On the Adequacy of Hierarchical Planning Characteristics for Real-World Problem Solving*

L. Castillo, J. Fdez-Olivares, A. González

Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Granada
18071 Granada. {L.Castillo,faro,A.Gonzalez}@decsai.ugr.es

Abstract Starting from a set of requirements which should be accomplished by any hierachical planning approach to real-world problem solving, we show how known hierarchical models may be improved by means of a hybrid approach. Additionally, on the basis of this hybrid model, which is being applied to a real domain, we present a compared criticism about known hierarchical planning properties and its usefulness to reflect the complexity of real-world problems.

Keywords: hierarchical planning, abstraction formalism, planning in real domains

1 Introduction

The use of hierarchical planning techniques is intended to achieve two goals. On the one hand, to improve the efficiency in time and space of non-hierarchical planning methods [3,4,14,17,18,24,23]. And, on the other hand, to achieve problem solving strategies closer to those exhibited by humans in real-world problems [5,10,11,20]. In any case, these hierarchical planning techniques search for a sequence of plans $\{\mathcal{S}^1,..,\mathcal{S}^n\}$ such that $\mathcal{S}^1$ is a solution at the highest level of abstraction, $\mathcal{S}^n$ is a solution at the lowest level of abstraction, that is, a primitive solution, and every plan $\mathcal{S}^{i+1}$ is a valid refinement of a more abstract plan $\mathcal{S}^i$.

In order to achieve those goals, a hierarchical planning model for real-world problems should be based both on some knowledge abstraction formalism for domains descriptions and also on some planning process able to obtain abstract plans and refine them into primitive plans, but in any case, these planning models should satisfy the following requirements:

- Expressiveness. The domain description language and the abstraction formalism should support the representation of real-world problems similarly to hierarchical problem solving knowledge representations of humans.
- Simplicity. The stage of domain description should be as simple as possible, that is, it must have the lowest number of syntactical restrictions and, if possible, it must be supported by knowledge acquisition tools.
- Autonomy. A hierarchical planning model should reduce at maximum the number of decisions taken during domain description, most of them manually defined by humans, and translate these decisions into the planning process which reaches a higher responsibility in the problem solving process. The fact is that domain descriptions may be used to code some procedural knowledge, that in many cases may be an excessive amount of this type of knowledge. The goodness of this practice is not clear. On the one hand, the encoding of procedural knowledge in domain descriptions may be very efficient and may be a straightforward domain description technique. However this could lead the planner not to solve some instances of problems if some "precoded knowledge" is missing. On the other hand, a more declarative knowledge representation in domain descriptions may be more difficult or not very intuitive to handle but, because of its generality, the scope of solvable instances of problems is increased.
- Soundness and completeness. A hierarchical planning process should find valid solutions at every abstraction level whenever they exist.
- Efficiency. In order to adequately exploit a hierarchy of knowledge, a hierarchical planning process should intensively reuse the knowledge embedded in higher level solutions as a guide to refine lower level solutions more efficiently.

* This work has been supported by the spanish government CICYT under project TAP99-0535-C02-01.

Known models of hierarchical planning hardly satisfy all of these requirements, but only some of them, since there seems to be a trade-off between the attainment of ones with respect to others. In general, this is true both in non-decompositional models based on abstraction hierarchies [3,17,18,24] and in decompositional models, either HTN-based models [11,16,20] or POCL-decomposition models [12,13,25].

In the case of decompositional methods, mainly HTN methods, it is widely recognized that they have a great expressiveness power for real world problems [16], however, there is a high number of decisions which must be taken during the description stage of a domain, decreasing the autonomy of the planning process and increasing the effort needed to represent a domain [20,23]. In the case of non-decompositional methods, mainly abstraction hierarchies, they are almost completely devoted to improve the efficiency of planning processes, disregarding expressiveness issues. They represent an excellent theoretical framework for the study of hierarchical planning processes, but its lack of expressiveness makes these methods more difficult to apply in real-world problems. These models may lead to conclude that any improvement in expressiveness implies a decrease in efficiency, and this is not always true [8].

The need to satisfy the above mentioned requirements has lead to the definition of a set of properties which only appear in hierarchical planning models. The main properties which can be found in the literature are *Upward Solution Property*, *Monotonic*, *Ordered Monotonic* and *Downward Refinement Property* [3,17,23].

These properties are intended to define a reference framework for the design of "good" hierarchical planning models so that any model which meet these properties, also satisfies some of the requirements, mainly, efficicency and completeness. However, despite its undoubted theoretical usefulness and the benefit achieved in the development of hierarchical planning models, this set of properties may be questioned because they do not take into account all of the requirements and some property may put in serious risk some of the requirements, mainly expresiveness, i.e. an appropriate representation of knowledge, one of the most important issues for solving real-world problems [9].

However, it is possible to achieve a higher attainment of these requirements by means of a hybrid planning process which could use decomposition techniques jointly with operator-based techniques to benefit of the advantages of both models [12]. This work analyzes the main shortcomings of known hierarchical planning methods and presents a hybrid hierarchical planning model which provides a higher accomplishment of the above requirements and that has been used to solve real-world problems [7,8]. The hybrid process presented in [12] is only focused on representational issues, by reusing concepts from HTN and operator-based planning, neglecting details about its impact on the planning process. This work is a deep effort to explicit the syntax and semantics of decomposition and modularity based on a representational scheme which differs from that of HTN and operator-based planning, and also to explicit the impact of this knowledge representation in the planning process.

