From Abstract Crisis to Concrete Relief

A Preliminary Report on Combining State Abstraction and HTN Planning

Susanne Biundo and Bernd Schattenberg

Department of Artificial Intelligence
University of Ulm, Germany

Abstract Flexible support for crisis management can definitely be improved by making use of advanced planning capabilities. However, the complexity of the underlying domain often causes intractable efforts in modeling the domain as well as a huge search space to be explored by the system. A way to overcome these problems is to impose a sophisticated structure not only according to tasks but also according to relationships between and properties of the objects involved. We outline the prototype of a system that is capable of tackling planning for complex application domains. It is based on a well-founded combination of action and state abstractions. The paper presents the basic techniques and provides a formal semantic foundation of the approach. It introduces the planning system and illustrate its underlying principles by examples taken from the crisis management domain used in our ongoing project.

1 Introduction

When trying to exploit planning technology for realistic applications like system support for crisis management, one of the main problems to be tackled is the complexity of the underlying domain. Not only does it cause intractable modeling efforts, a huge search space has to be explored by the system as well. Furthermore, such a system has to be flexible in the sense that mixed initiative planning has to be supported and incoming information as well as most recently arising tasks should be considered and integrated during runtime. In order to meet multiple requirements like in this case, hybrid planning approaches have to be developed to provide enough flexibility and lucidity as has repeatedly been argued by other authors as well (cf. [5] and [8]).
We introduce a planning approach for system support in the realistic and complex application domain of crisis management. It integrates hierarchical action- and state-based techniques in a consequent way by imposing hierarchical structures on both operators and states. The hierarchical concept is partly adopted from traditional hierarchical task network (HTN) planning (cf. [4]). Therefore, basic notions like primitive and abstract tasks as well as methods for decomposing the latter stepwise into primitive ones are among the core concepts. However, tasks do show pre- and postconditions –like operators do in classical state-based planning– on every level of abstraction. This provides the flexibility to make use of state-based planning techniques by introducing additional tasks when trying to establish missing preconditions, and enables the system to integrate incoming tasks on any level of abstraction at any time. This is done by allowing for so-called *decomposition axioms*, which are defined as part of the domain model. The planning approach is based on a formal semantics, which relies on previous work on logic-based planning [20]. Our formalism builds upon the semantics for basic

STRIPS-like operators and extends the formalism to abstract tasks and decomposition methods.
A first prototype implements our integrative approach. We use an object-oriented programming paradigm, thereby exploiting object-oriented structures and mechanisms to efficiently deal with the hierarchy of planning objects and their properties. According to our experience with this first implementation, the system is currently being completed and extended towards several directions.
The paper is organized as follows. In Section 2 we introduce the formal semantics. The application domain –a mission of the German Federal Agency for Technical Relief at a flood– is briefly introduced in Section 3. Section 4 describes our planning method and illustrates the techniques by means of examples taken from the crisis management domain of Section 3. In Section 5, we shortly report on the implementation and the lessons learned from this experiment. Section 6 is devoted to related work and finally we conclude with some remarks in Section 7.

2 Formal Framework

The Logical Language: The semantics of our planning approach is based on a many-sorted first-order logic. The logical language $L = (Z,R_r,R_f,C,V)$ consists of a finite set of *sort symbols* Z, Z^* indexed families of finite disjoint sets of *rigid* and *flexible relation symbols* (R_r and R_f, resp.), a Z indexed family of disjoint sets of *constants*, and a Z indexed family of disjoint sets of *variables*. Formulae over L are built as usual. The formal planning language is obtained by extending L by O, T, and E. O and T are Z^* indexed families of finite disjoint sets of *operator* and *task symbols*, respectively. For all $\bar{z} \in Z^*$ the sets $R_{r,\bar{z}}$, $R_{f,\bar{z}}$, $O_{\bar{z}}$, and $T_{\bar{z}}$ are supposed to be mutually disjoint. E denotes a Z^* indexed family of so-called *elementary operation symbols*. It provides for each flexible relation symbol R a so-called *add-operation* +R as well as a *delete-operation* –R.
As for the semantics, we adopt some essential features of the planning formalisms introduced in [19] and [20], which are based on programming and temporal logics, respectively.
Following a *state-based* planning approach, we use operators and tasks to take us from one state to another. The flexible symbols provided by our planning language are used to express the changes caused by these state transitions. Consequently, we introduce states as interpretations of the flexible symbols.

States and State Transitions: For a logical language $L = (Z,R_r,R_f,C,V)$ a *model* denotes a structure $M = (D,S,I)$, where D is a Z indexed family of *carrier sets*, S is a set of *states*, and I is a (state-independent) *interpretation* that assigns elements of the respective carrier sets to constants and a relation of appropriate type to each rigid symbol. As usual, sort preserving *valuations* $\beta : V_z \rightarrow D_z$ are used for variables. Given a model $M= (D,S,I)$, an atomic formula $\mathrm{R}(\tau_1, ..., \tau_n)$ is *valid* in a state $s \in S$ under a valuation β denoted by $s \models_{M,\beta} \mathrm{R}(\tau_1, ..., \tau_n)$ according to the following definition.

