Artificial Intelligence 222 (2015) 124–156

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

HTN planning: Overview, comparison, and beyond

Ilche Georgievski *, Marco Aiello

Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 7 April 2014
Received in revised form 3 February 2015
Accepted 6 February 2015
Available online 11 February 2015

Keywords:
AI planning
Hierarchical task networks
Web service composition

ABSTRACT

Hierarchies are one of the most common structures used to understand and conceptualise the world. Within the field of Artificial Intelligence (AI) planning, which deals with the automation of world-relevant problems, Hierarchical Task Network (HTN) planning is the branch that represents and handles hierarchies. In particular, the requirement for rich domain knowledge to characterise the world enables HTN planning to be very useful, and also to perform well. However, the history of almost 40 years obfuscates the current understanding of HTN planning in terms of accomplishments, planning models, similarities and differences among hierarchical planners, and its current and objective image. On top of these issues, the ability of hierarchical planning to truly cope with the requirements of real-world applications has been often questioned. As a remedy, we propose a framework-based approach where we first provide a basis for defining different formal models of hierarchical planning, and define two models that comprise a large portion of HTN planners. Second, we provide a set of concepts that helps in interpreting HTN planners from the aspect of their search space. Then, we analyse and compare the planners based on a variety of properties organised in five segments, namely domain authoring, expressiveness, competence, computation and applicability. Furthermore, we select Web service composition as a real-world and current application, and classify and compare the approaches that employ HTN planning to solve the problem of service composition. Finally, we conclude with our findings and present directions for future work. In summary, we provide a novel and comprehensive viewpoint on a core AI planning technique.

1. Introduction

Hierarchical Task Network (HTN) planning is an Artificial Intelligence (AI) planning technique that breaks with the tradition of classical planning [1]. The basic idea behind this technique includes an initial state description, a task network as an objective to be achieved, and domain knowledge consisting of networks of primitive and compound tasks. A task network represents a hierarchy of tasks each of which can be executed, if the task is primitive, or decomposed into refined subtasks. The planning process starts by decomposing the initial task network and continues until all compound tasks are decomposed, that is, a solution is found. The solution is a plan which equates to a set of primitive tasks applicable to the initial world state.

Beside being a tradition breaker, HTN planning appears to be controversial as well. The controversy lies in its requirement for well-conceived and well-structured domain knowledge. Such knowledge is likely to contain rich information and

* Corresponding author.
E-mail addresses: i.georgievski@rug.nl (I. Georgievski), m.aiello@rug.nl (M. Aiello).

http://dx.doi.org/10.1016/j.artint.2015.02.002
0004-3702/

guidance on how to solve a planning problem, thus encoding more of the solution than was envisioned for classical planning techniques. This structured and rich knowledge gives a primary advantage to HTN planners in terms of speed and scalability when applied to real-world problems and compared to their counterparts in classical world.

The biggest contribution towards this kind of "popular" image of HTN planning has emerged after the proposal of the Simple Hierarchical Ordered Planner (SHOP) [2] and its successors. SHOP is an HTN-based planner that shows efficient performance even on complex problems, but at the expense of providing well-written and possibly algorithmic-like domain knowledge. Several situations may confirm our observation, but the most well-known is the disqualification of SHOP from the International Planning Competition (IPC) in 2000 [3] with the reason that the domain knowledge was not well-written so that the planner produced plans that were not solutions to the competition problems [2]. Furthermore, the disqualification was followed by a dispute on whether providing such knowledge to a planner should be considered as "cheating" in the world of AI planning [4].

SHOP's style of HTN planning was introduced by the end of 1990s, but HTN planning existed long before that. The initial idea of hierarchical planning was presented by the Nets of Action Hierarchies (NOAH) planner [5] in 1975. It was followed by a series of studies on practical implementations and theoretical contributions on HTN planning up until today. We believe that the fruitful ideas and scientific contribution of nearly 40 years must not be easily reduced to controversy and antagonism towards HTN planning. On the other hand, we are faced with a situation full of fuzziness in terms of difficulty to understand what kind of planning style other HTN planners perform, how it is achieved and implemented, what are the similarities and differences among these planners, and finally, what is their actual contribution to the creation of the overall and possibly objective image of HTN planning. The situation cannot be effortlessly clarified because the current literature on HTN planning, despite being very rich, reports little or nothing at all on any of these issues, especially in a consolidated form.

In addition to these issues, we observe the applicability of AI planning techniques as an ultimate goal of their development. We are especially interested in novel and real-world domains which may require reconsidering established techniques. The growing trend on other than classical and synthetic domains leads to the need for algorithms and systems that reflect planning better and more closely to the real world. This perspective gives another view to the abilities of HTN planners (and HTN planning in general) to cope with various properties of an application in the real world.

We aim to consolidate and synthesise a number of existing studies on HTN planning in a manner that will clarify, categorise and analyse HTN planners, and allow us to make statements that are not merely based on contributions of a single HTN planner. We also hope to rectify the perception of HTN planning as being controversial and antagonistic in the AI planning community. Finally, we choose a non-traditional, dynamic and uncertain application domain to ascertain HTN planning with respect to various domain characteristics.

1.1. Approach

We take a framework-based approach to accomplish our objectives. We perceive a framework as an abstract and logical structure that we establish in need for support and guidance on the development of our study. Since we inspect HTN planning from four different perspectives, we provide a framework for each perspective. All four frameworks form what we refer to as a *pie of frameworks*. The pie of frameworks serves as a central and unifying point of our study organisation and presentation flow.

The first piece of the pie is a *theoretical framework* for HTN planning upon which we later build two models of HTN planning. The second slice is a *conceptual framework* for HTN planning that provides clarification of different concepts related to the search space, and context for interpretation of HTN planners. The next piece is an *analytical framework* for HTN planners that enables us to go deeper and beyond dry descriptions about HTN planners. The last piece might appear to taste differently than the other three, but it still has the flavour of HTN planning. The *application framework* concerns the application domain we choose to observe, and it helps us to analyse different studies in an organised and unified way, and possibly to identify points where HTN planning behaves as expected or can be further improved.

1.2. Inclusion of HTN planners and studies

We make use of two inclusion criteria for planners and studies. The *inclusion criterion of planners* relies on the inspection of existing literature for suggestions on HTN planners that have risen to some degree of prominence. For example, we accept the list of "best-known domain-independent HTN planning systems" as provided in [1]. In addition to those five suggested planners, we include two more. The complete list of HTN planners participants in our study is the following one[1]:

- NOAH, the first HTN planner emerged in mid-1970s [6,5],
- Nonlin that appeared one year later [7,8],
- System for Interactive Planning and Execution (SIPE) and SIPE-2 introduced in 1984 and 1990, respectively [9],
- Open Planning Architecture (O-Plan) and its successor O-Plan2 in 1984 and 1989, respectively [10,11],

[1] Henceforth, we refer only to the most recent version of each planner.

- Universal Method Composition Planner (UMCP) introduced in 1994 [12],
- SHOP and its successor SHOP2 that appeared in 1999 and 2003, respectively [2,13], and
- SIADEX that emerged in 2005 [14].

The *inclusion criterion of studies* relies on the theoretical contribution of a study with respect to HTN planning in general, and theoretical and practical issues of each chosen planner separately. The criterion is based on the coverage a study gives, which may include information that ranges from a general discussion of techniques and approaches, peculiar matters, such as task interactions and condition types, relevant to our conceptual framework, to properties, such as domain authoring, expressiveness and competence, that may be a part of the analytical framework. Finally, we include an extensive number of studies that employ HTN planning for the purpose of our application domain, that is, Web service composition.

There are several more research areas related to HTN planning that, however, we exclude from the present treatment (*e.g.,* [15–20]). The reason behind that is the underlying frameworks are substantially different from what we propose here, though HTN planning is used. This is the case of distributed planning or landmarks, for instance. The issue of learning domain knowledge is also relevant, but orthogonal to our approach. We provide an overview of related work on HTN planning that though does not fit the present treatment in Section 5.

1.3. Web service composition

We choose the domain of Web services as a non-traditional and real-world application [21]. *Web services* are software components that implement specific business logic, and are distributed over a network, typically the Web, to be used as Web resources for machine-to-machine interaction. For instance, travel agencies may provide a number of Web services, such as booking a flight ticket, reserving a hotel, renting a car, or organising sightseeing. The interaction is usually initiated by a client request which has to be satisfied by the functionalities that Web services offer. However, in cases when no single service can accomplish the request, a composition of several Web services might give a value-added functionality, and provide a way to request satisfaction. For example, a service to arrange a complete trip to some tourist destination might be of an exceptional use to the commercial travel agencies, and thus, it will not be offered as a Web service.

The research community of Web service composition (WSC) focuses on developing tools, techniques and intelligent systems able to return a composition with the best possible quality-of-service values [22], as it is the case with the Web Service Challenge [23,24]. In the Web Service Challenge, systems (*e.g.,* [25–27]) generate two compositions, namely a service composition with the lowest response time, and a service composition with the highest throughput. The response time expresses the delay between the time a request is received by a Web service and the time a reply to the request is sent, and the throughput expresses the amount of requests that a Web service can handle in a given time unit. On the other hand, the AI community tries to automate the process of Web service composition by viewing the composition problem as a planning problem [28–38]. The general assumption is that planning operators correspond to functionalities of Web services, while the goal, in the simplest example, is aggregated from the client request. In addition, the environment of Web services already shows some propensity to composite or hierarchical representation. For example, a composite service could describe a hotel reservation by searching for a hotel, registering on the hotel Web site, logging onto the hotel Web site, and finally, the actual booking of the hotel. Among AI planning techniques, HTN planning is well suitable for domains in which some hierarchical representation is desirable or known in advance, domains that encourage complex and composed constructs, and domains of large size. These indicators suggest ideal conceptual matchmaking between HTN planning and Web service composition, but are also a computational challenge. Thus, continuing with our example, if the client's objective is not only to reserve a hotel, but to arrange a complete trip, which includes also booking a flight, renting a car, and sightseeing, then, definitely, the complexity of services and their composition becomes an interesting and challenging task.

The environment of Web services offers more exciting challenges that make the effective selection and composition of services far from being plain and straightforward planning processes. In particular, Web services exist in a *dynamic* environment in which the availability of services is not guaranteed. This behaviour reflects the availability of information which, on the other hand, is assumed by planners to be complete and obtainable before the planning process is initiated. Furthermore, the environment of Web services favours techniques that are able to deal with *uncertainty* in terms of 1) incomplete information about the initial state; 2) uncertainty over the many possibilities for completion of missing information by invoking some sensing services at planning and/or execution time; 3) non-determinism caused by failed invocations of Web services (*e.g.*, renting a car is not viable at the moment of invocation), a service not responding at all, a service yielding undesired outcome (*e.g.*, booking a flight provides only business-class tickets); 4) services that show unexpected behaviour (*e.g.*, Byzantine failure). Moreover, *complex goals* possibly in the form of a workflow or conditioned with some organisational regulations or augmented with user preferences are the norm rather than exception. Finally, the *high cardinality* of the set of Web services available on the Web implies a large space to be searched by a planner.

1.4. Organisation

The remainder of the paper is organised as follows. Section 2 contains the theoretical and conceptual frameworks. Based on the ideas presented in these frameworks, we propose and formalise two models of HTN planning. We review the planners corresponding to an HTN model with respect to the conceptual framework. Section 3 provides details on each HTN planner

separately. Section 4 goes deeper into the application area of Web service composition accomplished by HTN planning. Section 5 includes a discussion on related work which is though not central to the present overview of HTN planning. Finally, Section 6 concludes the paper with considerations and directions for future work.

2. HTN planning: theory and concepts

HTN planning has been formalised in several studies, such as [39,2,1,40]. These formalisations include similar definitions of HTN terms, appropriate to the needs for their model of HTN planning. Based on these existing theories, we describe the first piece of our pie of frameworks, that is, the theoretical framework for HTN planning. In this framework, we keep the definitions of HTN terms high level. Further in the paper, we provide specific definitions of the terms that are characteristic for the model of HTN planning being analysed. The purpose of the theoretical framework is twofold. Firstly, it provides a basic understanding of HTN planning. Secondly, it determines and defines the focus of categorisation of HTN planning that we propose.

2.1. Theoretical framework

The theoretical framework is composed of a planning language, tasks, operators, task networks, methods, planning problem and solution. The *HTN planning language* is a first-order language that contains several mutually disjoint sets of symbols. As usual, a *predicate*, which evaluates to true or false, consists of a predicate symbol $p \in P$, where P is a finite set of predicate symbols, and a list of terms $\tau_1, \ldots, \tau_k$. A *term* is either a constant symbol $c \in C$, where C is a finite set of constant symbols, or a variable symbol $v \in V$, where V is an infinite set of variable symbols. We denote the set of predicates as Q. A predicate is *ground* if its terms contain no variable symbols. A *state* $s \in 2^Q$ is a set of ground predicates in which the *closed-world assumption* is adopted, that is, all and only the predicates that are true are specified in the state. We define a *primitive task* as an expression $t_p(\tau)$, where $t_p \in T_p$ and T_p is a finite set of primitive-task symbols, and $\tau = \tau_1, \ldots, \tau_k$ are terms. A primitive task is represented by a planning operator.

Definition 1 *(Operator).* An *operator* o is a triple $(p(o), pre(o), eff(o))$, where $p(o)$ is a primitive task, and $pre(o) \in 2^Q$ and $eff(o) \in 2^Q$ are preconditions and effects, respectively. The subsets $pre^+(o)$ and $pre^-(o)$ denote positive and negative preconditions of o, respectively.

A transition from a state to another one is accomplished by an instance of an operator whose precondition is a logical consequence of the current state. An operator o is *applicable* in state s, if $pre^+(o) \subseteq s$ and $pre^-(o) \cap s = \emptyset$. Applying o to s results in the state $s[o] = (s \setminus eff^-(o)) \cup eff^+(o) = s'$, where $eff^-(o)$ and $eff^+(o)$ are negative and positive effect of o, respectively.