In order to correctly argue this criticism, the main issues during the design of a hierarchical planning model have been identified, and they will be used to show how known hierarchical planning methods present some shortcomings in every category and how the approach presented in this work solves some of them providing a higher degree of accomplishement of the requirements. These issues are the following ones:

- Issues related to domain description.
  1. The definition of the abstraction formalism.
  2. A syntactic description to articulate the decomposition of actions.
  3. A semantic description to identify valid decompositions and modularity relations.
- Issues related to the planning process.
  1. Completeness issues in backtracking between different abstraction levels.
  2. Consistency issues of the causal structure of plans between different abstraction levels.

Next section will show the shortcomings of known approaches to hierarchical planning under the point o view of these issues, and this same point of view will be used to show the contributions of this paper in Section 3. Finally, some reflections about the role of these issues, and more complex interactions between the requirements are shown.

2 Shortcomings of known hierarchical models

2.1 Abstraction formalisms

Every hierarchical planning model establishes a set of syntactical tools to allow for the description of different abstraction levels in a domain. This abstraction formalism is used during the planning process in a "top-down"

refinement of high level plans into lower level plans which ends when primitive plans are obtained. The definition of this abstraction formalism directly has effects on requirements such as expressiveness, simplicity and autonomy.

An optimal abstraction formalism should allow for a real knowledge abstraction at different levels [5,15,21] so that actions and literals are represented at different granularity levels and the number of syntactical rules and human decisions are reduced to the minimum in the stage of domain description.

Abstraction hierarchies based models use "literal-oriented" abstraction formalisms in which every literal from level i remains at level $i+1$. This is a very hard syntactic restriction since it limits the abstraction of knowledge, mainly actions, which maintain their semantics at every abstraction level, but described with a different number of literals, so that this formalism is unable to represent compound actions. For this reason, it is very difficult to apply in real-world problems [5,21].

On the other hand, decompositional models use an "action-oriented" abstraction formalism, based either on static decompositions (reduction schemes) [11,23] or on dynamic decompositions (decompositon schemes) [12,13,25] which allow for a real abstraction of actions so that lower level actions are represented with a higher granularity that higher level actions. This formalism provides a great expressiveness power for complex problems [16]. However, in these models, every precondition and effect of a compound action must be somehow distributed amongst its constituent subactions [23,25]. This restriction limits the real abstraction of knowledge, since all of the abstraction levels share the same set of literals, it decreases the simplicity of domain descriptions and needs a great effort of human decisions.

2.2 Decomposition mechanism

This is a set of syntactic tools to decompose high level actions into lower level subactions and it influences expressiveness, simplicity, autonomy and completeness. With respect to expressiveness and completeness, the decomposition mechanism must allow for alternative decompositions and a true modularity of actions, that is, a different granularity of knowledge between the compound action and its subactions. With respect to simplicity and autonomy, action decomposition should provide the means to describe domains with the minimum amount of knowledge supplied by humans.

Action decompositions are present in decompositional models but do not appear in abstraction hierarchies. Although there are some minor differences between them, in both models the decomposition mechanism always need extra knowledge which has to be coded by hand. It is based on a set of static reduction rules which must be supplied by hand during domain description: the antecedent of the rule is a compound action and its consequent is the set of subactions, its relative ordering a set of causal links between them and even binding constraints. These mechanisms are very expressive but also make the stage of domain description very difficult.

2.3 Semantic validity of decompositions

The decomposition mechanism establishes a modularity relation between two plans $\mathcal{S}^i$ and $\mathcal{S}^{i+1}$ at different abstraction levels so that every action in $\mathcal{S}^i$ is mapped into a set of actions of $\mathcal{S}^{i+1}$. Not every syntactically valid decomposition is also a semantically valid one, so a set of criteria which feature semantically valid decompositions, i.e. modularity relations, should be defined taking into account the following issues.

- Every action at level $i+1$ must be related to an action at level i.
- A decomposition of an action should not have any internal unsolvable flaw.
- Subactions should not interfere with the effects of higher level of the action whose decomposition they belong to.

In known decompositional models these issues are responsibility of the human who writes a domain description, decreasing the autonomy of the planner. This shift in responsibility could be avoided by defining general semantic properties which must be satisfied by any modular decomposition, and giving more responsibility to the planner to find a valid decomposition by means the syntactic tools of the mechanism and reducing the effort of coding a domain.

2.4 Backtracking between abstraction levels

A key issue in any hierarchical planning algorithm is the need of backtracking between different abstraction levels during the search process. This is particularly important when at some abstraction level i a solution cannot be found and the algorithm needs to go back at level $i-1$ to search for other refinements. This can be seen as a knowledge-based pruning mechanism which is able to reject dead-end branches at high abstraction levels and provide an improvement in efficiency. However, depending on the abstraction formalism and on the features of a valid solution, there may be some *domains* in which a planner may lose its completeness due to this backtracking mechanism [17,23].