For $\mathrm{R} \in R_r$: $s \models_{M,\beta} \mathrm{R}(\tau_1, ..., \tau_n)$ iff $(I_\beta(\tau_1), ..., I_\beta(\tau_n)) \in I(\mathrm{R})$
For $\mathrm{R} \in R_f$: $s \models_{M,\beta} \mathrm{R}(\tau_1, ..., \tau_n)$ iff $(I_\beta(\tau_1), ..., I_\beta(\tau_n)) \in s(\mathrm{R})$

Based on these definitions, validity of complex formulae is defined as usual.

Now we are ready to turn from *states* to *state transitions*. To this end, we first assume that our models are *natural* ones [19]. This means, the carrier sets are supposed to be finite and we restrict the set of states to those, which assign finite relations to the symbols in R_f. Furthermore, for each flexible symbol $\mathrm{R} \in R_{f,\bar{z}}$, $\bar{z} = z_1, ..., z_n$, two functions d-$\mathrm{R} : D_{z_1} \times ... \times D_{z_n} \rightarrow S \times S$ and a-$\mathrm{R} : D_{z_1} \times ... \times D_{z_n} \rightarrow S \times S$ are defined as follows.

s d-$\mathrm{R}(d_1, ..., d_n)$ s' iff $s'(\mathrm{R}) = s(\mathrm{R}) - \{(d_1, ..., d_n)\}$ and $s'(\mathrm{R'}) = s(\mathrm{R'})$ for $\mathrm{R'} \neq \mathrm{R}$
s a-$\mathrm{R}(d_1, ..., d_n)$ s' iff $s'(\mathrm{R}) = s(\mathrm{R}) \cup \{(d_1, ..., d_n)\}$ and $s'(\mathrm{R'}) = s(\mathrm{R'})$ for $\mathrm{R'} \neq \mathrm{R}$.

Given a natural model, for any two states s and s' there exists a finite sequence of a-... and d-... function operations $op_1 ... op_n$ such that s $op_1 \circ ... \circ op_n$ s', where $\circ$ denotes functional composition [19].

Elementary Operations: Based on the definitions of a-... and d-... functions on states, we can now define the semantics of the elementary operations E of our planning language as follows. Given a model $M = (D,S,I)$ and a valuation β, a pair of states (s, s') satisfies an elementary operation $+\mathrm{R}(\tau_1, ..., \tau_n)$ according to

$$(s, s') \models_{M,\beta} +\mathrm{R}(\tau_1, ..., \tau_n) \text{ iff } s \ a\text{-}\mathrm{R}(I_\beta(\tau_1), ..., I_\beta(\tau_n)) \ s'$$
$$(s, s') \models_{M,\beta} -\mathrm{R}(\tau_1, ..., \tau_n) \text{ iff } s \ d\text{-}\mathrm{R}(I_\beta(\tau_1), ..., I_\beta(\tau_n)) \ s'$$

This means, elementary operations represent single state transitions.
We finally adopt from [19] the concept of *weakest preconditions* (wp) w.r.t. elementary operations. Let φ be a formula which contains only variables that are distinct from those occurring in τ_1,...,τ_n. The *weakest precondition* of φ w.r.t. $+\mathrm{R}(\tau_1, ..., \tau_n)$ is the formula resulting from φ when replacing all atomic sub-formulae $\mathrm{R}(\sigma_1,...,\sigma_n)$ by $[(\tau_1 \neq \sigma_1 \vee ... \vee \tau_n \neq \sigma_n) \rightarrow \mathrm{R}(\sigma_1,...,\sigma_n)]$. $\mathrm{wp}(\varphi, -\mathrm{R}(\tau_1, ..., \tau_n))$ results from φ by replacing all atomic sub-formulae $\mathrm{R}(\sigma_1,...,\sigma_n)$ by $[\mathrm{R}(\sigma_1,...,\sigma_n) \wedge (\tau_1 \neq \sigma_1 \vee ... \vee \tau_n \neq \sigma_n)]$.