A *compound task* is an expression $t_c(\tau)$, where $t_c \in T_c$ and T_c is a finite set of compound-task symbols, and $\tau = \tau_1, \ldots, \tau_k$ are terms. We refer to the union of the sets of primitive-task and compound-task symbols as a set of task names T_n. The following two definitions are further complemented for the respective model of HTN planning.

Definition 2 *(Task network).* A *task network* tn is a pair (T, ψ), where T is a finite set of tasks, and ψ is a set of constraints.

Constraints in ψ specify restrictions over T that must be satisfied during the planning process and by the solution. We refer to a task network over T_p as a *primitive task network*. The set of all task networks over T_n is denoted as TN.

Definition 3 *(Method).* A *method* m is a pair $(c(m), tn(m))$, where $c(m)$ is a compound task, and $tn(m)$ is a task network.

Definition 4 *(HTN planning problem).* An *HTN planning problem* $\mathcal{P}$ is a tuple (Q, O, M, tn_0, s_0), where

- Q is a finite set of predicates,
- O is a finite set of operators,
- M is a finite set of methods,
- tn_0 is an initial task network,
- s_0 is an initial state.

An operator sequence $o_1, \ldots, o_n$ is *executable* in s if there is a sequence of states $s_0, \ldots, s_n$ (also called a *trajectory*) such that $s_0 = s$ and o_i is applicable in s_{i-1} and $s_{i-1}[o_i] = s_i$ for all $0 \leq i \leq n$. Given an HTN planning problem $\mathcal{P}$, a *solution* to $\mathcal{P}$ is an operator sequence executable in s_0 by decomposing tn_0. The way of producing such a sequence is defined in the following sections.

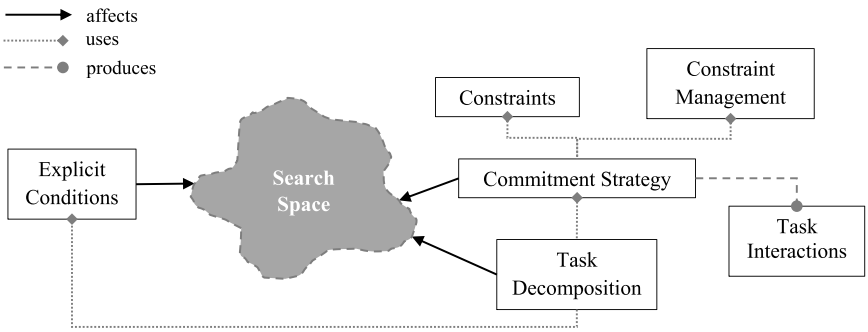


Fig. 1. Conceptual framework.

2.2. Conceptual framework

Literature reports vague information on HTN planning (and planners) especially in the early stages of hierarchical planning. First, it is difficult to understand the ideas and concepts used and how they are adapted for the purpose of HTN planning. The situation later improved with the evolution of HTN planners and the attempts at formalisation. Then, different models of HTN planning could be found at some point of the evolution, and at first glance, the model distinction seems not that obvious and comprehensible.

The motivation for the second slice of our pie lies exactly in these issues. We clarify them by designing and describing a conceptual framework shown in Fig. 1. This framework is less formal (compared with the theoretical framework) and based on specific concepts derived from empirical observation. We start by providing basic and general enough descriptions of concepts that characterise different HTN planners and cover most of their important features. The concepts are placed within a logical and sequential design as much as possible. In our framework, the key concept is the search space to which other concepts are related and interconnected in various ways. The purpose of the conceptual framework is manifold. Firstly, it clarifies concepts and proposes relationships among them. Secondly, it provides context for interpreting the findings presented in the paper, and helps in explaining observations. Finally, the view of key concepts enables us to categorise HTN planning based on the space the search is performed in.

2.2.1. Task decomposition

Given a task network *tn*, a *task decomposition* chooses a task *t* from *tn* and, if *t* is primitive and suitable for the current state *s*, the task decomposition applies *t* to *s*. Otherwise, all the methods are analysed that contain *t* as a part of their definition. Assuming that a set of methods is found, a non-deterministic choice of a method *m* is made, and *t* is replaced with the task network associated with *m*. Finally, the newly composed task network is checked against any constraint-related violation and modified, if necessary.

Task decompositions can be divided into three styles based on the representation of task networks in terms of task ordering, and the way of forming new task networks during decomposition. The first one is *totally ordered task decomposition* (TOTD). It follows the assumption of total order on task networks so as when a task is decomposed, the new task network is created in such a way that newly added tasks are totally ordered among each other and with respect to the tasks of the existing task network. Sometimes we refer to the HTN planning that uses this style as totally ordered HTN planning. The second style is *unordered task decomposition* (UTD) that relaxes the requirement of totally ordered task networks. That is, tasks can be totally ordered or unordered with respect to each other (but no tasks in parallel are allowed). When a task is decomposed, new task networks are created in such a way that newly added tasks are interleaved with the tasks of the existing task network until all permissible permutations are exhausted. Here as well, we refer to the HTN planning that embodies this style as unordered HTN planning. The last style is *partially ordered task decomposition* (POTD) that allows the existence of a partial order on tasks. When a task is decomposed, the tasks in the newly created network can be ordered in parallel whenever possible (with respect to the constraints). The HTN planning that uses this style is referenced as partially ordered HTN planning.

2.2.2. Constraints

Definition 2 suggests that constraints are found in a task network, but constraints can be also added during the planning process in order to resolve inconsistencies. HTN planners deal with several types of constraints, and most of them can be interpreted as in [41]. Namely, there are three interpretations. First, we meet a constraint that implies commitments about partial descriptions of state objects. Another type of constraint refines variable bindings if a certain variable binding does not satisfy some condition. Last, there is a constraint that expresses the relations between variables in different parts of a task network.

Commitment strategy As with most of the other AI planners, HTN planners also need to make two decisions on constraints. The first one is on constraints for binding variables, while the second decision is on constraints for ordering tasks in a task

network. HTN planners use mainly two strategies for when and how to make these decisions. The first strategy manages constraints in compliance with the *least-commitment strategy* so that task ordering and variable bindings are deferred until a decision is forced [42]. The second strategy handles constraints according to the *early-commitment strategy* so that variables are bound and operators in the plan are totally ordered at each step of the planning process. Planners employing this strategy greatly benefit from the possibility of adopting forward chaining where chaining of operators is achieved by imposing a total order on (some) plan tasks. The total ordering ensures that neither the current task to be added to the plan can interfere with some earlier operator's preconditions or effects, nor a later operator can interfere with current task's preconditions or effects.

Task interaction Depending on the commitment strategy chosen, and especially in the case of the least-commitment strategy, an inevitable consequence is the interaction among tasks in a given task network. Generally, an *interaction* is a connection between two tasks (or parts) of a task network in which these tasks (or parts) have some effect on each other. Based on this effect, we divide interactions into two categories. The first category, called *harmful interactions* (also threats or flaws), introduces conflicts among different parts of a task network that threaten its correctness. HTN planners consider harmful interactions individually, and provide rather intuitive descriptions. In the following list, we provide a general description of harmful interactions found in HTN planners.

- *Deleted-condition interaction* – appears when a primitive task in one part of a task network deletes an expression that is a precondition to a primitive task in another part of that task network.
- *Double-cross interaction* – appears when an effect of each of two conjunctive primitive tasks deletes a precondition for the other. That is, an effect of the first task deletes a precondition of the second primitive task, and an effect of the second task deletes a precondition of the first task.
- *Resource interaction* – appears in two situations, and it is subdivided accordingly. A *resource-resource interaction* is similar to the deleted-condition interaction, while a *resource-argument interaction* occurs when a resource in one part of a task network is used as an argument in another part of that task network.

The second category, called *helpful interactions*, refers to situations when one part of a task network can make use of information associated with another part in the same task network. The detection of these interactions implies the possibility for a planner to generate better-quality task networks and solutions. In fact, some tasks can be merged together, which eliminates task redundancy and potentially optimises the cost of the solution [43]. The following list contains general descriptions of helpful interactions.

- *Placeholder replacement* – appears when a real value already exists for a particular formal object. HTN planners allow tasks with variables to be inserted into a task network. If there is no specific value to be chosen for a particular variable choice, a so-called formal object is created to bind the variable [6]. The formal object is simply a placeholder for some entity unspecified at that point.
- *Phantomisation* – appears when some goal is already true at the point in a task network where it occurs. In the descriptions of some HTN planners, the term 'goal' is interchangeably used with the term 'precondition'. In fact, if some task precondition is not satisfied, it is inserted as a goal to be achieved.
- *Disjunct optimisation* – appears in disjunctive goals when one disjunctive goal is "superior to the others by the nature of its interaction" with the other tasks in a task network [6].

Constraint management Task interactions can be solved by posting various types of constraints onto a task network. This *constraint posting* is also known as conflict resolution [44] or critics [7,45]. HTN planners do not provide a general approach for handling interactions, thus each of the above interactions has its own resolution method. However, those methods are based on well-known operations on constraints generally described elsewhere, *e.g.*, [41]. We briefly describe some operations in the context of HTN planning.

The most basic operation is *constraint satisfaction* which happens when an HTN planner searches for a variable binding that satisfies the given constraints, and guarantees the consistency of, for example, a set of ordering constraints over a task network. *Constraint propagation* enables adding or retracting constraints to and from a task network. Variable constraints in one part of a task network can be propagated based on variable constraints in another part of that task network. With respect to ordering constraints, propagation is used when a linking process is performed. When some task interferes with another task, the *linking process* records a causal link, that is, a three-element structure of two pointers to tasks t_e and t_p, and a predicate q which is both an effect of t_e and a precondition of t_p. The phantomisation interaction is practically achieved by the linking process. Phantomisation of a task t with an effect e is accomplished by treating e as achieved, and finding an existing task t' in the task network that achieves the same effect e. If task t' is found, a constraint (t', e, t) is added to the task network to record the causal relation.

The last operation is different in that it does not happen during the planning process. *Constraint formulation* can be taken into account when modelling HTN domain knowledge, especially when the domain author is aware in advance of some possible impasse situations. By posting constraints as control information into the domain knowledge, the planner can gain on efficiency by refining the search space [46,47]. In some HTN planners, the phantomisation of a task is achieved by an

explicit encoding in the domain knowledge. Those planners handle the phantomisation of a rather recursive task by taking into account an alternative method that encodes the base case explicitly by a 'do-nothing' operation. These planners can be extended to infer such situations as it is accomplished in [48].

2.2.3. Explicit conditions

HTN planners depend on the quality of the domain knowledge to restrict and guide the search. The domain author is the one who has the responsibility of giving the information about the guidance in the search space. One way to represent such information is by using explicit conditions. We provide a general description of conditions found in HTN planners.

- *Supervised condition* – accomplished within a compound task. The condition may be satisfied either by an intentional insertion of a relevant effect earlier in the processing of the task network, or by an explicit introduction of a primitive task that will achieve the required effect. Generally, only this condition should allow further decompositions to be made and, since it may be included for the achievement of the condition invocation of another task, this condition corresponds to preconditions in STRIPS-like planning systems.[2]
- *External condition* – must be accomplished at the required task, but under the assumption that it is satisfied by some other task from the task network. An external condition can be seen as a sequencing constraint.
- *Filter condition* – decides on task relevance to a particular situation. In the case of method relevance to a certain task decomposition, this condition reduces the branching factor by eliminating inappropriate methods.
- *Query condition* – accomplishes queries about variable bindings or restrictions at some required point in a task network.
- *Compute condition* – requires satisfaction by information coming only from external systems, such as a database.
- *Achieve condition* – allows expressing goals that can be achieved by any means available to a planner.

2.2.4. Search space

So far we described concepts that affect the structure of the space to be searched. Next, we intuitively describe two structures of search spaces created by HTN planners. The first type of space consists of task networks and task decompositions as evolutions from one task network to another. Given an HTN planning problem $\mathcal{P}$, at the beginning of the search, a task decomposition is imposed on the initial task network tn_0, and the process continues by repeatedly decomposing tasks from a newly created task network until a primitive task network is produced. A linearisation of this primitive task network executable in the initial state s_0 represents a solution to $\mathcal{P}$.

The second type of search space is a subset of the state space. The subset consists of explicitly described states restricted by task decompositions. As in the classical state space, the search begins in s_0 with an empty plan, but instead of searching for a state that will satisfy the goal, the search is for a state that will accomplish tn_0. In particular, if a task from the task network is compound, the task decomposition continues on the next decomposition level, but in the same state. If the task is primitive, it is executed and the search continues into a successor state. This task is then added to the plan. When there are no more tasks in the task network to be decomposed, the search is finished. The solution to $\mathcal{P}$ is the plan containing a sequence of totally ordered primitive tasks.

Categorisation of HTN planners In the first type of search space, the initial task network is reduced to a primitive task network that constitutes a solution to the planning problem. At each point in the space, the task network can be seen as a partially specified plan until the search reaches the point where the task network is primitive and represents a solution plan. Thus, we employ the term *plan space* to refer to this type of search space. We refer to HTN planners that search in this plan space as *plan-based HTN planners*, and to the model of HTN planning as *plan-based HTN planning*. For the obvious reasons, we employ the term *state space* to refer to the second type of search space. We refer to HTN planners searching in this space as *state-based HTN planners*, and to the model of HTN planning as *state-based HTN planning*.

2.3. Plan-based HTN planning

We draw the formalism of plan-based HTN planning upon the work of Geier and Pascal [40]. With respect to Definition 2 of the theoretical framework, we complement a task network as follows.