The existence of these domains led to the definition of the *Upward Solution Property* (USP) [17,23]. This property states that if a primitive solution exists $\mathcal{S}^n$ in a hierarchical domain, then there is a sequence of refinements $\{\mathcal{S}^1,..,\mathcal{S}^n\}$ which starts at the highest abstraction level and ends at the lowest level such that every $\mathcal{S}^{i+1}$ is a valid refinement of $\mathcal{S}^i$. It may be seen that the contrapositve of this property allows for backtracking when a solution does not exist at any abstraction level, i.e., there will be no solution at primitive level.

This property is always satisfied by abstraction hierarchies based models [3,17]. But in the case of HTN based models, it is shown [23] that this property does not usually hold but it can be satisfied by the addition of some syntactic restrictions, mainly the *Unique Main Subaction* (UMS). This restriction states that all of the preconditions and effects of a compound action must be reproduced in only one subaction of its decomposition. This restriction leads to use the same granularity of knowledge in compound actions as well as in their subactions, thus completely losing the modular relation of a real decomposition of actions. This implies a lose in expressiveness in real-world problems and, therefore, most HTN based planners do not satisfy the USP. One might think that if USP is not satisfied, then completeness of decompositional planning algorithms may be put in risk in those situations in which the algorithm bactracks but, actually, it depends on the representation of the final solution.

When the solution to a problem is seen as the lowest level plan, i.e., a primitive plan, then hierarchical planning methods may be seen as another heuristic to improve the efficiency of planning, that is, a different way to arrive at a one-level plan which solves a planning problem. In these cases completeness may be lost since in some domains, there can be a primitive solution but no abstract solutions and thus a backtracking criterium based on the nonexistence of abstract solutions would not be enough. On the opposite, when a solution to a problem needs a complete hierarchy of valid plans where every abstract plan is completely autonomous and operational, and it is used as a modularization tool for lower level plans, then the nonexistence of any abstract solution impedes the existence of a complete hierarchy of valid plans and it can be used as a sound backtracking criterium.

Another property related to USP is the *Downward Refinement Property* (DRP) [2,3]. The DRP states that if a non-abstract, concrete level solution to the planning problem exists, then any abstract solution can be refined to a concrete solution without backtracking across abstraction levels. This property is also hard to satisfy in real-world problems. Its fulfilment implies a drastic improvement in efficiency since a hierarchical planner does not need to backtrack between abstraction levels, but in order to do that, there cannot be more than one decomposition for every compound action. This is also very restrictive for real-world problems in which the need to represent alternative decompositions has been widely recognized as a key requirement [16,20].

2.5 Consistency of causal structures through abstraction levels

This last issue consists of reusing the set of causal links established at some abstraction level i as a guide to refine a plan at level $i+1$. When every causal link established at level i is somehow "inherited" at level $i+1$ then a hierarchical planner is said to satisfy the *Monotonic Property* [2,17,23,24]. This property is very important with respect to the attainment of efficiency and consistency of the planning process:

- The efficiency may be improved since the reuse of causal links which have been established at higher levels avoids the redundancy of the planning process in the sense that flaws previously solved at higher levels do not need to be reconsidered again at every lower level. Furthermore, unsolvable threats may be detected earlier to prune dead-end branches.
- From the point of view of the consistency of the planning process, it would not be easily understandable that any action in a plan at some level could contradict any causal relation previously established at any previous level despite of the correctness of that plan in its own level of abstraction.

Therefore, a hierarchical planning model for real-world problems should satisfy this semantic property. In the case of abstraction hierarchies, this property directly holds, due to the abstraction formalism and independently of

the planning process followed, either ABSTRIPS [3,17] or ABTWEAK [24]. HTN based models provide the means to inherit causal links between abstraction levels, but once again, they must be hand-coded during domain descriptions [23] decreasing the autonomy of the planner, whereas in other decompositional models this issue is not addressed [12,13,25].

Next section will show the main contribution of this paper and how it provides helpful advances in every issue discussed in this section with respect to both non decompositional and decompositional approaches to planning for real-world problems.

3 A hybrid model for real-world problem solving

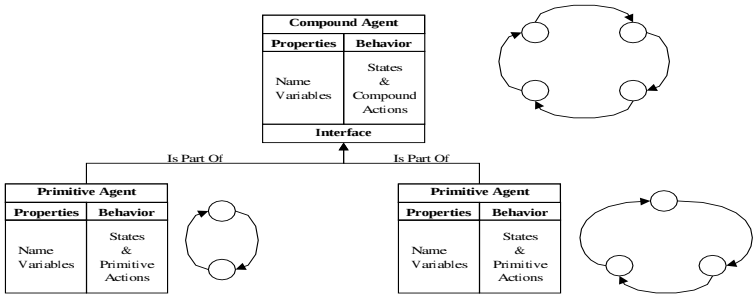

Figure1. Primitive and Compound agents.

In our model a domain is represented as a *compositional hierarchy of agents* (see Figure 1), i.e., an agent hierarchy with different levels of abstraction such that high-level agents (*compound agents*) are composed by lower-level agents. Leaf nodes of the hierarchy are *primitive agents*, which represent real world entities able to act. We can find many real applications which fit into these features [22] (robot planning and control [1], manufacturing systems [6,19] or aerospace applications [12]).

