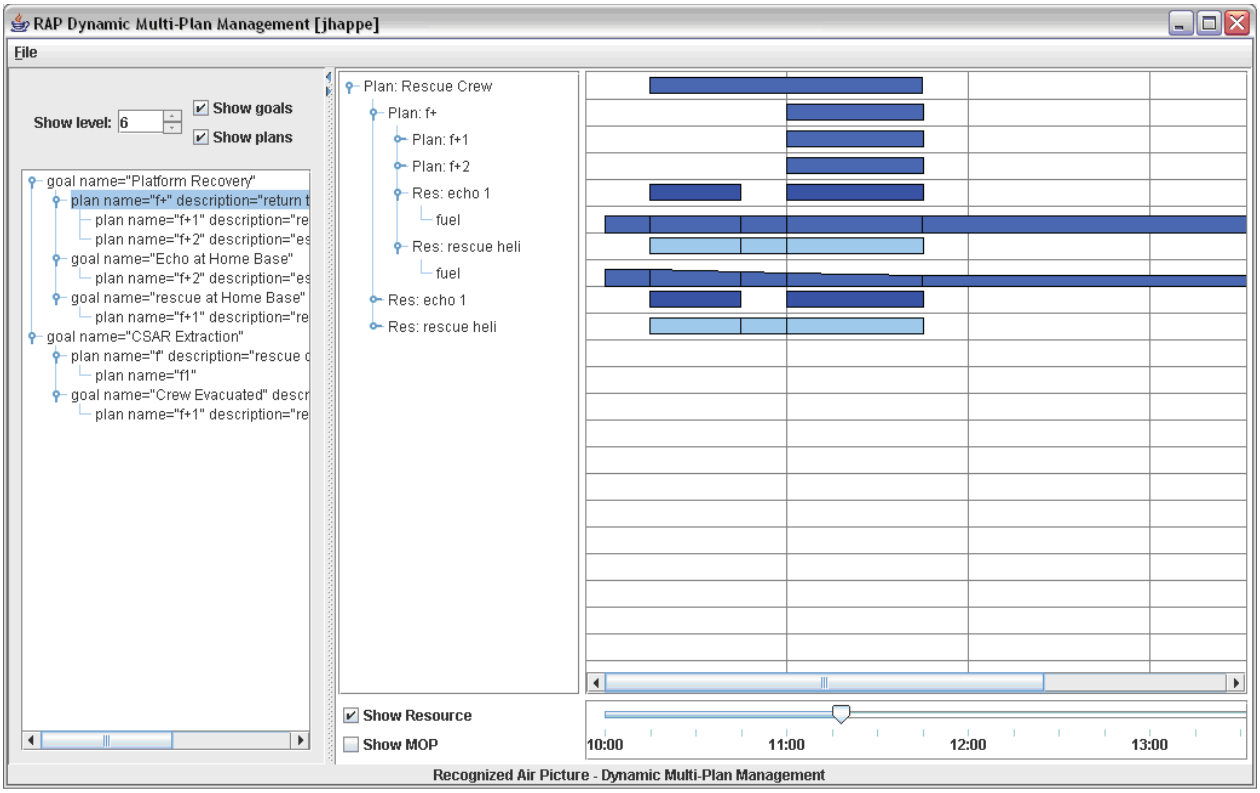

Figure 8 The Schedule View Displays Plans and their Resources

The right-hand side shows a schedule or graph for each row, depending on the item in that row:

- The blue bars in the rows containing a plan display the periods of time during which each plan is executed. For an action, this is simply the interval from the action's start time to its end time. For a composite plan, it is determined as the union of the time intervals for all component subplans.
- Allocatable resources (units) are displayed with a coloured bar indicating their status. The light green colour shows that the resource is available and not assigned to any plan; a blue colour shows that the resource is scheduled for this particular plan; a dark gray colour indicates that the resource

is scheduled for this action but could not be allocated because of a resource conflict, whereas a light gray colour shows that the resource is unavailable for assignment to any action.

- For consumable resources (materiel), an area graph is displayed, showing the projected amount remaining (in black) and the actual amount remaining. Thus the user can determine whether the resource is consumed at the predicted rate.
- Similar area graphs are used for the measures of performance: their predicted value and their actual value over time.
- By clicking on a row containing a (sub-)goal, the user can show details of this (sub-)goal, which are inserted below the parent goal. Thus it is possible to select any level of subgoals and -plans for a goal and have the information displayed in the schedule view. (The view will ensure that subplans that have already been shown will only be shown once.) Likewise, the subresources of a resource can be shown by clicking on the resource's row. This allows the user to drill down along these two different hierarchies.
- Higher-level resources are displayed by showing aggregate information of their subresources.