Definition 5 *(Task network).* A *task network* tn is a triple (T, φ, ψ), where

- T is a finite and non-empty set of tasks,
- $\varphi : T \to T_n$ labels a task with a task name,
- ψ is a formula composed by conjunction, disjunction or negation of the following sets of constraints:
  - $\prec \subseteq T \times T$ is a strict partial order on T (irreflexive, transitive, asymmetric),
  - $\mapsto \subseteq V \times V \cup V \times C$ is a restriction on bindings of task network variables, and
  - $\vdash_{\prec} \subseteq T \times Q \cup Q \times T \cup T \times Q \times T$ is a partial order on tasks and state predicates.

[2] With the "STRIPS-like" term, we refer to planning languages that use representations similar to the STRIPS one [49], that is, an operator is composed of preconditions, a delete list, and an add list.

Since some task name can occur many times in one task network, task labelling enables identifying uniquely many occurrences of that task name. For example, $tn = (\{t_1, t_2, t_3\}, \{(t_1, t'), (t_2, t''), (t_3, t')\}, \emptyset)$ denotes that the task network consists of two tasks associated with task name t' and one task associated with t''.

A task network $tn = (T, \varphi, \psi)$ is *isomorphic* to $tn' = (T', \varphi', \psi')$, denoted as $tn \equiv tn'$, if and only if there exists a bijection $\beta : T \to T'$, such that

- for all $t, t' \in T$ it holds $(t, t') \in \prec$ if and only if $(\beta(t), \beta(t')) \in \prec'$,
- for all $v_1, v_2 \in V$ and $c \in C$ it holds $(v_1, v_2) \in \mapsto$ or $(v_1, c) \in \mapsto$ if and only if there exist $v_1', v_2' \in V$ and $c' \in C$ such that $v_1 = v_1'$, $v_2 = v_2'$ and $(v_1', v_2') \in \mapsto'$ or $v_1 = v_1'$, $c = c'$ and $(v_1', c) \in \mapsto'$,
- for all $t, t' \in T$ and $q \in Q$ it holds $(t, q) \in \vdash_{\prec}$ or $(q, t) \in \vdash_{\prec}$ or $(t, q, t') \in \vdash_{\prec}$ if and only if $(\beta(t), q) \in \vdash'_{\prec}$ or $(q, \beta(t)) \in \vdash'_{\prec}$ or $(\beta(t), q, \beta(t')) \in \vdash'_{\prec}$,

and $\varphi(t) = \varphi'(\beta(t))$.

Definition 6 *(Decomposition).* Let m be a method and $tn_c = (T_c, \varphi_c, \psi_c)$ be a task network. Method m *decomposes* tn_c into a new task network tn_n by replacing task t, denoted as $tn_c \xrightarrow{t,m}_D tn_n$, if and only if $t \in T_c$, $\varphi_c(t) = c(m)$, and there exists a task network $tn' = (T', \varphi', \psi')$ such that $tn' \equiv tn(m)$ and $T' \cap^{t,m} T \neq 0$, and

$$
\begin{aligned}
tn_n &:= ((T_c \setminus \{t\}) \cup T', \varphi_c \cup \varphi', \psi_c \cup \psi' \cup \psi_D) \text{ where} \\
\psi_D &:= \{(t_1, t_2) \in T_c \times T' \mid (t_1, t) \in \prec_c\} \cup \{(t_1, t_2) \in T' \times T_c \mid (t, t_2) \in \prec_c\} \cup \\
&\quad \{(q, t_1) \in Q \times T' \mid (q, t) \in \vdash_{\prec c}\} \cup \{(t_1, q) \in T' \times Q \mid (t, q) \in \vdash_{\prec c}\} \cup \\
&\quad \{(t_1, q, t_2) \in T' \times Q \times T' \mid (t, q, t_2) \in \vdash_{\prec c}\}
\end{aligned}
$$

Given an HTN planning problem $\mathcal{P}$, $tn_c \to_D^* tn_n$ indicates that tn_n results from tn_c by an arbitrary number of decompositions using methods from M.

Definition 7 *(Executable task network).* Given an HTN planning problem $\mathcal{P}$, $tn = (T, \varphi, \psi)$ is *executable* in state s, if and only if it is primitive and there exists linearisation of its tasks $t_1, \ldots, t_n$ that is compatible with ψ and the corresponding sequence of operators $\varphi(t_1), \ldots, \varphi(t_n)$ is executable in s.

Definition 8 *(Solution).* Let $\mathcal{P}$ be an HTN planning problem. A task network tn_s is a solution to $\mathcal{P}$, if and only if tn_s is executable in s_0, and $tn_0 \to_D^* tn_s$ for tn_s being a solution to $\mathcal{P}$.

Our definition of the plan space is similar to the definition of the decomposition problem space in [50]. Intuitively, a problem space is a directed graph in which task networks are vertices, and a decomposition of one task network into another task network by some method is an outgoing edge, under the condition that the initial task network belongs to the graph.

Definition 9 *(Plan space).* Given a (plan-based) HTN planning problem $\mathcal{P}$, a *plan space* PG is a directed graph $(\mathcal{V}, \mathcal{E})$ such that $tn_0 \in \mathcal{V}$, and for each $tn \to_D tn'$: $tn, tn' \in \mathcal{V}$ and $(tn, tn') \in \mathcal{E}$.

2.3.1. Review of plan-based HTN planners

While we refer to [51] for a detailed illustration of task decomposition, constraints and constraint-based techniques, and explicit conditions, we here summarise the results in a tabular form, where we use the following notation. If a cell contains '✓', then the planner supports or defines the respective element. A cell with '✗' indicates no support or definition of an element, while '–' denotes that the planner does not need to support or handle the respective element. If a cell is empty, then it means that the information was not available from the public literature.

Table 1 demonstrates the concept of a task decomposition as realised in plan-based HTN planners. Since the task decomposition depends on the representation of tasks and task networks, in the column **Hierarchical representation**, we provide insights into how primitive and compound tasks are represented (column **Mechanism for primitive and compound tasks**), and what a task network consist of (column **Task network**). In the column **Task decomposition**, we show 1) how the decomposition of a 'compound' task is accomplished, and how a 'primitive' task is applied (column **Process**); 2) the style of a task decomposition (column **Type**); and 4) whether a task network is checked against any constraint violation during task decomposition (column **Constraint check**).

Most plan-based HTN planners perform a task decomposition in a slightly different way than the general process described in Section 2.2.1. The main reason lies in the approach that these planners use to represent tasks. In fact, with the exception of UMCP, the rest of the planners support only a single structure to encode both primitive and compound tasks. Although it is not always clear what is the purpose of the respective structure or how exactly the task decomposition

Table 1
Task decomposition in plan-based HTN planners.

	Hierarchical representation		Task decomposition		
	Mechanism for primitive and compound tasks	Task network	Process	Type	Constraint check
NOAH	single for both: node (code)	network of nodes	decomposition: code application: code	POTD	✓
Nonlin	single for both: node (schema)	network of nodes	decomposition: schema application: schema	POTD	✓
SIPE-2	single for both: node (operator)	network of nodes	decomposition: operator application: – (effect deduction)	POTD	✓
O-Plan2	single for both: schema	network of schemas	decomposition: schema application: schema	POTD	✓
UMCP	separate for each	network of primitive and compound tasks	decomposition: compound task application: primitive task	POTD	✓

Table 2
Constraint-related concepts in plan-based HTN planners.

Commitment	NOAH	Nonlin	SIPE-2	O-Plan2	UMCP
Strategy	least	least	least	least	least (+other)
Backtracking	no	yes	partially	yes	yes
Interaction					
Deleted-condition	Resolve Conflicts	Linking Process	Solving Harmful Int.	TOME/GOST Man.	Resolution Method
Double-cross	Resolve Double Cross	–	Solving Harmful Int.	–	–
Resource	×	×	Resource Conflicts	Resource Util. Man.	×
Placeholder replacement	Use Existing Objects				
Phantomisation	Eliminate Redun. Prec.	Linking Process	Goal Phantomisation	Question Ans.	Domain Method
Disjunct optimisation	Optimise Disjuncts				–

Table 3
Condition types in plan-based HTN planners (adapted from [54]).

Condition	NOAH	Nonlin	SIPE-2	O-Plan2	UMCP
Supervised	precondition	supervised	protect-until	supervised	goal task
External	×	unsupervised	external-condition	unsupervised	state constraints/external
Filter	×	use-when	precondition	only_use_if	filter
Query	×	×	×	only_use_for_query	×
Compute	×	×	×	compute	×
Achieve	×	×	×	achieve at N	×
	goal	goal	goal	achieve at N after < time point >	goal task

is accomplished, we try to make high-level statements on the main idea behind the task decomposition at each planner. For example, the statement "decomposition: code" implies that the decomposition of a 'compound' task in NOAH is accomplished by an evaluation of the respective node's code, but also the application of a 'primitive' task is done by the evaluation of that code ("application: code").

The constraint-related concepts, namely the commitment strategy and constraint management in the case of task interactions are shown in the upper part of Table 2. Plan-based HTN planners take advantage of the least-commitment strategy, however, we note two deficiencies. First, except UMCP, which supports additional commitment strategies [52], the rest of the planners take a rigid approach of incorporating the commitment strategy into the search mechanism, resulting in tightly coupled planning techniques. Second, only few planners backtrack on poor decisions, thus not implementing the concept of the least-commitment strategy completely. The lower part of Table 2 summarises and classifies resolution methods with respect to the task interaction they solve. Except for UMCP, which clearly defines its resolution method, the other plan-based HTN planners describe intuitively and, in some situations, inadequately their resolution methods. NOAH and SIPE-2 need to handle the largest set of interactions, while Nonlin and UMCP handle only one harmful and one helpful interaction.

Table 3 summarises and classifies explicit conditions that plan-based HTN planners employ. We observe that Nonlin initiated the idea of explicit conditions, supporting four types of conditions. O-Plan2 supports the largest set of conditions, while they play some "special role" in planner's planning process (since it does not consider any notion of a goal) [11]. In UMCP, conditions are represented as state constraints. In addition to explicitly typing them into the domain knowledge, the planner is extended to reason about implicit external conditions by examining the domain knowledge [53].

2.4. State-based HTN planning

We complement Definition 2 and Definition 3 of the theoretical framework as follows.

Definition 10 *(Task network).* A *task network* tn is a pair $(T, \prec)$, where T is a finite set of tasks, and $\prec$ is a strict partial order on T (irreflexive, transitive, asymmetric).

A task network tn in state-based HTN planning is less expressive than the one in plan-based HTN planning. That is, tn does not allow multiple occurrences of a same task in the partial ordering of tasks.

Definition 11 *(Method).* A *method* m is a triple $(c(m), pre(m), tn(m))$, where $c(m)$ is a compound task, $pre(m) \in 2^{\mathcal{Q}}$ is a precondition, and $tn(m)$ is a task network. The subsets $pre^{+}(m)$ and $pre^{-}(m)$ denote positive and negative precondition of m, respectively.

A method m is applicable in state s, if and only if $pre^{+}(m) \subseteq s$ and $pre^{-}(m) \cap s = \emptyset$. Applying m to s results in a new task network.

Definition 12 *(Decomposition).* Let m be an applicable method in s and $tn_c = (T_c, \prec_c)$ be a task network. Method m decomposes tn_c into a new task network tn_n by replacing task t, written $tn_c \xrightarrow[s,t,m]{}_D tn_n$, if and only if $t \in T_c$, $t = c(m)$ and

$$tn_n := ((T_c \setminus \{t\}) \cup T_m, \prec_c \cup \prec_m \cup \prec_D) \text{ where}$$
$$\prec_D := \{(t_1, t_2) \in T_c \times T_m \mid (t_1, t) \in \prec_c\} \cup \{(t_1, t_2) \in T_m \times T_c \mid (t, t_2) \in \prec_c\}$$

Definition 13 *(Solution).* Let $\mathcal{P}$ be an HTN planning problem. The sequence $o_1, \ldots, o_n$ is a *solution* to $\mathcal{P}$, if and only if there exists a task $t \in T_0$, where $tn_0 = \langle T_o, \prec_0 \rangle$, such that $(t, t') \in \prec_0$ for all $t' \in T_0$ and

- t is primitive and applicable in s_0 and the sequence $o_2, \ldots, o_n$ is a solution to $\mathcal{P}$ in which the task network is $tn_0 \setminus \{o_1\}$ and the state is $s_0[o_1]$; or
- t is compound and there is a task decomposition in s_0 such that the sequence $o_1, \ldots, o_n$ is a solution to $\mathcal{P}$ in which $tn_0 \rightarrow_D tn'$.

In [50], the authors define a progression problem space for this model of HTN planning. The space is a directed graph in which pairs of state and task network are vertices, and a progression from one pair to another is an outgoing edge. We take a slightly different approach in which a state is a vertex, and a task decomposition maps to the same state where the corresponding method is applicable, and an operator application leads to a successor state.

Definition 14 *(State space).* Given a (state-based) HTN planning problem $\mathcal{P}$, a *state space* SG is a directed graph $(\mathcal{V}, \mathcal{E})$ such that $s_0 \in \mathcal{V}$, and there is a state s_i and $t_k \in tn$ such that

- if t_k is primitive, then $s_i[t_k] = s_{i+1}$ such that $k = i + 1$, $s_i, s_{i+1} \in \mathcal{V}$ and $(s_i, s_{i+1}) \in \mathcal{E}$; or
- if t_k is compound, then $tn \rightarrow_D tn'$ is a self-transition such that $s_i \in \mathcal{V}$ and $(s_i, s_i) \in \mathcal{E}$.

2.4.1. Review of state-based HTN planners

Similarly to the plan-based HTN planners, we summarise topics on task decomposition, constraints and constraint-based techniques, and explicit conditions. The summaries are organised in a tabular form for which we use the same notation as in Section 2.3.1.