Every agent g of a compositional hierarchy is represented by means of its properties (name and variables) and its behaviour. The behaviour is modeled as a finite automaton in which every action a of g is represented by a set of requirements, *Req*(a), which must be satisfied in order to achieve a correct execution of the action, and a set of effects, *Efs*(a), which include the change of state produced by a over the agent g (we note *Ag*(a) the agent which an action a belongs to). In order to achieve an adequate expressiveness for real-world planning, the model of actions embodies two fundamental features:

- Actions are considered as intervals, that is, every action a of an agent g executes over an interval, $[a, \mathit{End}(a)]$, defined from a until the next change of state of g, produced by another action of the same agent, *End*(a).
- The set of requirements is divided into a set of condition types to represent different ways for preconditioning an action (see [7,8] for more details).

Actions in a compositional hierarchy may be *primitive actions*, if they are executed by primitive agents, or *compound actions*, if they are executed by compound agents. Both action types share the same structure, but compound actions, which are represented at a higher level of abstraction, additionally include a set of *expansion methods*, which are used to decompose them into subactions. Details about this decomposition mechanism will be described later in Section 3.2.

Remaining sections are devoted to explain in more detail our hybrid model, and to show how the key issues of known hierarchical models, shown in Section 2, may be improved in order to solve real-world problems.

3.1 Abstraction formalism

As opposite to the formalisms described in Section 2.1, the abstraction formalism of our hybrid approach is *behaviour oriented*, that is, the abstraction is centered on the behaviour of compound agents in such a way that the behaviour of high level agents is a more abstract representation of the behaviour of their components.

Firstly, the knowledge of a compound agent and the knowledge of its components are related by means of the *Interface* of the compound agent [6] (noted as Int_g). This is a set of simple association rules used to map literals, variables and states of an agent into literals, variables and states of its components. This mechanism allows an expert to easily describe and relate actions and literals of agents at different levels with a different semantic granularity, such that sets of literals and actions at different abstraction levels are also different.

Secondly, in order to articulate levels of abstraction downwards, during the planning process, we have defined an *articulation function* [6,15]. This function (noted as f_g) is a domain independent mechanism defined for every compound agent g which uses its interface to translate every literal into a new level of abstraction. Hence, every literal l=(N $x_1 \ldots x_n$) at level i, in the description of a compound action α^1 of a compound agent g, is translated into a set of literals of level $i+1$, whose arguments are consistent with the new level of abstraction. These literals are built according to these rules:

- If there is a rule in Int_g which specifically associates l to a set of literals, then $f_g(l)$ returns that set.
- Otherwise, the literal is translated component by component, i.e., $f_g(l)$ returns the single set $\{$ ($Int_g(n)$ $Int_g(x_1) \ldots Int_g(x_n)$)$\}$
- Finally, if there is no semantic correspondence in the next abstraction level for l, $f_g(l)$ returns the empty set.

Both the articulation function and the interface of compound agents may be used to relate knowledge at different granularities. The knowledge of a compound agent, with a lower granularity, may be translated into the knowledge of its constituent agents, with a greater granularity.

Additionally, there is a sort of assisting tool for domain descriptions: an expert may describe a domain as a compositional hierarchy from an existing *class library* of predefined agents which also contains predefined agent interfaces. That is, rules contained in interfaces may be easily reused, simplifying the process of domain encoding.

This abstraction mechanism provides a real abstraction of knowledge since actions and literals in the hierarchy are represented at different granularity levels, but related between them. As a result, the amount of syntactic constraints which must be observed during the domain description stage is reduced. There is no need to distribute preconditions and effects at lower abstraction levels since this knowledge will be translated by the articulation function. Moreover, a human domain designer only has to specify which are the components of a compound agent and, if needed, to specialize some agents by introducing domain specific knowledge.

As will be seen later, this formalism is able to maintain an HTN-like expressiveness, but with a simpler domain description process which reduces human decision making at early stages.

3.2 Decomposition Mechanism

Every compound action α of every agent at level i, contains a set of *expansion methods* $\{m_0, m_1, \ldots\}$ where every method $m_{j,j>0}$ is a set of literals at level $i+1$ which represent subgoals to be achieved by actions of agents at the next level $i+1$. This set of literals will be noted as *Decomp*(α) and they are a semantically equivalent representation of the effects of α at level $i+1$.

From the point of view of a human at the domain description stage, it must be taken into account that every compound action α contains a *default expansion method*, m_0, whose literals are the result of applying f_g to every literal in the set of effects of α. Moreover, in most cases this method has shown to be expressive enough to describe how a compound action may be decomposed.

The decomposition mechanism is a dynamic process, performed at planning time, which decomposes a compound action α at level i following these two steps:

1. Determine the set of literals at the next level $i+1$ which have to be achieved by actions of agents at that level. It may be obtained from m_0, without any additional knowledge, or by means of another alternative method. .
2. Determine, by means of POP-based techniques, the set of actions such that either they satisfy those literals generated by α or they contribute to their establishment. These actions will make up the real decomposition of α.

This decomposition process preserves the expressiveness of decomposition rules of other models, that is, it allows for alternative action decompositions, with different grain size. Additionally, it improves decomposition mechanisms discussed in Section 2.2 in the sense that it does not require to know the modular decomposition

[1] Greek letters represent higher level actions and latin letters lower level actions.

of every compound action, prior to the planning process, and in most cases the only required knowledge is the interface of every compound agent, covering much better requirements of simplicity and autonomy.

Another feature of this mechanism is that the correctness of a decomposition does not have to be checked "a priori" by hand. Instead of this, this task is shifted into the planning process who becomes the responsible to dynamically check the correctness of every decomposition at every level of abstraction.