Operators and Invariants: Given a planning language $P = (Z,R_r,R_f,C,V,O,T,E)$, an *operator* (*primitive task*) is a triple $(\mathrm{O}(\bar{\mathrm{x}}), \mathrm{prec}, \bar{\mathrm{e}})$, where O is an operator symbol, $\bar{\mathrm{x}} = \mathrm{x}_1...\mathrm{x}_n$ is a list of variables, prec is a formula over $L = (Z,R_r,R_f,C,V)$, and $\bar{\mathrm{e}} = \mathrm{e}_1...\mathrm{e}_m$ is a (finite) sequence of elementary operations from E. For a given model $M = (D,S,I)$ and a valuation β, this operator transforms a state s into a state s', denoted by $(s, s') \models_{M,\beta} (\mathrm{O}(\bar{\mathrm{x}}), \mathrm{prec}, \bar{\mathrm{e}})$, iff $s \models_{M,\beta} \mathrm{prec}$ (the operator is applicable in s) and s $op_1 \circ ... \circ op_m$ s' where op_i is the a-... resp. d-... function corresponding to e_i for $1 \leq \mathrm{i} \leq \mathrm{m}$. The operator *generates* a formula post over L if in addition $s \models_{M,\beta} \mathrm{wp}(\mathrm{post},\bar{\mathrm{e}})$. The weakest precondition of a formula φ w.r.t a *sequence* of elementary operations is generated according to a straightforward extension of the above definition.
Before finally defining *tasks* and *methods*, we introduce the notion of *invariant* in order to extend the *generation* of formulae from single operators to operator *sequences*.
For a given model $M = (D,S,I)$ and a valuation β, a formula φ is *invariant* against an operator $(\mathrm{O}(\bar{\mathrm{x}}), \mathrm{prec}, \bar{\mathrm{e}})$ iff for all states s and s' with $(s, s') \models_{M,\beta} (\mathrm{O}(\bar{\mathrm{x}}), \mathrm{prec}, \bar{\mathrm{e}})$: if $s \models_{M,\beta} \varphi$, then $s' \models_{M,\beta} \varphi$.
A formula post is *generated* by a sequence $\mathrm{O}_1...\mathrm{O}_n$ of operators iff it is generated by some O_i $(1 \leq \mathrm{i} \leq \mathrm{n})$ and is invariant against each O_j $(\mathrm{i} < \mathrm{j} \leq \mathrm{n})$.

Tasks and Methods: Given a planning language $P = (Z,R_r,R_f,C,V,O,T,E)$, a *task* is a triple (T($\bar{x}$), prec, post), where T is a task symbol, $\bar{x} = x_1...x_n$ is a list of variables, and prec and post are formulae over $L = (Z,R_r,R_f,C,V)$. For a given model $M = (D,S,I)$ and a valuation β, the task transforms a state s into a state s', denoted by $(s, s') \models_{M,\beta}$ (T($\bar{x}$), prec, post) iff $s \models_{M,\beta}$ prec and $s' \models_{M,\beta}$ post and there exist a finite sequence $s_1...s_n$ of states and a finite sequence $O_1...O_{n-1}$ of operators, where $s = s_1$, $s' = s_n$, $(s_i, s_{i+1}) \models_{M,\beta} O_i$ for all $1 \leq i < n$, and $O_1...O_{n-1}$ generates a formula post$'$ such that $s_n \models_{M,\beta}$ post$' \rightarrow$ post. The task *generates* a formula post$''$ iff in addition $s_n \models_{M,\beta}$ post $\rightarrow$ post$''$.
The hierarchical structure of planning domains is reflected in two ways. First of all so-called *methods* are used to specify how an abstract task can be subdivided into a set of (primitive) subtasks, like it is usually done in HTN planning. Secondly, a hierarchy is imposed on the formulae used to express the pre- and postconditions of primitive and non-primitive tasks. To this end, user-defined *decomposition axioms* of the form $\varphi \leftrightarrow [\psi_1 \vee ... \vee \psi_n]$ specify how an abstract condition φ can be refined into a more concrete one, each ψ_i being a possibility to do so.
A method $\{$(T($\bar{x}$), prec, post), $\mathcal{T}\}$ is given by a task and a set $\mathcal{T}$ of task sequences. For each such sequence $t_1...t_n$ the t_i may be primitive or non-primitive. A method is called *legal* iff each task sequence $t_1...t_n \in \mathcal{T}$ is a legal decomposition of the task.
For a given model $M = (D,S,I)$ and a valuation β a task sequence $t_1...t_n$ is a *legal decomposition* of a task (T($\bar{x}$), prec, post) iff the task sequence transforms a state s into s' such that for the precondition prec$'$ of task t_1 $s \models_{M,\beta}$ prec$' \rightarrow$ prec and the sequence $t_1...t_n$ generates a formula post$'$ such that $s' \models_{M,\beta}$ post$' \rightarrow$ post.
The above mentioned axioms are thereby used to justify legality of decompositions when specifying methods. Illegal decompositions can be detected during compilation of the domain.

3 The Application

Our planning domain is crisis management as being provided by organizations like THW. The German *Technisches Hilfswerk* is a governmental disaster relief organization that provides technical assistance at home as well as humanitarian aid abroad. Their mission within the flood disaster at the river "Oder" in July 1997 is used in our ongoing project to build a first realistic domain model for the planner. In the following, examples from this domain will be used to demonstrate our approach. The tasks of the THW are rich and widespread, they cover all aspects of crisis management, ranging from first measures after a hazardous event to long term supplies after clearing some disaster area. Therefore, Figure 1 only shows a relative small part of the complex task hierarchy, and most task networks are depicted as single, more "self-explanatory" actions.
The most abstract task is named `flood-disaster`. It comprises a management and communication task to determine which areas are endangered to what level. Furthermore, the logistics and supplies have to be installed, e.g. quarters for the relievers to be set up. The evacuation and the securing of the embankment are the most crucial sub-tasks in reality, and during all the activities, the relievers have to clear the area continuously, i.e. to check damaged buildings, to remove perished animals, etc.
In our examples we will focus on the evacuation task, which consists of two sub-tasks: one informs the population about the relief measures, the second brings people to safe