State-based HTN planners follow the task decomposition as described in Section 2.2.1, and indeed distinguish between primitive and compound tasks (Table 4). In both planners, the set of methods can be seen as an if-then-else representation, that is, the planners select the first method whose if-statement (preconditions) holds in the current state. Given a compound task, a task decomposition evaluates the preconditions of task's associated methods, and chooses the first method applicable in the current state to expand the existing task network. Two observations are in order. First, recall that in Section 2.2.1 we stated that the task decomposition makes a non-deterministic choice of which method to use for the decomposition. However, in the case of both planners, the choice is controlled, that is, the first method from the if-then-else representation that is applicable in the state is chosen. Second, SHOP2 uses the unordered task decomposition (its predecessor SHOP employs the totally ordered task decomposition), while SIADEX follows the partially ordered task decomposition. Consequently, SHOP2 does not need to check the task network for corrections during task decomposition, but SIADEX must verify that no (ordering) constraints are violated in the newly created task network.

Table 5 demonstrates the summary on the commitment strategy and constraint management of state-based HTN planners. As the upper part of the table depicts, state-based HTN planners employ the early-commitment strategy. If some

Table 4
Task decomposition in state-based HTN planners.

	Hierarchical representation		Task decomposition		
	Mechanism for primitive and compound tasks	Task network	Process	Type	Constraint check
SHOP2	separate for each	network of primitive and compound tasks	decomposition: compound task application: primitive task	UTD	–
SIADEX	separate for each	network of primitive and compound tasks	decomposition: compound task application: primitive task	POTD	✓

Table 5
Constraint-related concepts in state-based HTN planners.

Commitment	SHOP2	SIADEX
Strategy	early	early
Backtracking	yes	yes
Interaction		
Deleted-condition	Protection condition, domain axiom	Constraint propagation
Double-cross	×	×
Resource	×	×
Placeholder replacement	×	×
Phantomisation	Domain method	Domain method
Disjunct optimisation	×	×

Table 6
Condition types in state-based HTN planners.

Condition	SHOP2	SIADEX
Supervised	×	×
External	call	arbitrary code
Filter	precondition	precondition
Query	×	×
Compute	call	function
Achieve	×	×

task fails, both planners backtrack on other alternatives according to a list of variable bindings for the task precondition, or maybe to some criterion specified in the definition of the task. In addition, SIADEX supports cutting of backtracking points (as performed in Prolog [55]). By taking the early-commitment strategy into consideration, we could conclude that state-based HTN planners avoid task interactions altogether. However, this statement is not entirely correct. The lower part of Table 5 shows that in SHOP2, for example, a deleted-condition interaction may arise due to the process of interleaving tasks. The planner is able to solve this situation under a rather restricting assumption, that is, it requires a specification of 'protection' conditions in the effects of operators. A protection request enforces the planner from deleting conditions, and a protection cancellation allows the planner to delete these conditions. In some cases, the planner can deal with deleted-condition interaction using domain axioms. SIADEX needs a more powerful mechanism to accomplish planning and handle interactions that may arise in partially ordered task networks. The planner uses a causal structure of tasks and task networks. Constraint satisfaction checks the consistency of task networks (and the solution) based on that causal structure, and constraint propagation is used to post constraints, if necessary.

State-based HTN planners do not share the strong need for explicit conditions with plan-based HTN planners, as shown in Table 6. The whole reasoning power of SHOP2 and SIADEX is encapsulated in the preconditions of both primitive and compound tasks, thus they do not require other explicit domain knowledge. In the scope of preconditions, however, SHOP2 enables various types of computations, such as invocations of external knowledge resources by using the 'Call' condition. SIADEX also supports complex computations by incorporating complete (Python-based [56]) procedures in the domain. External conditions are modelled in a similar fashion.

3. Analysis of HTN planning and HTN planners

So far we categorised HTN planners based on the space they search in. This categorisation highlights the common search-related features among planners, though we have two more reasons to analyse the planners and HTN planning. First, we want to cover the capabilities of HTN planners with respect to their support for expressiveness, their domain dependence, soundness, completeness, fault tolerance, and efficiency. The second reason lies in some of the implicit assumptions made about HTN planners, that is, claims and beliefs accepted for granted and without evidence. These include the "sophistica-

tion" of domain knowledge provided to HTN planners, the expressive power of HTN planning between theory and practice, HTN planners being fast and scalable, and HTN planning being very suitable for and most applied to real-world problems.

To this end, we provide the third piece of our pie of frameworks, an analytical framework to collect and organise studies on HTN planning and planners. We then apply exploratory research to examine diversity and similarity of HTN planners within their category and between categories, and comparative research to make sense of a range of cases. We believe that, in this way, statements about domain knowledge, expressiveness, performance, and applicability can be made in a neutral and evidence-oriented way.

3.1. Analytical framework

The analytical framework directs us on where to look and what kind of properties to look for but without making specific hypotheses about relationships among properties. The framework consists of five main elements, namely, domain authoring, expressiveness, competence, computation, and applicability. Each element and the motivation for its inclusion in the framework are described in the following sections. Whenever possible, we provide formal definitions that may be related to the theoretical framework of Section 2.1 and the two models presented in Sections 2.3 and 2.4.

3.1.1. Domain authoring

An interesting perspective on HTN planners says:

> "[Compared with classical planners,] the primary advantage of HTN planners is their sophisticated knowledge representation [and reasoning capabilities] [1]."

Two remarks are in order. First, there is uncertainty in the meaning of "sophisticated". Does it refer to the complexity, richness or some other attribute of the representation? Let us assume that it refers to the so-called "knowledge-rich" representation [57]. The second remark is on HTN planners taking advantage of the use of knowledge-rich encodings. On the one hand, this could be correct, if we consider that these planners improve their performance (over classical planners) thanks to their domain knowledge [58]. On the other hand, why are HTN planners in advantage if we do not know at what expense, in terms of encoding effort, we obtain that improvement?

We define *domain authoring* the formulation of domain knowledge as performed by a domain author. What we are really interested in, in this process, is the relative effort needed to formulate such domain knowledge for an HTN planner. The community, however, has not yet found a way or measures to provide an objective answer to this type of questions. The ambiguity and difficulty to define an answer come directly from the capabilities and experience of the domain author with respect to the understanding of the underlying planner and the expertise for the respective domain [58].

Since the knowledge-rich representation is a strong requirement for HTN planning, we still want to give a flavour of the effort needed to provide domain knowledge for each planner. We take a model of the well-known and overused domain of block world [59] as described for each planner, and inspect each model from two viewpoints. We first take a single and the same task of each domain model and analyse closely what needs to be encoded. Second, we give a broader view of each domain model by quantifying its content with respect to knowledge symbols, keyword symbols, and domain elements.

3.1.2. Expressiveness

We tackle expressiveness from two perspectives. The first one refers to the formal properties of expressiveness of the HTN planning language. In order to completely determine what the language can express, we need formal semantics for the language. Fortunately, this issue has been a subject of interest for some time, resulting in a number of studies on expressiveness of HTN planning [60–62,57,63,12]. We analyse the expressiveness of HTN planning language from a model-theoretic, operational and computational aspect based on the results provided in [64]. In each aspect, the expressiveness of HTN planning is compared to the one of STRIPS-like planning.

The second perspective refers to the practical expressive power of HTN planners. We could determine the practical expressiveness of the planners' language by the assessment of the breadth of what the language can represent and communicate. The breadth may include the language's formal system, the support for preferences, time, *etc*. Unfortunately, there is no common planning language for HTN planners. The idea of standardising a planning language is introduced with the Planning Domain Definition Language (PDDL) [65] in 1998 for the purpose of the International Planning Competition, rather late with respect to the history of, above all, plan-based HTN planners. Although in the first version of PDDL there was an attempt to formalise a common syntax compatible to HTN planners, the idea was discarded with version 2.1 of PDDL due to the immense differences between planners [66].

We can still provide some insights about what HTN planners can express in practice by exploring the expressiveness of each planner's language separately. For this purpose, we use three categories of properties. The first category encompasses the system of first-order logic, particularly the support for a set of logical connectives: conjunction ($\wedge$), disjunction ($\vee$), implication ($\Longrightarrow$), and negation ($\neg$), and the support for universal ($\forall$) and existential ($\exists$) quantifiers. A logical connector or quantifier can be applied on $pre(o)$ and $eff(o)$, where o is an operator, on the formula ψ of a task network tn, and on $pre(m)$, where m is a method.

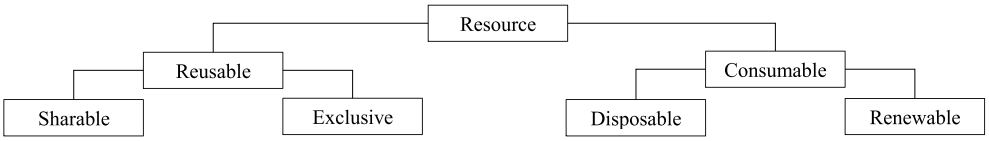


Fig. 2. Resource hierarchy.

The second category encompasses the quality constraints that can be expressed in some languages, particularly the support for typing, extended goals, and preferences. Given an HTN planning problem $\mathcal{P}$, we define each as follows.

- *Typing* enables expressing types of objects in a type hierarchy (similar to typing in PDDL). Each $v \in V$ may have a type $\mathbf{t} \in \mathbf{T}$, where $\mathbf{T}$ is a set of types. The type hierarchy is built by a sub-typing relation $st : \mathbf{T} \times \mathbf{T}$, which is reflexive and transitive.
- HTN planning assumes an initial task network tn_0 to be accomplished as an objective for $\mathcal{P}$. In its simplest form, tn_0 does not allow to specify conditions to be satisfied in some intermediate state during or in the final state of the execution of the solution to $\mathcal{P}$. *Extended goals* enable to express a planning objective in a way that its satisfaction could be on a part, on the whole trajectory of the solution, or in the final state. In classical planning, this is usually achieved through the use of temporal modal operators.
- A *preference* is a condition on the solution trajectory that some user would prefer satisfied rather than not satisfied, but would accept if the condition might not be satisfied [67].

The third category encompasses resource and time constraints. Given an HTN planning problem $\mathcal{P}$, a *resource* r is an object of limited capacity $\bar{r}$ for use by a task t within $\mathcal{P}$. The capacity $\bar{r}$ can be a categorical value, such as free (for use) and used, or a numerical value. If $c_i(r)$ denotes the current capacity of a resource r, then $c_0(r) = \bar{r}$ is the initial capacity. We use $c_t(r)$ to denote the consumption of r by a task t. We use $t_1 \parallel t_2$ to denote that tasks t_1 and t_2 are in parallel, and $c_{t_1}(r) \parallel c_{t_2}(r)$ for the consumption of resource r by tasks t_1 and t_2 at the same time. Thus, $c_{t_1}(r) \parallel c_{t_2}(r)$ is possible iff $t_1 \parallel t_2$.

- A resource is reusable if it can be used more than once. Let t be a task and r be a resource whose $\bar{r}$ has a categorical value. The resource r is reusable iff $c_i(r) = c_0(r)$ immediately after $c_t(r)$.
- A shared reusable resource can be shared among several tasks at the same time. Let t_1 and t_2 be tasks and r be a resource. The resource r is shared reusable iff r is reusable and $c_{t_1}(r) \parallel c_{t_2}(r)$.
- An exclusively reusable resource cannot be used by two tasks in parallel. Let t_1 be a task and r a resource. The resource r is exclusively reusable iff r is reusable and $c_{t_1}(r)$ such that $c_{t_1}(r) \nparallel c_{t_i}(r)$, where $i > 1$.
- A resource is consumable if it is usable only a limited number of times. Let t be a task and r be a resource whose $\bar{r}$ has a numerical value. The resource r is consumable iff $c_i(r) = c_{i-1}(r) - c_t(r)$ immediately after $c_t(r)$. A consumable resource can be replenished or not.
- If the resource cannot be restored after the use of the set amount, it is called disposable consumable resource. Let t be a task and r a resource. The resource r is disposable consumable iff r is consumable, and there exists i such that $c_i(r) = 0$ and there is no $i + k$, $k \in \mathbb{N}$ such that $c_{i+k}(r) > 0$.
- If the resource amount can be topped up, it is called renewable consumable resource. Let t be a task and r a resource. The resource r is renewable consumable iff r is consumable, $c_i(r) = c_{i-1}(r) - c_t(r)$ and there exists $o \in O$ and $k \in \mathbb{N}$ such that $c_{i+k}(r) = c_i(r) + rep$, where $rep \in eff(o)$.

Fig. 2 shows the relations between the aforementioned types of resources. Finally, we define time as usually considered, that is, a consumable resource that cannot be reproduced. We are interested in how HTN planners represent and handle temporal information.

3.1.3. Competence

We use the term *competence* to encompass a category of functional and formal properties that relate to specific abilities of HTN planners. We begin with properties of the functional design of HTN planning.

Domain dependence defines the ability of a planning technique to solve planning problems in different domains [4]. This is the issue of *domain-specific planning*, which is designed to solve problems only in a particular domain, versus *domain-configurable planning*, which solves planning problems in any domain given specific knowledge for every domain, versus *domain-independent planning*, which solves planning problems in any domain without specific demands. Given that HTN planning can solve problems in various domains, and it requires specific-domain knowledge provided in the set of methods M, HTN planning is a domain-configurable planning technique. This implies a design and implementation of HTN planners that include general problem-solving mechanisms. A problem-solving mechanism takes in a given $\mathcal{P}$ and computes a solution. It makes use of various algorithms and backtracking mechanisms, heuristics, specific control knowledge, and constraint management. Thus, several options arise for the design of a planner's mechanism:

- *Algorithm* represents the search procedure incorporated in the problem-solving mechanism. The mechanism can employ one or more kinds of search strategies, such as depth-first search (DFS), breadth-first search (BrFS), iterative deepening search (IDS), best-first search (BFS), or other heuristic search (HS) strategies. The algorithm incorporates the process of task decomposition (see Section 2.2.1), and may traverse the data structure of $\mathcal{P}$ with or without *backtracking* points. Recall from Section 2.2.2 that in HTN planning there are three types of backtracking or decision points: which task to deal with next, which method to use for some task, and which value to bind to a variable.
- *Heuristics* are functions that help the problem-solving mechanism to guide and speed up the search for a solution. In some HTN planners, heuristics may trade completeness for speed.
- *Domain-specific control* is represented by the set of methods M. The problem-solving mechanism evaluates the preconditions of methods in M to guide the planning process.
- *Interactive control* involves user's decisions during planning. A user may guide the problem-solving mechanism by choosing values to bind to variables in V, imposing ordering constraints on a set of tasks T, and decomposing the current task network tn.
- *Constraint management* (CM) deals with constraints that are part of task networks of methods in M, and those that can be added during planning. The problem-solving mechanism makes use of the constraint-related operations discussed in Section 2.2.2.