3.3 Semantic validity of decompositions

A key issue in the previously described decomposition process is that it allows to shift the task of determining the set of subactions which corresponds to a compound action into the planner, while preserving semantic correctness. This feature leads to define a *modularization relationship* between two plans at different levels of abstraction. In the following we will introduce some fundamental concepts that, finally, will allow us to define what we consider a valid decomposition as a *correct modularization relationship*.

Modularization relationships In our compositional hierarchy we use two functions, *Sub*(α) and *Scope*(a), to represent hierarchical relationships "is composed by" and "is part of" respectively, so we say that $a \in$ *Sub*$(\alpha) \Leftrightarrow \alpha =$ *Scope*(a). Function *Scope*(a) (Figure 2) gathers the general criteria that a hierarchical planer has to take into account to dynamically find the set of subactions $\{a_1, a_2 \ldots\}$ of a compound action α. It takes as input an action a, from a plan $\mathcal{S}^i$ at level i, which satisfies a literal l from the same level, and it decides which is the *scope* of a, that is, the compound action α at the inmediately higher level which contains the action a in its decomposition.

Scope(a)
IF $\exists \alpha$ / $a \xrightarrow{Sat} l, l \in$ *Decomp*(α)
THEN Let $\{\alpha_1, \ldots, \alpha_n\}$ such that $a \xrightarrow{Sat} l_i, l_i \in$ *Decomp*(α_i)
Let α = Non Deterministically Choose One Of **First**$\{\alpha_1, \ldots, \alpha_n\}$
ELSE Let $\{a_1, \ldots, a_n\}$ such that $a \xrightarrow{Sat} l_i, l_i \in$ *Req*(a_i)
Let α = Non Deterministically Choose One Of **First**$\{$*Scope*$(a_1), \ldots,$ *Scope*$(a_n)\}$
Return α

Figure2. Function *Scope*

Summarizing, the scope of an action a is a higher level action α if one of the following conditions holds:

- a establishes several literals generated by different higher level actions, $\{\alpha_1, \ldots, \alpha_n\}$, and α is one of the first actions of this set (function **First** returns a set of actions for which no other actions precede them in the set of actions used as argument). It must be said that a same action may contribute to different higher level actions but it is assigned to only one of them.
- a establishes several requirements of a set of actions at its same level, $\{a_1, \ldots, a_n\}$, and α is one of the first *scopes* of this set of actions.

This definition is based on literal satisfaction, so it is possible to dynamically generate, at planning time by means of POP-based techniques, the set of subactions for a given compound action at level i. Moreover, functions *Scope*(a) and *Sub*(α) establish a modularization relationship between plans at two consecutive abstraction levels, such that every action in a plan at level i is mapped into a set of actions of a plan at the next level $i+1$.

Taking into account the terms of this definition, POP-based causal link management is the natural way to dynamically guarantee a valid causality between the subactions of a compound action.

Correct modularization relationships In order to guarantee a correct modularization relationship, high-level effects of a compound action α at level i should not be deleted by any of its subactions at level $i+1$. The set of literals *Decomp*(α), generated at level $i+1$ by the above described decomposition mechanism of α, are a semantically equivalent representation of the effects of α at level i, therefore these are actually the literals which should be protected at level $i+1$.

The new concept of *hybrid causal link* is used to protect these literals. In order to understand how a hybrid causal link is represented, and its semantic implications, it is necessary to know the extended representation of a causal link used in our model, which embodies the notion of actions interval, a more suitable representation for real-world domains [7,8].

A causal link $[a \xrightarrow{l} b, c]$ is a structure used for representing that a requirement l of an action b has been satisfied by another action a and this has to be protected during the action interval $[a, c]$ and c is not necessarily $End(a)$. So, an action a' threatens a causal link $[a \xrightarrow{l} b, c]$, when a' deletes l and the interval $[a, c]$ is unordered with respect to the interval $[a', End(a')]$[2]

Definition 1 (Hybrid causal link.) *A hybrid causal link at level i is a structure represented as $[a_\alpha \xrightarrow{l} \beta, \gamma]$, where*

- *β and γ are two compound actions at level $i-1$ which belong to a causal link $[\alpha \xrightarrow{r} \beta, \gamma]$ at level $i-1$.*
- *l is a literal at level i generated by α, at level $i-1$, such that $l \in f_{Ag(\alpha)}(r)$.*
- *a_α is an action which satisfies l and $\alpha = Scope(a)$.* □

A hybrid causal link describes that a literal generated by α and satisfied by some subaction a_α of α has to be protected from any other threatening action a', at the same level than a_α, which could delete that literal. Thus, a hybrid causal link will be used to detect and solve a *hybrid threat*, an analogous concept to that of classic threat which takes into account the existence of causal links at different abstraction levels.

Definition 2 (Hybrid Threat.) *An action a' produces a hybrid threat to a hybrid causal link $[a_\alpha \xrightarrow{l} \beta, \gamma]$ when*

i) $a' \xrightarrow{Del} l$
ii) $[a', End(a')]$ *is unordered with respect to* $[a, \gamma]$ □

Finally, based on these concepts of hybrid causal link and hybrid threats we are able to establish what we consider a *correct modularization relationship*. A correct modularization between two plans at consecutive abstraction levels exists when there are no hybrid threats within the scope of any compound action, that is, when any effect of a compound action is not deleted by any of its subactions.