Browser View

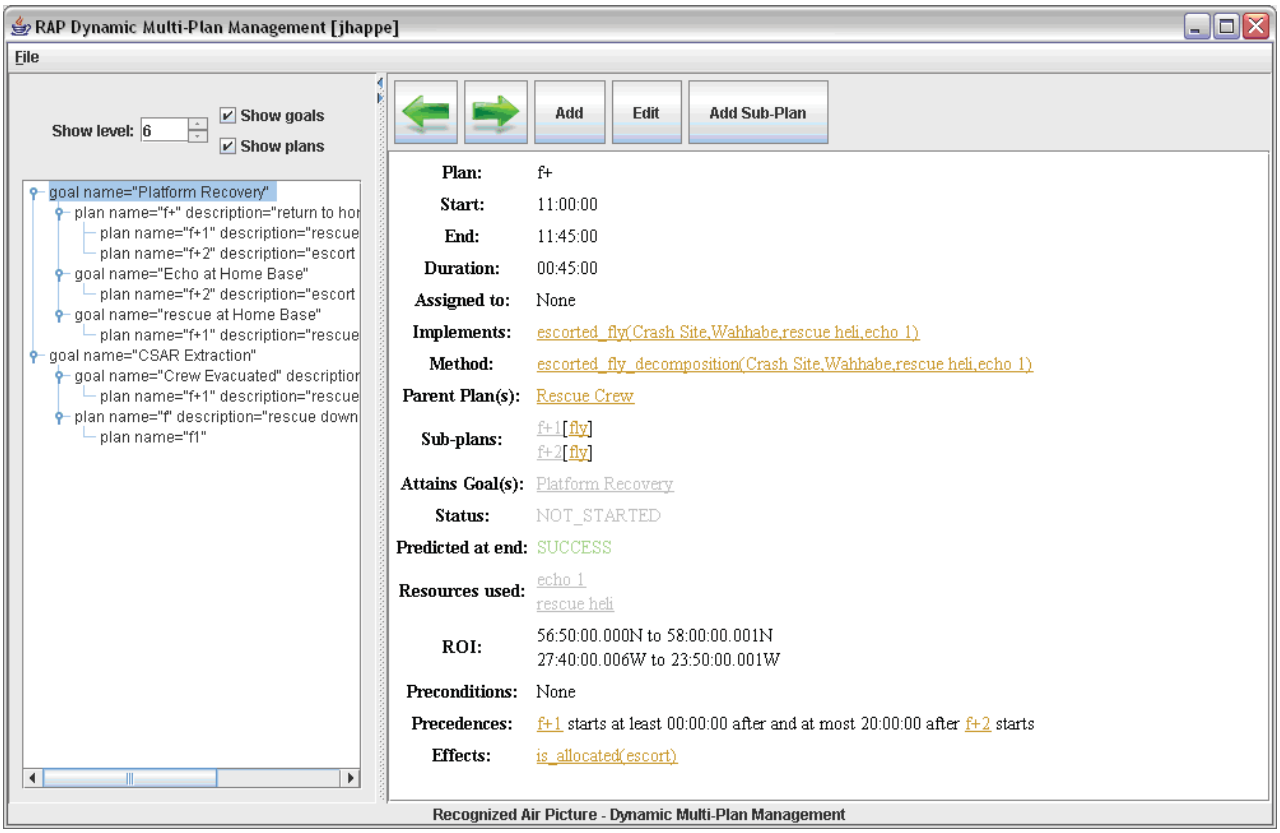

Figure 9 The Browser View, Showing a Plan and its HTN Task and Method

The browser view allows viewing plan elements with all their attributes. Any attributes that are themselves plan elements—such as subgoals, plans, resources, tasks—can be clicked upon, which updates the browser view to show this plan element. Thus it is possible to browse all information in a number of hierarchies, notably the goal, plan, and resource hierarchies. Unlike the other views, the browser view allows following the links between elements "across" the different hierarchies. Finally, this is the view in which the HTN task of a plan can be viewed.

Figure 9 shows a plan in the browser view. It reveals that this plan instantiates the task

"escorted_fly" and that the hierarchical task network underlying the plan is given by the method "escorted_fly_decomposition". One can also see that the two subplans of f+ each instantiate one of the "fly" subtasks. The complete browser view includes other information about goals and plans as well, such as the current and the predicted plan status. Any highlighted (underlined) plan element is a hyperlink, which provides the mentioned navigation capabilities. The view has standard browser features such as a back and forward button to aid navigation; a history and bookmarks list could also be implemented.