The following formal properties show when an HTN problem-solving mechanism (or, equally, an HTN planner) is sound and complete, and when the solution that the mechanism generates is flexible.

- Solution flexibility defines the ordering of operators in the solution to a planning problem. Let $\mathcal{P}$ be an HTN planning problem. The solution to $\mathcal{P}$ is *flexible* if it is partially ordered.
- Let $\mathcal{P}$ be an HTN planning problem. An HTN planner is *sound* if every plan it gives is a correct solution to $\mathcal{P}$.
- Let $\mathcal{P}$ be an HTN planning problem. An HTN planner is *complete* if it always finds a solution to $\mathcal{P}$ when such a solution exists.

Problem-solving mechanisms perform off-line planning with the closed-world assumption: the environment can only be changed by the execution of operators selected by the planner. However, this is not the case in real-world environments, which are of complex and dynamic nature and include other agents executing their own independent actions. During the execution of a plan, some unexpected event may occur that invalidates the solution being executed. If an event represents a state transition, then, from a planner's perspective, the environment changes its state as a result of two event classes: plan operators and fault occurrences. The planner itself is responsible for the selection of plan operators. Otherwise, the planner sees an unexpected or unwanted state transition in the environment as a *fault*. The augmentation of an HTN planner with the ability to handle faults in a well-defined way at execution time makes the planner *fault tolerant*. In order to do so, it is a prerequisite to specify the set of faults that an HTN planner can handle. In the event of a fault at execution time, the planner must monitor and recognise the fault, deduce the parts of the solution that are affected by the fault, and repair the existing and affected part of the plan, or re-plan for a new solution. Thus, the planner must ensure that there is a valid plan that accomplishes the initial task network. In the following, we define the notion of fault and fault tolerance formally.

- Let $\mathcal{P}$ be an HTN planning problem, $\pi = o_1, \ldots, o_n$ be the solution to $\mathcal{P}$, where $s_0[\pi] = s_n$. Let π_e be the partially executed part of π, and π_r the remaining part still to be executed. The execution of π is in a correct state s iff $s[\pi_r] = s_n$.
- A fault f is a state transition $s[f] = s'$ such that $f \notin O$ and $s'[\pi_r] \neq s_n$. The fault set is denoted as F.
- Let $\mathcal{P}$ be an HTN planning problem and π be a solution to $\mathcal{P}$. An HTN planner is said to *tolerate faults from a fault set* F during the execution of π iff for each $f \in F$, there exists a sequence of operators π_f such that π_f is a solution to $\mathcal{P}$.

3.1.4. Computation

Similarly to expressiveness, we are interested in computation from two perspectives. The first one refers to the theoretical computational boundaries of HTN planning. Under this perspective, we deal with the following decision problem PLAN-EX: Given an HTN planning problem $\mathcal{P}$, does $\mathcal{P}$ have a solution? We answer this question by using the results presented in [64], where time and space complexity of HTN planning is analysed.

The second perspective refers to the computational performance of HTN planners. We are interested in the runtime and scalability results of each HTN planner. We say that a planner is *scalable* if it is capable to cope and acceptably perform under a varying size of planning problems. Anything but easy is to define dimensions that could measure the size of a problem, nevertheless, scalability is highly desirable in practical settings with an increasing and large number of facts about the state, a large number of users, or a large number of tasks. We are also interested in how well planners scale relative to one another assuming increasingly difficult problems. As for *runtime*, we are interested in pairwise comparisons between HTN planners with respect to the amount of time they spend on the same sets of problems.

3.1.5. Applicability

Applicability is the last element of the framework and concerns the use of planners in actual applications. It appears to be orthogonal to previous categories, but we have two reasons for its inclusion. First, we strongly believe that the ultimate objective of research on automated planning must be exploitation of planners in a variety of real applications. Oil spills [68], spacecraft assembly [69], microwave manufacturing [70], smart spaces [71], and Web service composition [32] are a few prominent examples. Second, HTN planning is promoted as the most applied automated planning technique for real-world problems [72], mostly referring to the applications of SHOP2. Thus, we want to see whether and how HTN planning contributes towards the aforementioned objective, and what is the applicability of HTN planners.

3.2. Outcome

Next, we check selected literature for the properties of each element of the analytical framework. In two cases, we provide theoretical and practical interpretations of a framework element. Where possible, we also show comparison of HTN planners. In some cases, we aggregate the data on planners in tabular form. We use the following common notation for all tables. The '✓' denotes that a planner supports the respective property, the '✕' indicates that a planner does not support a particular property, and an empty cell denotes that it is unknown from the literature whether the planner is able to deal with a given element. There are rated properties, where the rate ranges from '★', denoting limited support for the given property, to '★★★', indicating extended support.

Domain authoring The first part of domain authoring deals with the encoding of the same task in the domain model of each planner. Fig. 3 shows the description of the 'put-on' task provided to each HTN planner. We start by describing NOAH's task which specifies that three statements should be evaluated in sequence (lines 4, 10 and 12). The first statement is handled by evaluating lines 5 and 7, which cause new tasks to be added. If a predicate (*e.g.*, '(cleartop $x)') is not true, we should expect that a task would be added to achieve the predicate, otherwise some form of a 'do-nothing' is assumed. The statement in line 10 is evaluated analogously. For the last statement in line 12, we need to be aware that it deletes a predicate from the (global) state, but also inserts that predicate in the add list of some dummy task at the current level (of planning). In contrast to NOAH, the encoding of Nonlin is more clear. In Nonlin, the task contains three filter conditions (lines 3 to 6). The first two filter conditions state that before applying the task two blocks must be clear. The third filter condition is used to bind some variable with what is on top of the variable from line 3. If all of these conditions hold, the effects of the task are applied (lines 7 to 11). With SIPE-2 things are getting complicated again. A simplified analysis says that line 3 specifies the goal that this task can achieve. In the element of line 5, a task network of two parallel tasks is contained with the purpose of achieving some predicates. The element in line 9 specifies the action that should be used in order for the task to be accomplished. The task contains no information about when a block is clear or not, or when a block is not on another block. This information would be inferred by planner's deductive theory. O-Plan2's task is very similar to the one of Nonlin. Line 3 specifies the task network used to accomplish the task, that is, a single action. The expressions in lines 4 to 9 are the actual effects of the action. On the other hand, three conditions must be satisfied in order for these effects to be applied. The conditions in lines 9 and 11 might be achieved by other tasks during the planning process. The condition in line 12 is used to bind a variable for the purpose of specifying the effects.

The task descriptions for UMCP, SHOP2 and SIADEX differ only in the notation, but specify almost the same meaning. All tasks (that is, operators) contain simple applicability preconditions (line 2 in SHOP2), and effects (line 4) as a postcondition in UMCP, and as a delete and an add list in SHOP2 (line 3 and line 4, respectively). Beside the representational simplicity, the power of these operators is, however, much weaker than the tasks of the previously discussed planners. The operators cannot handle situations where some block is above another one or when a block is not clear. The approach to achieve fairly equal functionality would be to include methods that describe all possible situations.

The second part of domain authoring gives us insights into encoding domain models by measuring and comparing the sizes of tasks and domain models for each planner, an approach inspired by the one used in [73]. Although we use domain models for the same domain, we do not assume that the models have the same level of expressiveness. The idea is to establish a relation between the size of a domain and the effort needed to encode that domain. When domain models would have the same level of expressiveness, a smaller domain size would mean that the domain model is easier to encode as compared to the one with a larger size. From Fig. 3, we can observe that UMCP and SHOP2 have a smaller size of the 'put-on' task as compared to the ones of other planners, but looking at the number of symbols at the domain level, as shown in Fig. 4a, SIPE-2 has the largest domain model, however, almost half of it belongs to keyword symbols. On the other hand, SHOP2 has a slightly smaller domain model than SIPE-2, but the number of keyword symbols is negligible, which means that the rest of the symbols represent the actual domain knowledge. In addition, in UMCP and SHOP2, the knowledge is partitioned in a larger number of tasks as compared to the rest of HTN planners. SHOP2 uses knowledge structured in 13 primitive and compound tasks in total, and 6 axioms, while O-Plan2, for example, uses three tasks in total. There are four main reasons for these observations:

- In SHOP2, a predicate q can only be satisfied by specifying a separate task with one or more methods that should make q true. In the block-world domain, the top-level task `achieve-goals` is responsible for this. In UMCP, q can be

NOAH

```
(puton
  (qlambda
    (on <-X <-Y)
     (pand
          (pgoal (clear $x) (cleartop $x)
               apply (clear))
          (pgoal (clear $y) (cleartop $y)
               apply (clear))
      )
     (pgoal (put $x on top of $y)
              (on $X $y) apply nil)
     (pdeny (cleartop $y))))
```

Nonlin

```
actschema puton
  pattern <<put $*x on top of $*y>>
  conditions
    holds <<cleartop $*x>> at self
    holds <<cleartop $*y>> at self
    holds <<on $*x $*z>> at self
  effects
    + <<on $*x $*y>>
    - <<cleartop $*y>>
    - <<on $*x $*z>>
    + <<cleartop $*z>>
  vars x undef y undef z undef;
end;
```

SIPE-2

```
operator: puton
arguments: block1, object1 is not block1;
purpose: (on block1 object1);
plot:
parallel
  branch 1: goals: (clear object1);
  branch 2: goals: (clear block1);
end parallel
process
action: puton.primitive;
arguments: block1, object1;
resources: block1;
effects: (on block1 object1);
end plot end operator
```

O-Plan2

```
schema puton;
 vars ?x=undef, ?y=undef, ?z=undef;
 expands {put ?x on top of ?y};
 only_use_for_effects
  {on ?x ?y}=true,
  {cleartop ?y}=false,
  {on ?x ?z}=false,
  {cleartop ?z}=true;
 conditions
  only_use_for_query {on ?x ?y}=true,
  achievable {cleartop ?y}=true,
  achievable {cleartop ?x}=true;
endschema;
```

UMCP

```
(:operator puton (x y)
  :pre ((clear x)(on x table)(clear y))
  :post ((~on x table)(on x y)(~clear y)))
```

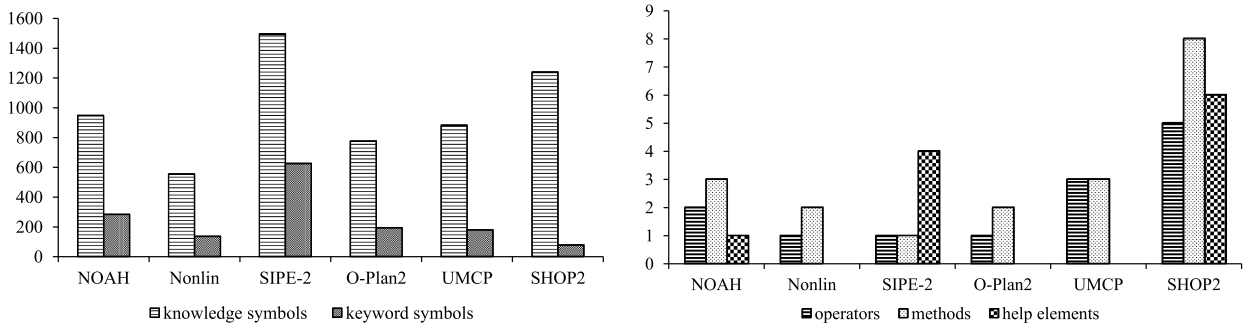
SHOP2

```
(:operator (!puton ?x ?y)
  ((clear ?x)(on-table ?x)(clear ?y))
  ((clear ?y)(on-table ?x))
  ((on ?x ?y)))
```

SIADEX

```
(:action puton
 :parameters (?x ?y - block)
 :precondition (and (grasping ?x)(clear ?y))
 :effect (and (not(grasping ?x))(not(clear ?y))
         (clear ?x)(on ?x ?y)(handempty)))
```

Fig. 3. The 'put-on' task from the block-world domain in HTN planners.



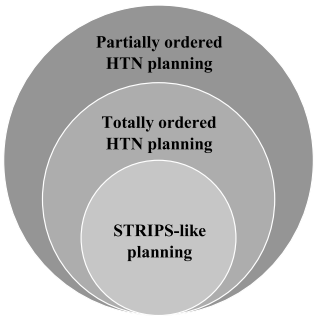
(a) Number of knowledge and keyword symbols

(b) Number of domain elements

Fig. 4. Quantitative perception of the block-world domain in HTN planners.

achieved through the use of the syntactic form *achieve*(q). In the rest of plan-based HTN planners, there is no need for a separate task. The predicate can be part of the initial task network.