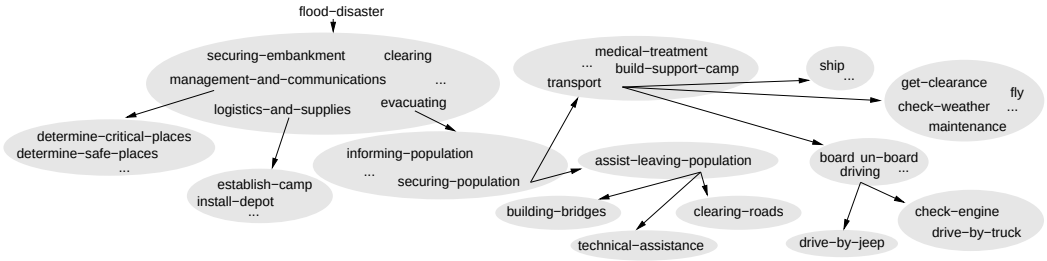

Figure 1. Task hierarchy in the *flood disaster* scenario, ellipses represent task networks

areas. The methods for latter define two expansions, depending on the initial situation. The relievers can help the people to leave the endangered area by themselves, or the circumstances might require the population to be moved by the THW.

4 Combining hierarchical action- and state-based planning

Our planning approach flexibly combines classical HTN and classical state-oriented POCL planning based on the formal framework introduced in Section 2. The prototype implements a simple top-level algorithm comparable to [16, p. 374], which is basically a classical nonlinear planning algorithm with decomposition of abstract tasks as an additional plan modification step. Although it looks very similar to those used by existing hybrid planning systems (see section 6), we will use it to outline the underlying principles in the sub-routines. First we will focus on the closing of open preconditions. In order to enable the planner to reason about the plans' causal structures and dependencies at all levels of abstraction, complex tasks do carry preconditions and effects like the operators do. For the time being they are assumed to be conjunctions of positive and negative literals.
While the relation between abstract and primitive tasks is given by a number of methods as in classical HTN planning, in our approach relations between the respective preconditions and effects are specified by the decomposition axioms. The example in Figure 2 shows two methods for the expansion of the abstract transport task in the evacuation context. The ordering constraints represent all possible sequences of subtasks, sort information for the variables is given in the task definitions.
One of the decomposition axioms that will be applied in the respective expansion steps, will e.g. look like this (assuming the intuitive subsort relationships):

$$\begin{aligned}\text{At}(\text{Unit } u, \text{Location } l) \leftrightarrow [\ &\text{Standing-at}(\text{Vehicle } u, \text{Location } l, \text{Road } r) \vee \\ &\text{Aircraft-at}(\text{Aircraft } u, \text{Location } l, \text{Height } h) \vee \\ &(\text{At}(\text{Container } c, \text{Location } l) \wedge \text{In}(\text{Container } c, \text{Unit } u)) \vee \dots\ ]\end{aligned}$$

The specified decomposition axioms together with the sort and subsort definitions, represent a hierarchy on the relations and objects in the domain. We can make use of this knowledge when closing open preconditions with tasks on different levels of abstraction. When some (possibly abstract) effect of a task is needed to establish the precondition of another, the planner can provide this by choosing some –according to

```
method m_1
  expands transport(?passengers,?from,?to,?by)
  vars    ?road Road
  nodes   (1:board(?passengers,?from,?by))
          (2:driving(?by,?from,?to,?road))
          (3:un-board(?passengers,?from,?by))
          ...
  order   1<2, 2<3
  causal  1--in(?passengers,?by)--2
  binding -
```

```
method m_2
  expands transport(?passengers,?from,?to,?by)
  vars    ?tower Tower
  nodes   (1:get-clearance(?from,?tower,?by))
          (2:check(?by))
          (3:board(?passengers,?by))
          (4:fly(?by,?from,?to,?tower))
          ...
  order   1<4, 2<4, 3<4
  causal  1--cleared(?by)--4,
          2--checked(?by)--4,
          1--in(?passengers,?by)--4
  binding -
```

Figure 2. Example for a method definition

the decomposition axioms– suitable tasks in the partial plan to close the open condition. We then add causal links like in classical non-hierarchical POCL planning to represent causality. But no establisher may be identified, even in the initial state. In this case the planner can introduce a suitable establisher for the open condition from the domain description. Figure 3 shows the planning process in such a situation.