Therefore, any modularization relationship must satisfy that

$$\nexists a_\alpha, a'_\alpha \in Sub(\alpha), \text{ such that } a'_\alpha \text{ produces a hybrid threat to a hybrid causal link } [a_\alpha \xrightarrow{l} \beta, \gamma].$$

The definition of a correct modularization relationship is a particular case of a hybrid threat in which the threatening action and the causal link belong to the same scope. It allows a planner to decide which decompositions are correct and also to establish the order relations and causal links between its subactions, both dynamically. Moreover, from the point of view of a human domain designer, the planner will be able to modularize the actions of one level with respect to the actions of a higher level, a task which is only carried out by humans in HTN methods.

This procedure to dynamically identify valid decompositions of actions allows for an increase of autonomy of the planner and a greater simplicity of domains descriptions, since the knowledge which identifies these valid decompositions does not have to be provided during domain coding. Instead, this knowledge is "distilled" during the planning process.

3.4 The planning process

The goal of the planning process is obtaining a hierarchical plan which correctly describes the behaviour of a compositional hierarchy of agents at different levels of abstraction. A solution to a problem is a hierarchical plan, that is, a sequence of plans $\{\mathcal{S}^1 \ \ldots \ \mathcal{S}^n\}$ at different abstraction levels, where every plan $\mathcal{S}^i$ must be a valid solution at level i, which will be called a level i partial solucion, since it describes the behaviour of agents at level i. The lowest level plan is completely made up of primitive actions, and every plan at level i has a correct modularization relationship with respect to the plan at the next abstraction level $i+1$. The algorithm which generates such hierarchical and modular plans is shown in Figure 3.

[2] Two intervals are unordered when their limits are unordered [8].

HYBIS(Domain, Agenda, H-Plan)
IF Agenda *is empty* AND **IsPrimitivePlan**(H-Plan[CurrentLevel])
THEN RETURN H-Plan
ELSE LET Flaw = **SelectFlaw**(Agenda)
IF Flaw *consists on a hierarchical refinement*
THEN RETURN **REFINE**(Domain, Flaw, Agenda, H-Plan)
ELSE RETURN **GENERATE**(Domain, Flaw, Agenda, H-Plan)

REFINE(Domain, α, Agenda, H-Plan)
LET Methods = **HowToRefine?**(α, Domain, H-Plan)
WHILE Methods *is not empty*
LET m = **ExtractExpansionMethod**(Methods)
Insert(m, Plan-H])
LET Result = **HYBIS**(Domain, Agenda, H-Plan)
IF Result $\neq$ FAIL THEN RETURN Result
RETURN FAIL

GENERATE(Domain, Flaw, Agenda, H-Plan)
LET Alternatives = **HowToDoIt?**(Flaw, Domain, H-Plan)
WHILE Alternatives *is not empty*
LET How = **Extract**(Alternatives)
DoIt(How, Domain, Agenda, H-Plan)
LET Result = **HYBIS**(Domain, Agenda, H-Plan)
IF Result $\neq$ FAIL THEN RETURN Result
RETURN FAIL

Figure3. A hierarchical hybrid algorithm based on hierarchical HTN-like refinement and generative POP-like refinement.

This algorithm synthesizes a hierarchical plan by generating and refining a set of plans at different levels of abstraction. Every plan $\mathcal{S}^i$ describes the behaviour of a set of agents at the same level, and it may be considered a complete solution to a level i problem or a partial solution to the whole problem.

The highest level plan, $\mathcal{S}^1$, is obtained by generative techniques, and every plan $\mathcal{S}^i$, for $i > 1$, is obtained by means of a hybrid process which interleaves hierarchical (function **REFINE**) and generative (function **GENERATE**) refinements. Function **REFINE** performs the step 1 described in Section 3.2, that is, it decomposes every compound action α into a set of literals by any of its existing methods and then it inserts these literals in the plan. Function **GENERATE** performs the step 2 described in Section 3.2 taking into account the validity of decompositions described in Section 3.3. That is, it satisfies every literal by means of generative techniques and establishes a correct modularization relationship between every compound action and its subactions. Function **GENERATE** is based on a non-hierarchical POCL planner described in [8].

The process ends when all of the actions in the current plan are primitive actions and that plan has been correctly modularized.

Next, we will discuss the key issues of backtracking between levels of abstraction, and the consistency of causal structure through levels of abstraction.

3.5 Backtracking between abstraction levels.

In the algorithm described in Figure 3, backtracking between abstraction levels is done when there is no possibility to generate a plan $\mathcal{S}^i$ modular and causally correct with respect to $\mathcal{S}^{i-1}$ as seen in Section 3.3. It must be said that, taking into account the type of solutions obtained by our approach, backtracking between abstraction levels has no negative effect over the completeness of the algorithm, as discussed in Section 2.4.