- In SHOP2, the recursive tasks (*e.g.*, `find-movable`) need an additional method which encodes the base case whose decomposition does nothing. In plan-based HTN planners, such method is not necessary, because the phantomisation takes care when a certain predicate is already achieved and nothing needs to be done.
- In SHOP2 and UMCP, there is a need for an explicit check of deleted-condition interaction. In the block-world domain of SHOP2, deleted-condition interaction is handled by using Horn clauses to reason about stacks of blocks. In UMCP, this can be accomplished generally by using *achieve*(q), which constraints q to be true right after accomplishing the corresponding task. The rest of plan-based HTN planners do not use the domain knowledge to handle the deleted-condition interaction, but instead use their problem-solving mechanisms to solve it (see Section 2.2.2).
- SHOP2 uses special so-called book-keeping operators to keep track of what needs to be done during planning. For the block-world domain, SHOP2 uses two book-keeping operators (`assert` and `remove`) that are not part of the actual block-world domain. On the other hand, other HTN planners do not use any special operators.



(a) Hierarchy of model-theoretical and operational expressiveness

(b) Hierarchy of computational-based expressiveness

Fig. 5. Hierarchies of expressiveness.

The analysis of tasks shows, on the one hand, that a prerequisite to author domain knowledge for most plan-based HTN planners is the comprehension of their underlying systems, such as expectations of what the system would do in a particular situation. On the other hand, a domain author does not need to have a priori familiarity with a state-based HTN planner, but the author must provide an elaborate domain model. This is supported by the analysis of results shown in Fig. 4. Additional evidence to the latter observation is the criticism of SHOP2 planner that it is a problem-solving programming language rather than a planner [74]. Since we cannot assume that the domain models of HTN planners have the same level of expressiveness, we are prevented from making statements about the relative effort needed for domain authoring. Finally, Fig. 4a indicates the richness of domain models with knowledge.

Expressiveness We gain a perspective in theoretical expressiveness by summarising the findings in [64].[3] Fig. 5a depicts the model-theoretic expressiveness. From this aspect, the HTN language is strictly more expressive than the STRIPS language, but totally ordered HTN planning is less expressive than partially ordered HTN planning and strictly more expressive than STRIPS-like planning [62]. That is, an HTN planning problem (with totally or partially ordered task networks) can be transformed into a STRIPS-like planning problem, but the converse is not true – STRIPS language cannot express the hierarchical constructs. On the other hand, a totally ordered HTN planning problem can be transformed into a partially ordered HTN planning problem (that is, the latter one supports totally ordered task networks), but the converse is not true – totally ordered HTN planning does not support partially ordered task networks.

Fig. 5a shows that the operational aspect has the same expressiveness hierarchy as the model-theoretic aspect. That is, HTN planning is strictly more expressive than STRIPS-style planning, and totally ordered HTN planning is strictly between STRIPS-like planning and partially ordered HTN planning. Fig. 5b depicts the computational-based hierarchy. Once more, HTN planning is strictly more expressive than STRIPS-like planning. In particular, there is a (polynomial) transformation from STRIPS-like planning to HTN planning, but there is no computable transformation from HTN planning to STRIPS-like planning. Intuitively, HTN elements can represent computationally more complex problems than STRIPS-like operators. However, these results are true when partially ordered task networks are allowed. In fact, totally ordered HTN planning is at the same level of expressiveness as STRIPS-like planning, but significantly less expressive than partially ordered HTN planning. This is because totally ordered HTN planning avoids interleaving of tasks from different compound tasks.

We can conclude that HTN planning is able to express a broader and more complex set of planning problems than STRIPS-like planning. This statement is however controversial since the assumption is that the theoretical model of HTN planning uses an infinite set of symbols to represent tasks. But in reality, this model cannot be supported by any planner unless restrictions are imposed [63].

Table 7 illustrates the practical expressiveness of the planning languages of HTN planners. SIPE-2, UMCP, SHOP2 and SIADEX employ the set of logical connectives ($\wedge$, $\vee$, $\Longrightarrow$, $\neg$) in task preconditions with some restrictions, while $\vee$ is not allowed in the effects of tasks. Thus, task preconditions are more expressive than the task effects. Given a predicate q, except Nonlin and SHOP2, which use deletion of q, other HTN planners use $\neg q$ in the effects. SHOP2's and SIADEX's languages support about the level 2 of the PDDL version 2.1 (*i.e.*, numeric extensions), and allow $\Longrightarrow$ in task preconditions and $\forall$ in task preconditions and effects (with different semantics, however). Furthermore, the most extensive support for typing has SIPE-2 that goes even beyond what we defined. For example, we can state that a variable must not be of a certain type. The rest of HTN planners have either very limited or no support at all for typing. With respect to extended goals, SIPE-2, O-Plan2 and SIADEX support temporally extended and deadline goals. Temporally extended goals are expressed through the use of temporal modal operators, while deadline goals expresses conditions that must be achieved at a specific point in time in the trajectory. Default mechanisms of HTN planners do not support preferences, but SIPE-2 and SHOP2 have been extended to handle some forms of preferences. SIPE-2 is extended to two forms of preferences [75,76]. The first form

[3] We assume that the reader is familiar with model-theoretic, operational and computational-based expressiveness. Otherwise, we refer to [64] for details.

Table 7
Practical expressiveness of HTN planners.

Property	NOAH	Nonlin	SIPE-2	O-Plan2	UMCP	SHOP2	SIADEX
Conjunction	✓	✓	✓	✓	✓	✓	✓
Disjunction	preconditions	×	preconditions	constraints	preconditions	preconditions	preconditions
Negation	effects		preconditions effects	effects	preconditions constraints effects	preconditions	preconditions
Implication	×	×	×	×	×	preconditions	preconditions
Existential q.			preconditions conditions		constraints		
Universal q.	×	×	effects	×	×	preconditions effects	preconditions effects
Sort hierarchy	×	×	★★★	★	×	×	★
Extended goals	×	×	★	★	×	×	★
Preferences	×	★	★★	★	★	★★★	×
Resource	×	×	★★★	★★★	×	×	×
Time	×	×	★	★★	×	★	★★★

prescribes or prohibits the use of some variables within a task, while the second one prescribes or prohibits the use of a particular task when accomplishing some objective. SHOP2 is extended to support three types of preferences [77,78]. The first type are basic constructs of linear temporal logic. The second type are constrains, such as the precedence constraint `before(t,t')` and state constraints `holdBefore(t,q)`, where `t` and `t'` are tasks and `q` is a predicate. The third and most interesting type are the preferences over how tasks are decomposed into task networks (*e.g.*, prefer to apply a certain method over another), preferences over the parametrisations of decomposed tasks (*e.g.*, preferring one task grounding to other[4]), and a variety of temporal and non-temporal preferences over task networks.

Finally, SIPE-2 and O-Plan2 support our resource model completely. In addition to what we defined, O-Plan2 allows sharing reusable resources unary, where a sharable resource cannot be shared among many tasks at the same time, or simultaneously, where a resource can be shared among many tasks without any specific control. SIPE-2 offers a limited mechanism for temporal reasoning, but full support can be achieved by using an external temporal reasoning system, such as Tachyon [79]. O-Plan2 appears to have the most comprehensive support for temporal reasoning compared to its predecessors [11], however, this cannot be easily confirmed from the literature. SHOP2 does not explicitly reason about time. The planner has been temporally enhanced in several studies [13,80,81], but at the expense of degrading its performance and soundness. On the other hand, SIADEX provides clear explanations about its temporal reasoning mechanism and supports all relations defined independently by Allen [82] and van Benthem [83].

We may say that both categories of HTN planners are able to address similar level of expressiveness. It appears, however, that planners and their corresponding category still have some challenges to address. For example, first-order logic is not fully supported by most planners, and extended goals and quality constraints are not implemented or only partially implemented.

Competence Table 8 summarises the properties related to the competence of HTN planners. We begin with the property of solution flexibility. The result of the planning process in plan-based HTN planners is a partially ordered plan which is in compliance with the definition of flexibility. An exception to this is the UMCP planner which restricts the tasks of the solution to be totally ordered. With respect to state-based HTN planners, there are two cases as well. SHOP2 produces a totally ordered plan, while SIADEX is able to plan for a flexible solution.

Except for Nonlin and O-Plan2, all planners implement DFS as their main algorithm, however, not all of them backtrack to all alternative points. NOAH, Nonlin, SIPE-2 and O-Plan2 all make use of heuristics to guide their search for a solution. During the search, Nonlin employs dependency-directed backtracking – it backtracks on choices of variable bindings and choices of task orderings, while SIPE-2 backtracks chronologically, and uses the heuristics to limit backtracking points to alternative tasks and variable binding choices only in two places. SIPE-2 does not backtrack the addition of ordering constraints. O-Plan2 uses heuristics over its choices of tasks in the plan space for which an evaluation function based on the opportunistic merit of the state is used. In addition, user interaction addresses some decision-related issues that are beyond the capabilities of the algorithms. Among HTN planners, SIPE-2, O-Plan2, UMCP and SIADEX provide user interfaces for guiding purposes. Backtracking points indeed ensure completeness, however, only UMCP and SHOP2 are provably complete and sound planners [39]. Nonlin and O-Plan2 are designed but not proved to be complete with respect to the provided domain knowledge.[5]

[4] For the sake of completeness, the default mechanism of SHOP2 supports a similar feature called 'sort-by' that sorts variable bindings according to some criteria (*e.g.*, ascending order).

[5] Austin Tate, personal communication, November 23, 2012.

Table 8
Competence of HTN planners.

Property	NOAH	Nonlin	SIPE-2	O-Plan2	UMCP	SHOP2	SIADEX
Algorithm	DFS	BFS	DFS	HS	DFS	DFS	DFS
Backtracking	×	DD	Chronological				
Heuristics	✓	✓	✓	✓	×	×	×
Domain-specific control	×	conditions	preconditions	conditions	conditions	preconditions	preconditions
Interactive control	×	×	✓	✓	✓	×	✓
Constraint management	✓	✓	✓	✓	✓	×	✓
Alternative algorithms	×	×	×	×	BrFS, BFS	IDS	×
Solution flexibility	Flexible				Total		Flexible
Completeness	×	✓*	×	✓*	✓	✓	
Soundness	×	×	×	×	✓	✓	
Execution monitoring	★	×	★★★	★★★	×	×	★★★
Replanning	★	×	★★★	★★★	×	×	★★★

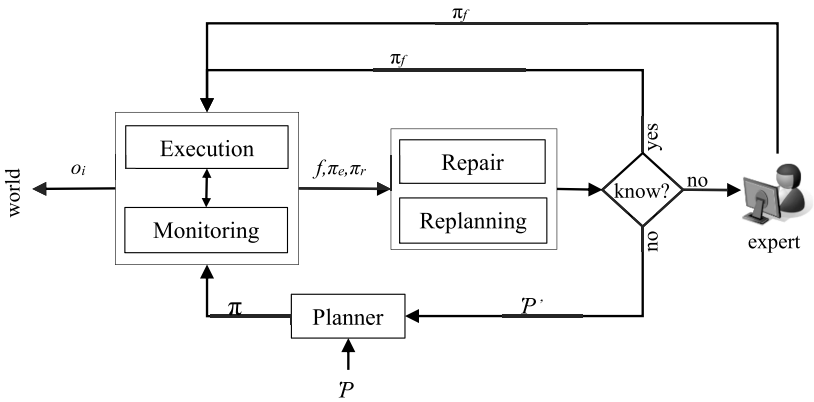


Fig. 6. General architecture for HTN planning systems augmented to monitor and execute plans, and tolerate faults.

From Table 8, we can observe that state-based HTN planners have much simpler problem-solving mechanisms than plan-based HTN planners. This might also confirm our statement that the underlying mechanisms of plan-based HTN are much more difficult to grasp.

With respect to the ability of HTN planners to monitor the execution of plans, recognise faults, and handle them accordingly, we abstract away the mechanisms of SIPE-2, O-Plan2 and SIADEX in a general architecture shown in Fig. 6. The architecture consists of the following components:

- **Planner** takes in the current HTN planning problem $\mathcal{P}$ and computes a solution plan π.
- **Execution and Monitoring** takes π and executes the actions one by one to the environment. It also processes the observations of the actual resulting state by comparing them with the expectations made during planning. If some discrepancy is recognised, a fault f is generated.
- **Repair and Replanning** takes f and tries to repair the current plan. If the component is able to do so, it sends the repaired plan for execution, otherwise it may ask a user for help, and if that does not work, it asks the planner to re-plan given a modified HTN planning problem $\mathcal{P}'$.

Computation In order to properly understand the theoretical point of view of computation, we need to take several possible settings into account [64]. To that end, let $\mathcal{P}$ be an HTN planning problem.

- The sets O and M can be provided in two ways. The sets can be a part of the *input*, or they can be *fixed* in advance, that is, the tasks are allowed to contain methods corresponding only to predicates in s_0.
- A compound task can be defined in several ways: (1) a compound task is without any restriction (*yes*); (2) a *regular* task in task networks – at most one compound task followed by all primitive tasks; (3) an *acyclic* task – a task can be decomposed to only a finite depth; and (4) compound tasks are not allowed at all (*no*).
- A task network containing primitive and compound tasks as defined in the previous point can be *totally ordered* or *partially ordered*, whenever there is a total or partial order among those tasks, respectively.
- Variables can be allowed or not in $\mathcal{P}$.

Table 9
Computational complexity of HTN planning (adapted from [64]).

Operators and methods	Compound task	Task network	Variables	Plan existence
fixed	yes	partially ordered	yes	undecidable
input	yes	partially ordered	yes	undecidable
input	yes	partially ordered	no	undecidable
input	yes	totally ordered	yes	2-EXPTIME EXPSPACE-hard
input	yes	totally ordered	no	EXPTIME PSPACE-hard
input	no	totally ordered	yes	NP-complete
input	no	totally ordered	no	P
input	no	partially ordered	yes	NP-complete
input	no	partially ordered	no	NP-complete
input	acyclic	partially ordered	yes	decidable
input	acyclic	partially ordered	no	decidable
input	regular	unimportant	yes	EXPSPACE-complete
input	regular	unimportant	no	PSPACE-complete
fixed	regular	unimportant	yes	PSPACE PSPACE-complete

Table 10
Performance of some of HTN planners.