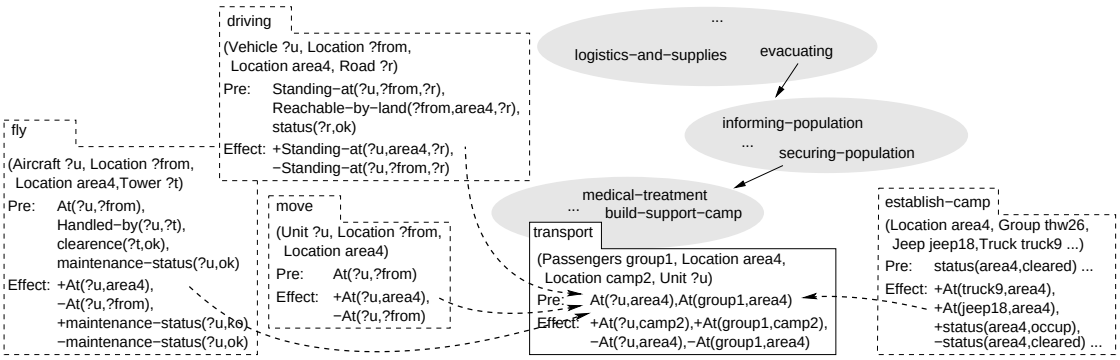

Figure 3. Closing an open precondition along the condition hierarchy.

The abstract need for an arbitrary THW unit to be present in the evacuation area can be fulfilled by any of the tasks shown. The first task `move` is a "classical" candidate. Its sub-task `fly` and the `establish-camp` action qualify in our system, because of aircraft and jeeps being sub-sorts of the abstract sort `unit`. The driving task establishes a more specialized effect than the precondition needs in two ways. Not only vehicles are more special objects, but also the relation `Standing-at` is more concrete than `At` (see decomposition axiom above).

At this point, search control has more choices to investigate, some of which might specialize the involved objects too early, an effect sometimes called *hierarchical promiscuity*. But on the other hand, the commitment to a less abstract establisher can rule out inconsistent solutions at an early stage. We may use the driving task for closing the condition at this point and add a variable assignment for the "downcast" of the unit

in the transportation task. Later in plan generation we might find out, that there is no road to the evacuation area anymore, and the planner has to backtrack and focus on solutions with aircraft.

We note, that especially at this point the search strategy plays a crucial role, i.e. when to insert new tasks. Currently, we work on an extension to the algorithm, that is looking for invariants in expansions. If an open precondition is invariant against all tasks expanded so far, it is obvious, that the planner has to insert a new task. But if ther are tasks in the current plan, against which the condition is not invariant, it is a promising strategy to enforce their expansion. The rationale behind this strategy is to check, whether the expansions in which the desired effect might manififests eventually result in consistent solutions.

Now assume, the abstract movement is chosen for establishing the condition, and a causal link with the label `At(?u,area4)` is inserted. Furthermore, let the planner decide to expand the transport task according to the above definition in method `m_1` into the task network describing a transport by land vehicles. As in classical HTN planning, the specified network substitutes the expanded task in the net, respecting existing orderings and variable bindings, but we cannot update the causal links, because the less abstract tasks show more concrete, and hence syntactically not equivalent, conditions. Our solution lies in the decomposition axioms, according to which we distribute the abstract effects and conditions under the tasks of the expanded network. This means, the decomposition axioms are used to inherit causal links from an abstract level to a more concrete one. Figure 4 shows the result: the passengers are boarded on some vehicle, driven to the camp and then un-boarded again. The more abstract link carried

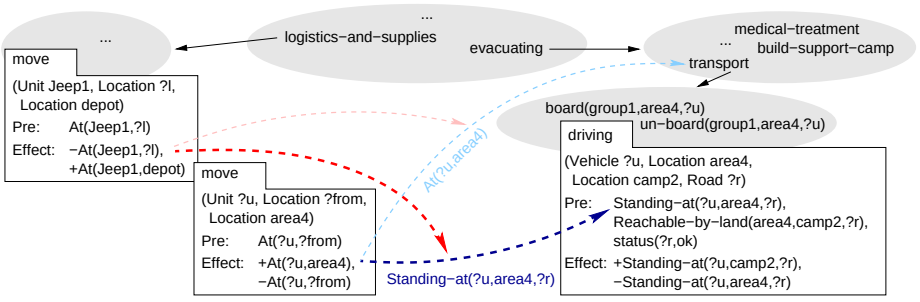

Figure 4. Handling task interactions on different abstraction levels

the `At` relation between a vehicle `?u` and the location `area4`. The decomopsition axioms justify a specialization of this link into `Standing-At` for vehicles. Please note, that the vehicle boarding task may carry the same precondition, which leads to a second inherited causal link derived from the abstract causal relation. Furthermore, when the abstract `move` task is specialized, it has to be checked against the decomposition axioms, whether the newly introduced causal links are inheritable or not. If not, the plan has to be considered inconsistent.