A solution is not a primitive plan, as in most HTN approaches, but a complete hierarchical and modular plan, that is a sequence of plans $\{\mathcal{S}^1 \ \ldots \ \mathcal{S}^n\}$, where the lowest level plan is completely made up of primitive actions, every plan $\mathcal{S}^i$ is a partial solution at level i and it has a correct modularization relationship with respect to the plan at the next abstraction level $i + 1$. Hence, if there is no partial solution at some abstraction level, then there is no possibility to obtain such a complete hierarchy of valid plans, despite of the existence of any later primitive partial solution. Therefore, in these situations, the need to obtain a complete hierarchy of plans leads to backtrack to the previous abstraction level to continue the search for alternative refinements. Given that a solution is a complete hierarchy of plans, as opossite to most HTN approaches in which a solution is a primitve plan, a backtracking criterium based on the contrapositve of USP does not put in risk the completeness of the algorithm.

As can be seen, the usefulness of the USP to backtrack between abstraction levels in real-world planning depends on how a correct solution is defined. However, our experience in solving real-world planning cases points to an interesting role of this property in domain validation, which will be outlined in Section 4.

Finally, next section will be devoted to show the monotonicity of the causal structure through different abstraction levels.

3.6 Consistency of causal structures through abstraction levels.

In our hybrid model, a plan $\mathcal{S}^i$ inherits the causal link structure generated in the plan $\mathcal{S}^{i-1}$ by means of a dynamic process at planning time. This process is based on reusing the high level structure of causal links, taking into account the inheritance rules shown in Figure 4.

RULE 1:	**RULE 2:**
A causal link $[\alpha \xrightarrow{r} \beta, \gamma]$ from $\mathcal{S}^{i-1}$, such that $f_{Ag(\alpha)}(r) \neq \text{NIL}$, has associated a hybrid causal link $[a_\alpha \xrightarrow{l} \beta, \gamma]$ in $\mathcal{S}^i$ if i) $l \in f_{Ag(\alpha)}(r)$ ii) $a_\alpha \xrightarrow{Sat} l$	A hybrid causal link $[a_\alpha \xrightarrow{l} \beta, \gamma]$ from $\mathcal{S}^i$ has associated a causal link $[a \xrightarrow{l} b, c]$ in $\mathcal{S}^i$ if i) $b \in Sub(\beta)$ ii) $a \xrightarrow{Sat} l$, $l \in Req(b)$

Figure4. Inheritance rules of causal links

This monotonic inheritance of higher-level causal structures is used to detect and solve hybrid threats. These hybrid threats represent a violation of a causal link established at level i by an action at level $i+1$ and they will become a classic threat at level $i+1$ when all of the actions at level i had been decomposed. Hence, unsolvable hybrid threats, which will later become unsolvable classic threats, may be used for an early detection of dead-end branches and backtracking before all of the decompositions have been completed.

In summary, this inheritance of causal structures not only provides a greater expressiveness, since it is the basis of the dynamic decomposition mechanism, but it also provides a greater efficiency, since it allows for an early detection of some unsolvable threats exploiting the knowledge synthesized at previous levels.

4 Conclusions

In this work we have shown some advances in hierarchical planning which overwhelm a set of shortcomings on known hierarchical models, and we have presented a discussion on the adequacy of hierarchical planning properties for real-world planning, on the basis of a hybrid planning approach developed to solve real-world problems.

Space precludes to justify our proposal with an appropriate experimentation. It must be said that our model has been extensively used for the automatic synthesis of hierarchical control programs for manufacturing systems in which simple domain descriptions and modularity relations play a very important role. This experimentation is being carried out in close collaboration with experts on industrial domains within a research project, and it will appear in other paper in preparation.

In any case, all of the improvements which have been proposed have a common basis: the abstraction formalism based on the *articulation function*. This abstraction formalism allows for a high accomplishment of formerly enumerated requirements (see Table 1 for more details):

- Simplicity: it reduces human effort on syntactic constraints observance and decision making, at domain description stage.
- Autonomy and expressiveness: it provides the basis for a dynamic action decomposition process performed at planning time and it allows to obtain "ready-to-use" plans which describe the behaviour of a compositional hierarchy of agents.
- Soundness and completeness: it also provides the basis for dynamically checking the semantic correctness through plan levels with different semantic granularity, while preserving completeness.
- Efficiency: the inheritance of causal structures provides the means for an intensive reuse of the knowledge embedded in higher levels. This reuse of higher level knowledge produces a great benefit on the efficiency and, in addition, it allows for a more understandable planning process from the point of view of a human.

On the other hand, we have also discussed the adequacy of known hierarchical planning properties for real-world problem solving. In summary, the DRP is too restrictive and it leads to reduce expressiveness for real world problems. However, we have shown that the Monotonic Property is very useful for real-world planning. In particular, we have introduced a new mechanism which monotonically inherits causal structures between plans at different abstraction levels, which are represented at different semantic granularity by changing the representation

language. Finally, we have shown that USP may be used to bactrack between abstraction levels without putting in risk the completeness of our hybrid algorithm. This is because the type of solution needed in our approach is not a primitive plan, but a complete hierarchy of plans. However, the accomplishment of this property is still useful during domain description process, from a knowledge engineering perspective.

Although the completeness of the planning algorithm could be formally proven, it is always possible to describe a hierarchical domain with no partial solution at some abstract level, but with a partial solution at ground level. This means that the USP is an inherent property to every hierarchical domain, whose accomplishment is necessary to prevent the description of bad domains. However, it could be argued that there could be some feedback between the planning process and the domain description process, such that the planner could be able to detect and suggest some domain coding errors. This strategy could be implemented by means of a mixed initiative planning process, such that the planner would inform the expert about the circumstances which invalidate the execution of actions at a given level (for instance, a unsolvable threat could mean a deadlock between two agents). This situation would interrupt the hierarchical refinement process until a redefinition of conditions at that abstraction level had been done.