(a) Runtime and scalability of UMCP and SHOP

Domain	Property	UMCP	SHOP
UM Translog	Runtime	10 problems > SHOP	100 problems > UMCP
	Scalability		> UMCP

(b) Runtime and scalability of SHOP2 and SIADEX

Domain	Property	SHOP2	SIADEX
Zeno travel	Runtime	2 problems = SIADEX	18 problems > SHOP2
	Scalability	= SIADEX	

The complexity results are summarised in Table 9. When no restrictions on compound tasks are imposed and task networks are partially ordered, then giving O and M in the input or fixing them in advance, or allowing variables or not, does not affect the outcome and PLAN-EX is undecidable. However, given O and M in the input, and being every task acyclic and every task network partially ordered, PLAN-EX becomes decidable. PLAN-EX is decidable when task networks are totally ordered. In particular, when unrestricted compound tasks and variables are allowed, PLAN-EX is EXPSPACE-hard in double exponential time (2-EXPTIME), or, if no variable is allowed, PLAN-EX is PSPACE-hard in exponential time. When only primitive tasks and variables are allowed, PLAN-EX is NP-complete, irrespective of the ordering of task networks. Furthermore, forbidding the use of variables makes PLAN-EX to be in P. However, disallowing variables when task networks are partially ordered tasks does not change the outcome and PLAN-EX remains NP-complete. Regardless of the ordering of task networks, when compound tasks are regular, there are two outcomes. When O and M are given in the input, and if variables are allowed, then PLAN-EX is EXPSPACE-complete, otherwise PLAN-EX is PSPACE-complete. When O and M are fixed in advance, and variables are allowed, PLAN-EX is PSPACE-complete in PSPACE.

With respect to practical results, unfortunately, for most of HTN planners the performance is unknown. To the best of our knowledge, three pieces of evidence report on performance and pairwise comparison results. The first evidence compares UMCP and SHOP under loads of different problems [84]. The experiments are based on the UM Translog [85], a domain similar, but larger than the standard logistics domain. For this domain, a set of problems with increasing number of boxes to be delivered is randomly created. The results show that the run time for UMCP is several orders of magnitude larger than the run time for SHOP. Only in first ten problems UMCP appears to perform better than SHOP, as depicted in Table 10a. Additionally, UMCP faced some difficulties when trying to find solutions to the problems. The planner tries to solve only 37% of total number of problems, and failed 45% of those 37%. The reasons for such behaviour are due to running out of memory, inability to find an answer within a specific time frame, or returning a failure.

The second evidence compares the performances of SHOP2 and SIADEX [14,86]. The planners are tested on the Zeno travel domain [58] under a set of different temporal problems. In all cases, SIADEX outperforms the temporal version of SHOP2 [13]. Table 10b summarises the results of this comparison.

The third evidence compares the performances of SIPE-2 and Nonlin [9]. The planners are compared in the domain of house construction [7] in which SIPE-2 has four times better planning time than Nonlin for the same planning problem.

HTN planners, especially plan-based HTN ones, report obscure results about their performance. To the best of our knowledge, performance results for SIPE-2 in various domains are reported only in [9]. The runtime of the planner varies from one second up to six minutes for a “typical problem” in each domain. We also know that SIPE-2 is able to handle a domain

Table 11
Applications of HTN planners.

Domain	NOAH	Nonlin	SIPE-2	O-Plan2	SHOP2	SIADEX
Air-campaign planning			[87]	[88]		
Biological pathways				[89]		
Business process management						[90]
Construction planning		[7]	[9]	[10,69,91]		
Crisis management and logistics		[7]	[92]	[93–96]	[97,98,72]	[99]
E-learning						[100]
Geosocial networking					[72]	
Healthcare						[101,102]
Mission planning			[45,103]	[104,105]		
Equipment configuration			[68]			
Plan recognition					[106,107]	
Production planning	[5]		[9]	[108]	[72]	
Project planning		[7]		[11]	[109]	
Tourism planning						[110]
Mobile robot planning		[111]			[112,113]	
Service composition		[114]		[115]	[116]	[117]
Software system integration					[72]	
Ubiquitous computing					[118–120]	[121,122]
Video games					[123]	
Total applications	1	5	7	14	**16**	9

that includes up to 200 tasks, 500 objects with 10 to 15 properties per object, and a problem that includes a few thousand predicates [57].

Applicability We define a set of domains based on the set of applications we found in the literature. The purpose of domains is to help us cluster together applications with commonalities. Some domains may not be mutually exclusive, but, for simplicity, we dispose an application only to one domain. Table 11 shows the list of domains where each state-of-the-art HTN planner is applied. SHOP2 and O-Plan2 are the most widely used planners, SIPE-2 and SIADEX have also a relatively high number of applications. SIPE-2 is used in at least seven applications ranging from air-campaign planning, crisis management and logistics, mission planning and oil spills, to beer production. SIADEX is employed in at least nine applications ranging from business process management, fire forest fighting, e-learning, oncology treatment, planning tourist visits and Web service composition, to planning in ubiquitous computing. O-Plan2 is used in least 14 applications ranging from air-campaign planning, biological pathway discovery, house and space platform construction, crisis management and logistics, mission planning, production and project planning, to service composition. SHOP2 is used in at least 16 applications ranging from crisis management and logistics, location-based services, plan and goal recognition, production and project planning, mobile robot planning, Web service composition, to planning in ubiquitous computing and video games. Recalling our ultimate objective for AI planning, HTN planning has so far contributed by being employed to more than 50 applications. More than half of them are tackled with plan-based HTN planning, and SHOP2 appears to be the most applied HTN planner, while O-Plan2 is the most applied plan-based HTN planner.

In the following, we give a flavour of how HTN planning is applied to most of the domains. For the sake of readability, we choose to describe only one application per domain, that is, how a single HTN planner is used in a particular domain. The domains that are represented by a single application and for which we do not have access to the details are not described, but they are included in Table 11 for the sake of completeness.

Air-campaign planning. Air-campaign planning deals with achieving air superiority, which means having a set of actions executable by aircraft in order to provide protection from enemy threats or attacks [124]. A group of aircraft performs a mission described by its type, a time and a place, a type of aircraft, munition, and support missions, such as refuelling and reconnaissance [57]. SIPE-2 is employed with an objective to achieve air superiority, where the objective is represented as the highest-level task, which is further decomposed into two tasks solved separately [87]. These tasks can be decomposed in multiple ways reflecting different strategies and tactics. The tasks are decomposed to the level of mission tasks, which are decomposed into primitive tasks for individual missions (*e.g.,* an unmanned strike mission), and for their support missions (*e.g.,* reconnaissance). Primitive tasks include operations, such as strike, fighter sweep, escort, tanker orbits, *etc.*

Biological pathways. The domain of biological pathways deals with living systems that are able to self-organise at a molecular level according to the changes in their environment. The biological process by which cells detect, convert and transmit information with respect to environmental changes is called *signal transduction pathways* [89]. The task of signal transduction pathways discovery can be seen as a hierarchical planning problem. O-Plan2 is used to solve the problem and to provide its solution [89]. Compound tasks correspond to biological processes, while primitive tasks correspond to molecular-level functions of those processes. Individual pathways are represented as tasks that are combined in a hierarchical task network to form a larger pathway. This way, pathways can be encoded in multiple levels of abstraction.

Business process management. In the domain of business process management, a business process, such as the process of hotel reservation, is defined by a process model, and represented by a process instance at a specific time. A process model

may be associated with many distinct process instances whose deployment and execution depend on the organisational context at the moment of enactment [90]. The problem of finding a process instance suitable to a given context can be seen as a planning problem – finding a plan whose execution depends on the given context. Procedures, decisions and actions in the process model are encoded as compound tasks, methods and primitive tasks, respectively, in the hierarchical model of SIADEX. Each activity in the process model is represented as a primitive durative task. Workflow patterns and their control flow mechanisms are encoded as compound tasks with ordering constructs or with alternatives.

Construction planning. The domain of construction planning encompasses several sub-domains, namely house construction planning, space platform construction planning, and oil platform construction planning. Nonlin [7], SIPE-2 [9] and O-Plan2 [10] are employed in the domain of house construction planning. House building tasks, such as decorate, install services, and build house, are described as compound tasks. Primitive tasks, such as excavate, pour concrete foundations, *etc.* are encoded independently of their use in compound tasks. The domain employed by O-Plan2 includes temporal and resource constraints, and typed preconditions.

Crisis management and logistics. This domain encompasses those problems that deal with management of crisis situations, logistics, or both. SIPE-2 is employed in the domain of crisis action planning [92]. Given a crisis situation, the objective is to find flexible and accurate courses of actions that describe employment plans for dealing with one or more enemies, and identify deployment plans for disposing combat forces, supporting forces and equipment to their positions. The sort hierarchy of SIPE-2 is used to encode information about terrain analysis, combat forces, and so forth. Compound tasks encode abstract strategies, while primitive tasks represent specific military operations that can accomplish employment and deployment objectives.

E-learning. The objective of e-learning platforms is to provide students with customised learning processes. These platforms should consider various factors, such as heterogeneity of students, their study performance, needs and study [100]. The student data is stored in a repository in form of hierarchical and sequential meta-objects. The objects that have no children are encoded as primitive tasks. Atomic objects that are labelled with “optional” are represented by a compound task with two methods, one that includes the object corresponding primitive task, and the other that does not include the task. Every compound object is encoded as a compound task, while its child objects are represented as methods to this task. Such encoded domain knowledge is used by SIADEX to find a customised learning process.

Healthcare. One of the sub-domains of the healthcare domain is paediatric oncology, where planning can be employed to obtain therapy plans to treat and monitor patients. The therapy planning must be in accordance with a set of oncology treatment protocols which define various operating procedures and policies [101]. The main goal when planning for a treatment is to schedule chemotherapy, radiotherapy and patient evaluation sessions. The sessions must be in accordance with the workflow patterns, which define a set of tasks at different level of abstraction together with sequential, conditional, and iterative control flow constructs. Moreover, therapy actions, which are related to drug administration and patient evaluation, need to be performed with respect to various temporal constraints that describe their relative order and the delays between them. In SIADEX, the procedures, decisions and actions are encoded as compound tasks, methods and primitive tasks, respectively, taking into account the abstraction levels, control flow constructs and temporal constraints as defined in the oncology protocols.

Mission planning. The domain of mission planning encompasses several sub-domains. SHOP2 is employed in planning missions for unmanned aerial vehicle [98]. It is allowed to express temporal constraints related to wishes or requirements stated in mission requests. Elementary tasks, such as take off, go to, take shot, and wait, can be processed by any unmanned aerial vehicle present in the system. Furthermore, there are two types of operators. Actual operators mainly correspond to elementary tasks. However, these operators may also represent higher level tasks which cannot be decomposed only in the context of a single vehicle. Convenience operators, which are the second type, deal with intermediary data, and their application is required before applying actual operators.

Equipment configuration. The domain of equipment configuration involves planning for equipment deployment and employment. The equipment deployment and employment operations correspond to primitive tasks, while possible decision points together with constraints on equipment capabilities are encoded as compound tasks. SIPE-2 is employed for the purpose of allocation and siting of the equipment to clean up unanticipated oil spills [125,68]. The planner is used to keep track of a large number of equipment operations and the constraints of their use, and to help the analysis of spill responses to the level of detail of individual pieces of equipment.

Plan recognition. Plan recognition is the process of inferring the goal and plan of an agent by observing its actions. In order to support the recognition of plans, a set of labelled plans and their associated goals may be needed. SHOP2 is used to randomly generate sets of plans given some goal and an initial state [106]. The idea is to identify the key decision points (which task to do next, which method to use for a task, which value to bind for a parameter) in the planner and to randomise the order that they are searched in.

Production planning. The domain of production planning encompasses several sub-domains too. O-Plan2 is used to coordinate the production of an oil tanker truck [108]. Two types of tasks are encoded. Process tasks encode the knowledge about the production of parts of a tanker. Alternatives for producing the tanker oil vessel are represented as methods (by spinning, by pressing, by rolling and welding). Project tasks encode the knowledge about the steps needed to configure and make a truck.

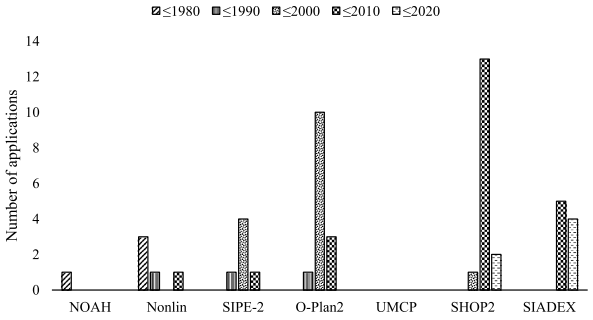


Fig. 7. Applicability of HTN planners from a time perspective.

Project planning. Project management is a business process for delivering products and services with time and resource constraints. An important element of project management is the project plan whose development represents a difficult task [109]. SHOP2 is used to support project management for which a so-called work breakdown structure is used [109]. This structure is a hierarchical organisation of elements, which can be either tasks or activities. An activity does not support attachment of additional elements. Tasks support activities and other tasks to be attached. The structure tasks are encoded as compound tasks, the activities are modelled as primitive tasks, while precedence constraints included in the structure are represented as ordering constraints.

Tourism planning. The domain of tourism planning involves finding a plan for a person to visit one or more tourist places. The plan, which provides information on how to go from one place to another, takes into account various places, distances between places, and timetables. SIADEX is employed to design tourist visits in which tasks can represent totally instantiated goals (*e.g.,* visit a specific museum), partially instantiated goals (*e.g.,* visit any museum), or abstract goals with different levels of granularity (*e.g.,* do a cultural visit) [110]. The activities, such as visit some place, attend some event, or move from one place to another, are encoded as primitive tasks.