Now we come to threat handling, where the system can make use of the decomposition axioms like it did for condition establishment. Conflicts can be detected and resolved

between arbitrary expansion levels, as the negation of an abstract condition implies the negation of every concrete one specified in the decomposition axioms, and a negation of one of the concrete conditions threatens the abstract one. This mechanism guarantees correct solutions when using the standard POCL conflict resolution strategies at any time the algorithm chooses to check for threats. In addition, besides orderings and non-codesignation, expansion becomes a reasonable threat resolution mechanism. Due to a finer granularity, the conflicting effect might turn out to be harmless at the primitive operator level where the conflicting tasks may overlap.
The example in Figure 4 shows the expansion of a support procedure with a second movement task, and how it interferes with an established condition. The light line indicates the conflict at an abstract level, the dark one does so for the more concrete link. The negated effect can be specialized in a way that makes the plan inconsistent. A simple non-codesignation or some ordering constraint can solve this situation, for a conflict involving several layers of abstraction.

5 Implementation

We implemented a first prototype of the planning system in Java. It integrates task decomposition with state-based planning techniques for conflict handling and closing of preconditions. The main algorithm, which is briefly described in Section 4, performs non-deterministic steps according to a very simple strategy that first tries to close open preconditions, then resolves threats and binds variables, and with least priority expands complex tasks. This rather simple procedure is able to perform a systematic planning strategy like it can be found in classical nonlinear planners: As a last choice in closing preconditions, binding variables, and resolving threats, it tries to expand the respective task. In a least commitment fashion it tries to develop the plan at the most abstract level. The upcoming next version of the system will use a more flexible top level routine which determines the appropriate sequence of plan manipulating steps by analyzing the visited and expected plan space, thereby projecting causal interactions. As already suggested above, we found it e.g. very useful to concentrate expansion within the conflicting tasks to rule out inconsistent solutions at an level as early as possible. However, we note that in the first experiments the proposed modeling approach seems to lead to more “benign” domain models in terms of efficient task hierarchies, that prune large parts of the non-useful search space quite efficient.
To increase the system’s performance, we use a conservative algorithm for manipulating a global plan structure representing the expanded networks. By doing so, we have the additional advantage to automatically bookkeep the performed expansion steps as well as all other choices made by the algorithm (cf. decomposition links in [24]). When looking for an appropriate effect to close an abstract precondition, the system can easily inspect already expanded abstract tasks and follow their decomposition to less abstract levels.
The algorithm allows recursive task expansion schemata to model loops. If for every recursive task a terminating method is provided and if an appropriate search algorithm is used (here: iterated deepening) then the recursion is harmless with respect to program termination and “increasing the incompleteness” of the planning process. These loops are very useful in the presented examples, e.g. evacuation has to be performed until all persons safe, although recursion handling still requires improvement (cf. future work in section 7).

6 Discussion and Related Work

Hierarchical task network (HTN) planning as described and analyzed in [3] is the basis for systems like O-Plan, UMCP and Shop.
In contrast to our approach, which makes use of state abstractions in condition achievement, abstract tasks in O-Plan [2] do not carry preconditions and effects. Instead, the system relates conditions of primitive operators over different levels in the plan generation process by introducing *condition types* in the abstract expansion schemes [21]. These types specify how conditions of the tasks in the expansion can be achieved: by the effect of a task that is *(a)* inside or outside the current expansion and *(b)* introduced at the current plan generation level, above, or below. Please note, that this technique requires the domain encoder to structure the task hierarchy very carefully as methodologically it's pruning works rather on the system's search space structure than on that of plan space. Compared to O-Plan, UMCP [4] is a much more puristic implementation. The search space is constraint pruned, down to the most concrete operator level, where the typical conditions are introduced. Both systems merely rely on action abstraction.
A new direction in the HTN paradigm is given by the Shop system [14], that proposes ordered task decomposition, using if-then-else cascades in method selection. The main idea is to plan all tasks in the order they are later executed. This enables the system to deduce complete state descriptions, beginning with the initial state. The developers met the criticism on their linearity assumption with a modified system, called M-Shop [15] which can handle planning problems with parallel goals in the initial task. Many realistic domains may meet this partial linearity property, the crisis management domain in our work, however, does not as task execution itself is higly distributed and the execution order for most tasks is not known in advance.
Planning using state abstraction was the earliest form of hierarchical planning in linear planning systems. Nonetheless, the Abstrips system [17] is still discussed [7], and has influenced many modern planners. Classical state abstraction works by deleting certain sets of preconditions, thereby defining "criticality levels" for each of which the system plans in a classical manner. Alpine in [9] automatically generates these levels, building abstraction hierarchies with "ordered monotonicity", i.e. detailed action levels do not interfere with more abstract established conditions. Similar work in the context of nonlinear planning has been done by Yang in the Abtweak planner [23].
Exploiting object-oriented formalisms or state abstraction is a comparatively new technique in planning. Semantic foundations for "real" object oriented approaches can be found in the literature, ranging form reasoning about object database models in the style of terminological logics [1] to specification oriented work [18]. [6] uses plans as object methods for an hybrid reactive robot controller, mapping incoming percepts on partially specified object templates as plan selection criteria. More related to our view is that of object centered planning [13], where objects are organized in static and dynamic sorts. Each instance of a dynamic sort has its own local state which is defined by a set of predicates. Consequently, predicates are owned by exactly one sort, the key attribute of the predicate, and thereby becoming static or dynamic themselves. For all sorts legal local states are specified, and over their transitions again operators. OCLh [12] extends this formalism to action abstraction by introducing a sort hierarchy, in which dynamic predicates are inherited from super-sorts. So-called guards play the role of pre and postconditions of objects transition sequences that build the semantics