This mixed initiative process could provide the basis for a dynamic knowledge validation, at different levels of abstraction, during the planning process. But it must be said that such as mixed initiative redefinition process, at a single level of abstraction, is only operative with an abstraction formalism like the one presented here, since it needs a completely differentiated representation at every abstraction level, otherwise this redefinition should also be propagated into lower level representations.

At present we are developing an intelligent digital assistant for the interactive development of industrial control programs on the basis of this hybrid model.

	Abstraction formalism	**Decomp. mechanism**	**Decomp. validity**	**Backtracking**	**C.Links Inherit.**
Non-Dec. Models	Literal Oriented Abstraction. Same set of literals for every level. Hierarchies may be self-generated.	No	No	Always correct. Satisfy USP,DRP.	Satisfy Monotonic Prop. Direct inheritance.
Decomp. Models	Action Oriented Abstraction. Same set of literals for every level. Syntactic constraints about actions literals.	Reduction schemes. Need extra knowledge. Total or partial description.	It is restricted by hand at domain description.	USP depends on UMS.	Directly inherited or restricted at domain description.
Hybrid Model	Behavior Oriented Abstraction. Granularity levels of knowledge. No syntactic constraints about actions literals.	Default expansion method. No need of extra knowledge.	All the decision taken on planning time. Correct modularization relationship.	Correct, due to solution features. Mixed initiative may help to accomplish USP.	Hybrid c.links are built at planning time. Causal inheritance with change of language.

Table1. Characteristics of hierarchical planinng

References

1. R.C. Arkin. *Behavior-Based Robotics.* MIT press, Cambridge, MA, 1998.
2. F. Bacchus and Q. Yang. The downward refinement property. In *Proceedings of IJCAI 91*, pages 286–292, 1991.
3. F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1994.
4. C. Backstrom and P. Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In *Proc. of IJCAI 95*, pages 1599–1604, 1995.
5. R. Bergman and W. Wilke. Building and refining abstract planning cases by change of representation language. *JAIR*, 3:53–118, 1995.
6. L. Castillo, J. Fdez-Olivares, and A.González. A hybrid hierarchical/operator-based planning approach for the design of control programs. In *Proceedings of ECAI'2000. Workshop on Planning, Scheduling and Design. PUK'2000.*, 2000.

7. L. Castillo, J. Fdez-Olivares, and A. González. A three-level knowledge-based system for the generation of live and safe petri nets for manufacturing systems. *Journal of Intelligent Manufacturing*, 11(6):559–572, 2000.
8. L. Castillo, J. Fdez-Olivares, and A. González. Mixing expresiveness and efficiency in a manufacturing planner. *To appear in Journal of Experimental & Theoretical Artificial Intelligence (JETAI)*, 2001.
9. S. Chien, R. Hill, Jr. X. Wang, and H. Mortenson T. Estlin, K. Fayyad. Why real-world planning is difficult: a tale of two applications. In M. Ghallab and A. Milani, editors, *New directions in AI Plannig*, pages 287–298. IOS Press, 1996.
10. K. Currie and A. Tate. O-Plan: Control in the open planning architecture. In *BCS Expert systems conference*, 1985.
11. K. Erol, J. Hendler, and D. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS-94*, 1994.
12. T.A. Estlin, S.A. Chien, and X. Wang. An argument for a hybrid HTN/operator based approach to planning. In *Recent Advances in AI Planning.Proc. of 4th European Conference on Planning ECP'97*, pages 182–194, 1997.
13. M. Fox. Natural hierarchical planning using operator decomposition. In *Recent Advances in AI Planning.Proc. of 4th European Conference on Planning ECP'97*, pages 195–207, 1997.
14. F. Giunchiglia. Using abstrips abstractions – where do we stand? *Artificial Intelligence Review*, 13:201–213, 1999.
15. J. Hobbs. Granularity. In *IJCAI 85*, pages 432–435, 1985.
16. C. Knoblock. AI Planning systems in the real world. *IEEE Expert*, pages 4–12, 1996.
17. C. A. Knoblock. *Generating Abstraction Hierarchies*. Kluwer Academic Publishers, 1993.
18. D. E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.
19. S.Viswanathan, C.Johnsson, R.Srinvivasan, V.Venkatasubramanian, and K.E. Arzen. Automating operating procedure synthesis for batch processes. part I: Knowledge representation and planning framework. *Computers and Chemical Engineering*, 22:1673–1685, 1998.
20. D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.
21. D. E. Wilkins. *Practical planning: Extending the classical AI planning paradigm*. Morgan Kaufmann, 1988.
22. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, October 1995.
23. Q. Yang. *Intelligent Planning. A decomposition and Abstraction Based Approach*. Springer Verlag, 1997.
24. Q. Yang, J. Tenenberg, and S. Woods. On the implementation and evaluation of ABTWEAK. *Computational Intelligence*, 12:307–330, 1996.
25. R.M. Young, M.E. Pollack, and J.D. Moore. Decomposition and causality in partial order planning. In *Proceedings of the Second International Conference on AIPS*, 1994.