4.3. Plan adaptation

The system provides a simple functionality to edit elements of a plan. All plan and situation elements can be edited by right-clicking on the desired element—whether in the hierarchy, map, schedule or browser view. Whenever a suitable element is selected, the context menu will show an option to edit it. Thus, the user can edit a plan element "on the spot", for instance to repair a resource conflict or adjust a plan whose execution is running late.

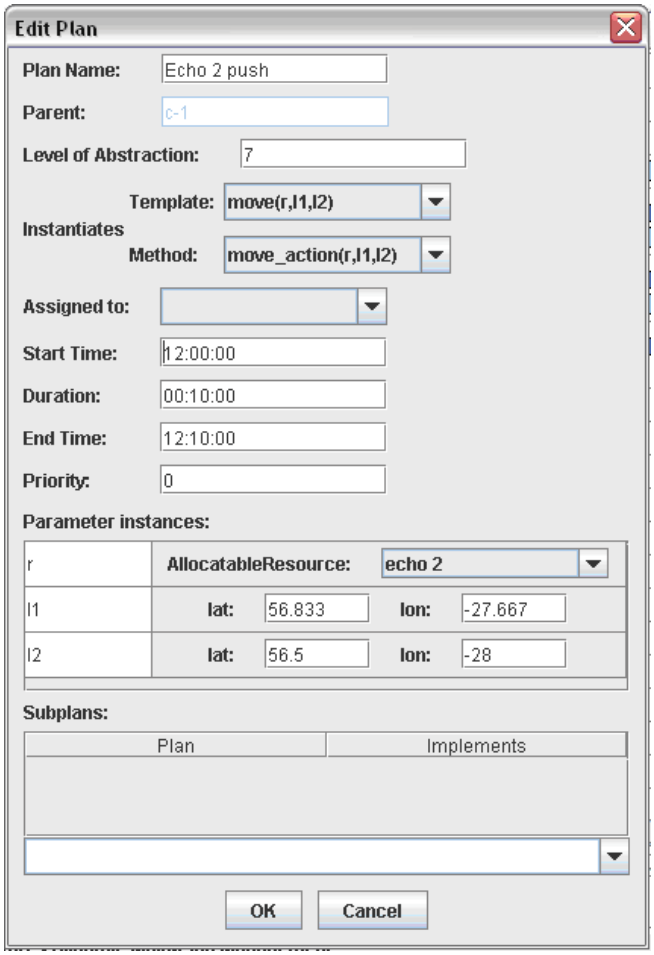

Figure 10 Editing a Plan

4.4. Plan failure analysis in the CSAR vignette

The most important aspect of plan analysis is analyzing the causes of plan failures, so a targeted effort can be made to repair the plans. We distinguish several types of plan failures:

- *Plan not expandable*: the plan is unspecified, i.e. it does not yet implement any method

- *Plan inconsistent*: the plan hierarchy is invalid in some way. For instance, a plan might be a subplan of another plan at the same or lower level of abstraction
- *Plan incompletely expanded*: the plan instantiates a method that has subtasks which are not yet instantiated by subplans. To repair this failure, the commander must specify these subplans (or delegate this task to a commander at the subtasks' level of abstraction)
- *Plan cannot execute*: usually because a constraint is violated or because the parent plan is blocked
- *Plan unsuccessful*: The plan executes, but one or more of the anticipated goals are not (expected to be) attained

We obtain a notion of the *status* of a plan recursively from the status of its failure-free subplans (not started, executing, successfully completed)..

For goals, we have the following reasons for failure:

- *Goal not specified*: no plan has been assigned to this goal or any of its subgoals
- *Goal inconsistent*: the goal hierarchy is invalid in some way. For instance, a subgoal might have a predicate at the same or higher level of abstraction than the goal itself
- *Goal incompletely expanded*: this goal instantiates a method with subtasks which are not yet specified. Repair of this failure is analogous to tasks
- *Goal not attained*: Some expectation involving the goal itself or some subgoal fails to be true

5. Conclusion and Discussion

This paper reports on an underlying theory for planning and plan representation and on a new approach and a human-computer interface for managing of multiple distributed plans at multiple levels of abstraction. At this point, it was not our goal to produce a full-fledged operational distributed plan management system, but rather to investigate how the new approach can address the requirements of military planning. In particular, we implemented a prototype that represents multiple, possibly conflicting plans, and identifies sources of conflicts or dependencies between plans. It offers new perspectives for military users and showcases the benefits of different representations of plans at multiple levels of abstraction.