for abstract tasks. The planning algorithm in this framework repeats an expand then make-sound cycle: after expanding one level of task networks, the system is checking for inconsistencies and repairing them. Although our state abstraction is similar (and so far yet simple, compared to "full" object oriented systems), we can handle the refinement of objects and predicates, likewise, and are not restricted to a fixed planning strategy.

Integrating state-based nonlinear planning capabilities, i.e. reasoning about operator interactions and inserting new plan steps/tasks in the fashion of e.g. UCPop [11], into an action abstracting system promises many advantages. As Estlin, Chien and Wang point out in [5] it adds the strengths of both, at the same time softening their weak points. This is reflected in the modeling process: task networks more naturally represent hierarchy and modularity and enable the user to represent domains in an object oriented form which easier to write and reason about. Decomposition rules can refer to either low- or high-level forms of a particular object or goal, as the information pertaining to specific entities is contained in smaller, more specialized rules. The drawback of this technique is that "inter-modular constraints" [5], i.e. exceptions or special cases in action execution, cannot be represented adequately, which often leads to overly-specified reduction rules. This can be seen in the example in figure 3, where classical hierarchical planners would introduce expansion schemes for every kind of support task to be ordered before the evacuation. Operator based techniques on the other hand help encoding implicit constraints, as their kind of plan refinement is more general and provides more compact representations. In addition, it brings with it an early detection of inconsistencies at an abstract level, together with means of resolving the conflicts. But using solely operators, certain aspects are difficult to represent (for a discussion about the expressive power of HTN planning, see [3]). The advantages of a natural mixed domain knowledge representation are obvious, although difficult to evaluate quantitatively: "[it is] easier to encode the initial knowledge base, fewer encoding errors occur [...], and maintenance of the knowledge base is considerably easier."[5]

Such hybrid systems had been watched suspiciously a long time, because the planning paradigms were considered to be conflictive. New AI textbooks present this approach in the style of state abstraction planning in [23], i.e. the abstract tasks carry preconditions and effects from a subset of the less abstract tasks. Yang suggests in [22] to keep hierarchical models restricted in such a way, that in every reduction schema there is one task carrying the main effects of the network and hence those of the associated abstract task. In such domains the downward solution property holds as a basis for effective search space reduction. A similar approach is presented by Russell and Norvig [16], who allow distribution of conjuncts of conditions among the sub-tasks of the network. One of the very few existing systems is DPOCL [24], mentioned before with its introduction of decomposition links to record decisions during planning. DPOCL decomposes abstract tasks into networks with additional initial and final steps which carry the conditions of the abstract tasks. Some of the techniques used there raise the crucial question of user intent. The system prunes unused steps and takes condition establishers from every level of abstraction, even from sub-tasks of potential establishers. The problem of when to insert new tasks, and where to use decomposition rules only, is very hard to solve, as it depends in part on the modeller's intention. So far, we have provided the system a switch for explicitly not inserting new tasks in precondition achievement, as well as an output, indicating the inserted tasks Moreover, premature insertion of new tasks may lead to non-optimal short plans, but we postpone this problem for this

time as a matter of "good" search strategies, like it is solved for classical state-based nonlinear planners –but of course it will be tackled in the future.

Closely related to our approach is the work of Kambhampati [8]. He integrates HTN planning in a general framework for refinement planning, thereby making use of operator based techniques. This unified view should help making use of recent progress in planning algorithms, e.g. by giving propositional encodings for SAT based planners [10]. In his view, the algorithm uses reduction schemes where available, and primitive actions otherwise. Causal interaction is analyzed also at the abstract level, and refined by a mapping of conditions and effects of abstract tasks on conditions and effects in its sub-tasks. Abstract conditions are closed by phantom establishers that are identified at a later stage, while our algorithm just "waits" if no suitable task is less abstract enough. Conflict detection and resolution can only be done at the primitive level, as in contrast to our methodology, there is no "vertical" link between causalities in the different levels of abstraction. Kambhampati addresses user intent by defining a subset of abstract effects explicitly for condition establishment, and by explicit representing the incompleteness of scheme definitions. For the latter, a specific predicate prevents insertion of new steps.

Another aspect of hybrid planning to mention is its relevance to the relative new area of mixed initiaive planning. Only small modifications to hybrid planning algorithms allow the user to propagate decisions as commitments to the planner, including insertion of new tasks (many technical problems concerning systematicity of the system, etc. are of course beyond the scope of this paper). The resulting system benefits from our state abstraction technique, because the intermediate results, which are the basis for user interaction, become more usable in two ways: *(a)* The explicit representation of causal interactions is intuitive, even for abstract tasks, and *(b)* all modifications can be done at an arbitrary level of abstraction. An example might be an abstract plan with transport tasks, for some of which a human user can decide – on the basis of the plan developped so far – not to be performed by aircraft. He can introduce at this level constraints to choose land vehicles.

7 Conclusions and Future Work

We have introduced a planning approach that integrates hierarchical task networks and state-based POCL planning techniques by imposing hierarchical structures on both tasks and state descriptions. Tasks on all abstraction levels are extended by pre- and postconditions, which enable the flexible integration of hierarchical decomposition and nonlinear planning. A formal semantics of the approach provides the notion of legal decomposition, among others. It is an essential means to ensure that during domain modelling tasks and state abstractions are defined in a mutually consistent way. A planning system has been presented, which implements this integrative planning approach. It will be used to flexibly generate mission plans for environmental disasters.

Future work will, among others, be devoted to even further exploit the object-oriented implementation paradigm and to the implementation of a more flexible search strategy. Furthermore, the example domain strongly demands resource reasoning, especially time, and a specialized loop mechanism.

References

1. D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of DOOD-95*, volume 1013 of *LNCS*, pages 229–246, Berlin, 1995. Springer.
2. K. Currie and A. Tate. O-Plan: The Open Planning Architecture. *AI*, 52(1):46–86, 1991.
3. K. Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation.* PhD thesis, The University of Maryland, 1995.
4. K. Erol, J. Hendler, and D. S. Nau. UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning. In *Proc. of AIPS-94*, pages 88–96, Chicago, IL, 1994. American Assoc. for AI, AAAI Press, Menlo Park, California.
5. T. A. Estlin, S. A. Chien, and X. Wang. An argument for a hybrid HTN/operator-based approach to planning. In *Proc. of ECP-97*, volume 1348 of *Lecture Notes in AI*, pages 182–194, Berlin, Sept. 24–26 1997. Springer.
6. U. Fonda, A. Natali, and A. Omicini. An object-oriented approach to planning. In *FAPR'96 Workshop "Reasoning about Actions and Planning in Complex Environments"*, pages III–1/8, Darmstadt, Germany, 1996.
7. F. Giunchiglia. Using ABSTRIPS abstractions – where do we stand? IRST Technical Report 9607-10, Istituto Trentino di Cultura, Italy, Jan. 1997.
8. S. Kambhampati, A. D. Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *Proc. of AAAI-98*, pages 882–888, 1998.
9. C. A. Knoblock. Automatically Generating Abstractions for Planning. *AI*, 68:243–302, 1994.
10. A. Mali and S. Kambhampati. Encoding HTN Planning in Propositional Logic. In *Proc. of AIPS-98.* AAAI Press, 1998.
11. D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. of AAAI-91*, pages 634–639, 1991.
12. T. McCluskey. Object transition sequences: A new form of abstraction for HTN planners. In *Proc. of AIPS-2000*, pages 216–225, 2000.
13. T. McCluskey, D. Kitchin, and J. Porteous. Object-centred planning: Lifting classical planning from the literal level to the object level. In *Proc. of 11th IEEE Int. Conf. on Tools with AI*, Toulouse, 1996. IEEE Press.
14. D. S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proc. of IJCAI-99*, pages 968–975, S.F., 1999. Morgan Kaufmann Publishers.
15. D. S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP and M-SHOP: Planning with ordered task decomposition. Technical Report CS TR 4157, University of Maryland, 2000.
16. S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach,.* Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-103805-2, 912 pp., 1995, first edition, 1995.
17. E. D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *AI*, 5:115–135, 1974.
18. A. Sernadas, C. Sernadas, and J. F. Costa. Object specification logic. *Journal of Logic and Computation*, 5(5):603–630, 1995.
19. W. Stephan and S. Biundo. A New Logical Framework for Deductive Planning. In R. Bajcsy, editor, *Proc. of IJCAI-93*, pages 32–38. Morgan Kaufmann, 1993.
20. W. Stephan and S. Biundo. Deduction-Based Refinement Planning. In B. Drabble, editor, *Proc. of AIPS-96*, pages 213–220. AAAI Press, 1996.
21. A. Tate, B. Drabble, and J. Dalton. The Use of Condition Types to Restrict Search in an AI Planner. In *Proc. of AAAI-94*, pages 1129–1134. AAAI Press, 1994.
22. Q. Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach to Classical Planning.* Springer, Berlin, 1997.
23. Q. Yang, J. Tenenberg, and S. Woods. On the implementation and evaluation of abtweak. *Computational Intelligence Journal*, 12(2):295–318, 1996.
24. R. M. Young, M. E. Pollack, and J. D. Moore. Decomposition and causality in partial order planning. In *Proc. of AIPS-94.* American Assoc. for AI, AAAI Press, 1994.