The proposed plan representation structure is hierarchical and based on a combination of Hierarchical Task Networks and Hierarchical Goal Analysis. It is sufficiently generic and flexible to represent plans in many domains, and at all levels of abstraction.. The domain-specific knowledge really resides in the goal and task decomposition methods and can be instantiated by, say, land-based or joint domains, or even non-military planning applications. The results of this paper and the benefits of the system are applicable, as long as there is a need for hierarchical (rather than linear procedural) planning and a mix of levels of abstraction for representing plans.

The proposed concept and system naturally support a distributed planning process, in which high-level goals and plans can be specified first and then validated and projected on a high level. The components of the high-level plans can be distributed to other planners at a later time, who fill in plan details at lower levels. This process can proceed in parallel, with all users immediately seeing the results and possible conflicts caused by other users changing plans.

As future work, we would suggest more research on suitable visual concepts to support planning and to automatically propose suitable plans for given planning problems. Intuitive functionality is key to obtaining acceptance among military users, who are used to a graphical, drawing-board-style plan development environment.

References

Ambite, J. L. (2003). "Hierarchical Task Network (HTN) Planning." Course Presentation, from http://www.isi.edu/~blythe/cs541/2003-9-18-htn.pdf.

Chow, R., R. Kobierski, C. Coates, J. Crebolde r (2006). Applied Comparison Between Hierarchical Goal Analysis and Mission, Function and Task Analysis. Proceedings of the H uman Factors and Ergonomics Society 50th Annual Meeting.

COPlanS (2007) – Collaborative Operations Planning System. Fact Sheet IS-228-A, DRDC Valcartier. http://www.valcartier.drdc-rddc.gc.ca/sciences/coplans-ft-fs-eng.asp.

Erol, K., D . Nau, J. Hendler (1994). UMCP: A Sound and Complete Planning Procedure for Hierarchical Task-Network Planning. Proceedings of The International Conference on AI Planning Systems (AIPS), Chicago, IL.

Erol, K., J. Hendler, D. Nau (1994). Semantics for Hierarchical Task-Network Planning. College Park, Maryland, University of Maryland at College Park**:** 28.

Farrell, P. S. E. (2007). Control Theory Perspective of Ef fects-Based Thinking and Operations: Modelling "Operations" as a Feedback Contro l System Tech. Report TR 2007-168, DRDC Ottawa, Canada.

Hales, D. and Scipione, A. (2008), Joint Comma nd Decision Support for the 21st Century (JC DS21) Technology Demonstration Project - Concept of Operations (CONOPs), DRDC. CRDC-CR-2008-02.

Hartmann, L. (2004). A VHDL Implementation of an Onboard Autonomy Solution, Canadian Space Agency.

Hendy, K. C., Beevis, D., Lichacz, F ., & Edwards, J. L. (2002). Analyzing the cognitive system from a perceptual control theory point of view. In M. D. McNeese & M. A. Vidulich (Eds.), Cognitive systems engineering in m ilitary aviation environments: Avoiding cogminutia fragmentosa! (pp. 201-250). Dayton, OH: Wright-Patterson AFB.

Hunter, A., J. Happe, M. Dutkiewicz (2007). D ynamic Plan Managem ent in the Context of a Recognized Air Picture, Contract Report CR 2007-446, DRDC Valcartier, Canada.

Hunter, A., J. Happe et al. (2008) . Execution Management and Plan Adaptation. Tech. Report, DRDC Valcartier, Canada.

Kobierski, R. (2004). Hierarchical Goal Analysis and Perform ance Modelling for the Control of Multiple UAVs/UCAVs from an Airborne Platform, DRDC Toronto.

Muñoz-Avila, H., F . Weberskirch (1996). A Specification of the Dom ain of Process Planning: Properties, Problems and Solutions. Technical Report, Centre for L earning Systems and Applications, University of Kaiserslautern, Germany.

Nau, D., M. Ghallab, P. Traverso (2004). Automated Planning: Theory and Practice, Elsevier Science & Technology.

Nau, D. (2005). May all your plans succeed! Invited talk. National Conference on Artificial Intelligence (AAAI 2005).

Sacerdoti, E. D. (1975). The Nonlinear Nature of Plans. Fourth International Joint Conference on Artificial Intelligence (IJCAI).

Strategic Joint S taff (SJS). (2008). The Canadian Forces Operational Planning Process (O PP). Department of National Defence. Government of Canada. B-GJ-005-500/FP-000.

Tate, A. (1975). Project Planning U sing a Hierarchic Non-L inear Planner. Research Report, Dept. of Artificial Intelligence, University of Edinburgh.