Mobile robots planning. SHOP2 is used to generate robot plans by using knowledge that is not specific to any robot platform [113]. Robot tasks represent high-level specifications. Recipes are composed of a set of robot primitives or other recipes. These recipes are encoded as methods in compound tasks. The requirement is to have an explicit binding between the objects in recipes to each variable referenced in the preconditions of tasks (and effects for primitive tasks). The robot primitives, which represent actions directly executable by a robot, are encoded as primitive tasks.

Service composition. The domain of service composition deals with problems of how to assemble and connect services, such as computer services and Web services, in order to automate a particular task. For example, Nonlin is employed to compose computer services for an experimental parallel computer [114], while O-Plan2, SHOP2, and SIADEX are used to compose Web services. We cover the composition of Web services in details in the next section.

Ubiquitous computing. Ubiquitous computing characterises living spaces as environments equipped with various systems and devices that provide information about the current state of environments, and they may also help in adapting environments to new desired states. These environments are associated with a broad spectrum of properties, such as time-based events, spatial relationships between entities, uncertainty, *etc.* The most simple problem that planners have to solve is how to accomplish a specific behaviour, for example, a user request, given the information about the environment's current state and spectrum of properties. SHOP2 is employed to support the interoperation between networked robots and devices integrated in ubiquitous computing environments [118]. The knowledge about the environment is represented in the form of device services and service models. Device services represent the operations that devices can execute directly, while service models represent composite processes used to describe how services can be composed. Each device service is encoded as a primitive task, and a composite process is modelled as a compound task whose methods encode all constructs of the process.

Video games. Non-playing characters as computer-controlled entities in video games can be made more intelligent and interact with each other according to some plan. Instead of developing scripts with complete solutions, tasks can be used to abstract details away and provide a more general structure suitable to many types of characters and problems. For the problem of modelling such characters and finding a plan for the ElderScrolls IV: Oblivion game, JSHOP2 (the Java version of SHOP2) is employed [123]. The objective for the planner is to find a daily schedule in which each non-playing character performs an activity, such as eat and sleep. Each character has a set of packages and/or a script to control the character's behaviour. A package describes an activity. Scripts are used to control complex behaviours, such as dialogue.

Fig. 7 shows the applicability of HTN planners from a time perspective. We consider as a time point of a specific application the year when its publication has appeared, or when a reference about the application is made in some publication. Majority of applications are developed in the decade 1990–2000, the time when O-Plan2 was in its prime, and the decade 2000–2010, the time when SHOP2 was in its heyday. In the most recent times, two applications are implemented by SHOP2, and three applications by SIADEX.

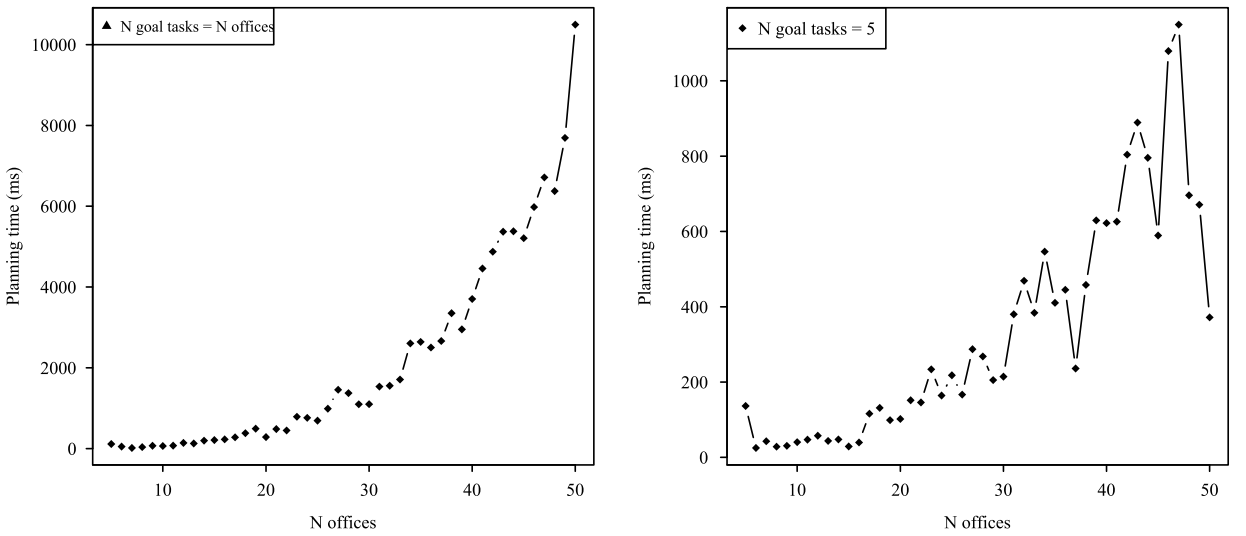


Fig. 8. Performance results when scaling the size of the initial task network (left plot), and scaling the number of offices under constant size of the initial task network (right plot).

4. HTN planning for Web service composition

Among the applications of HTN planning, Web service composition is a recent one that reflects many of the challenges that modern real-world domains are characterised with (see Section 1.3). HTN planning is recognised as a particularly suitable technique for Web service composition because it provides the following benefits [29]:

- Modularity – HTNs enable encoding knowledge at different levels of abstraction, and thus focusing on a particular level at a time. With the modularity, simple services and each aspect of complex services can be modelled as primitive and compound tasks. We witness the support and use of modularity in the discussion of the applications of HTN planning.
- Transparency – HTNs help to better interpret and understand different situations and decisions, which, in some cases, means a possibility to minimise some failures or costs.
- Scalability – HTN planning scales well to large number of tasks, but also generally to increasing size of a planning problem. In Section 3.2, however, we show that it is difficult to confirm practically the scalability statement. Therefore, we here provide some experimental results of an HTN planner, called Scalable Hierarchical (SH) planner, for composing Web services in smart offices [126]. In this domain, primitive tasks encode Representational State Transfer (REST) Web services [127], which represent operations of the office devices, and compound tasks encode services that represent processes and policies that an office should implement. The planner generates a sequence of device operations (*e.g.*, turn on/off a lamp/monitor) that adapts the office according to the policies and preferences. We generate two sets of different problems for which we change the load profiles. In the first set of tests, we evaluate the performance of SH in terms of scalability with respect to the number of tasks in the initial task network under increasing number of offices. That is, we are interested in the behaviour of the planner when there is an activity associated to each office. Fig. 8 shows the scaling of the planner. The left plot depicts the planning time under increasing number of tasks in the initial task network (goal tasks) and offices. We can notice that the planning time tends to be constant and near one second only up until 20 goal tasks. In the second set of tests, we evaluate the performance of SH in terms of scalability with respect to the number of offices under a constant number of tasks in the initial task network. In our sample domain, there could be 29 predicates that evaluate to true in total per office, which will be the actual increase when the number of offices is scaled. The right plot in Fig. 8 shows the planning time and indicates exponential increase where the worst execution time is just above one second.
- Knowledge acquisition – additional information can be gathered by invoking Web services during planning. This can be provided by using explicit conditions as discussed in Sections 2.2.3, 2.3.1, and 2.4.1.
- User/software intervention – preconditions support service invocations for some specific information to be provided by a person. Also, in cases when the planner is unable to plan, a person or a software agent can help to proceed with decomposition (see Section 3.2).

4.1. WSC framework

We establish a strong relationship between models of Web services and HTN planning, and assess how HTN planners address the challenges of Web service composition. For this purpose, we go one step back and propose the last slice of our pie which represents a general framework for Web service composition based on AI planning. The WSC framework is shown

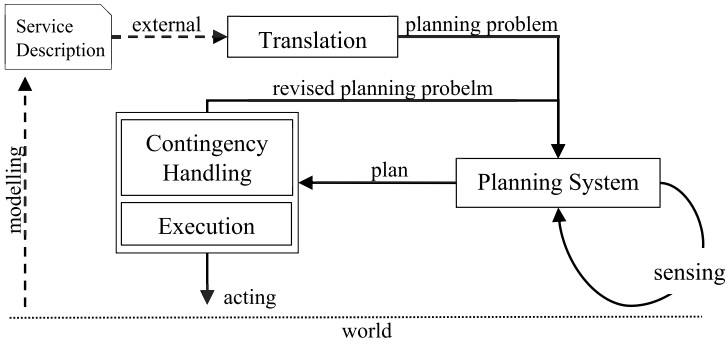


Fig. 9. WSC framework.

in Fig. 9. It provides an abstract view without requiring a particular description language, planning algorithm, or monitoring and execution approach. A Web service, which models some business logic, is described in some (external) language and provided to a translator that creates appropriate (internal) representation, that is, a planning problem. Consequently, the planning problem is given to a planner to search for a solution. If there is a solution found (*i.e.*, a plan), it is passed for execution and monitoring for potential faults. In case of a fault, appropriate actions are taken.

- **Service description**: The description of Web services offered to the global market usually consists of three parts. The first part refers to the information about the data transformation during the execution of a service. The information is presented in form of input, output and possibly exceptions. The input contains the information required for service execution, while the output presents the information the service provides after its execution. The second part refers to when and how a service transforms the world. This part consists of preconditions, that is, requirements that must be satisfied for the service to be invoked, and postconditions, that is, physical changes to be made to the world. The last part contains the non-functional properties of a service, such as cost, reliability, and service quality.
- **Translation**: Services described in a standard Web service language appear to be hard to handle by planning systems unless they are translated into an understandable form. The translation component accepts service descriptions and converts them into formal and unambiguous encoding. The result of the translation is a planning problem. In fact, this component enables the relationship between Web service composition and AI planning.
- **Planning System**: It takes the planning problem and tries to find a solution. Many planning systems distinguish between world-altering and sensing actions. The former can change the world when executed, while the latter cannot modify the state, but only acquire additional information needed to support the planning process. The most common approach is to perform off-line planning, that is, to simulate the execution of world-altering actions, and to do sensing. The solution, if it exists, consists of world-altering actions only. Many planning systems make several assumptions while planning. These assumptions simplify the planning process, but impose restrictions about what might happen in the world and distance further from the reality. The assumptions are:

  A1: The world is static – it can be modified only by the actions resulted from the planning process, and not by some external agent or event. All information about the world is expected to be valid till the end of the execution.

  A2: Sensing actions succeed – the execution of a sensing action will always return the acquired information.

  A3: Sensing actions are repeatable – the first sensed information is assumed to be valid for each action (service) further in the planning process.

  A4: No changes are made to services – service's functional properties are constant during the planning and execution processes.

- **Execution Monitoring and Contingency Handling**: Considering that the world is dynamic and uncertain, the execution of actions might not proceed as expected. A contingency may be inconsistent sensed information, failures of service invocations, timeouts, or unexpected change in the world. These observations suggest that the problem of Web service composition should not be tackled decoupled from the process of action execution. Monitoring of execution and contingency handling appears to be suitable to address the aforementioned issues. Execution monitoring checks the validity of off-line calculated actions when executed and, in case of contingency, reacts appropriately. For

example, if the execution time of some service takes too long, then it might be possible to proceed with the execution of subsequent actions. Other types of contingency may require repair of the existing plan, or even planning from scratch.

4.2. WSC problem as an HTN planning problem

Once the general steps of WSC are clear, we make a concrete and strong connection between WSC and HTN planning. That is, we choose OWL-S as a service description language upon which we define the problem of Web service composition and its corresponding HTN planning problem. OWL-S [128] is a Web ontology [129] for Web services used to support automated discovery, enactment and composition of Web services. The OWL-S ontology has three components: service profile, process model and service grounding. The service profile indicates the purpose of a service, and comprises the elements of part one and part three described in 'Service description' step in the framework. The process model indicates how to accomplish the service purpose, how to invoke the service, and what happens after the service execution. The service grounding specifies the way of interaction with the service, including a communication protocol.

The similarity of OWL-S with HTN planning lies in the services that OWL-S perceives as processes. OWL-S differentiates three classes of processes: atomic, simple and composite. An atomic process has no sub-processes, has a grounding associated with it, and can be executed in a single step. Then, a simple process provides an abstraction for an existing service, and has no associated grounding. Finally, a composite process consists of other processes via control constructs, such as sequence, split, any order, *etc.*

The services described in OWL-S need to be encoded in corresponding HTN elements. Intuitively, each atomic process is translated to an operator, and each simple and composite process is translated to a method [29]. If we consider that $\mathcal{P}^{\mathcal{W}} = (s_0, K, C)$ is a WSC problem described in OWL-S, where s_0 is an initial state of the world, K is a collection of OWL-S process models, and C is a composite OWL-S process defined in K, then the following relationship could be established (adopted from [130]).

Definition 15 *(WSC relationship to HTN planning).* Let $\mathcal{P}^{\mathcal{W}} = (s_0, K, C)$ be an OWL-S WSC problem. Then, the sequence $p_1, \ldots, p_n$, where each p_i is an atomic process defined in K is a solution to $\mathcal{P}^{\mathcal{W}}$ if and only if $t_1, \ldots, t_n$ is a solution to an HTN planning problem $\mathcal{P} = (Q, O, M, tn_0, s_0)$, where

- Q, O, M are generated by an OWL-S to HTN translation for the OWL-S process models K,
- tn_0 is generated by an OWL-S to HTN translation for the OWL-S process C, and
- each t_i is a primitive task that corresponds to an atomic process p_i defined by some OWL-S to HTN translation.

4.3. Review

Now that we have the Web service composition framework, and the relation between the WSC OWL-S problem and HTN planning problem, we are able to classify and compare several studies. Table 12 summarises the studies with respect to indicators extracted from the framework (a detailed discussion on each study can be found in [51]). Some indicators are associated with rates. The rates range from '★', indicating limited focus or limited support for a respective indicator, to '★★★', specifying comprehensive focus or extended support for the corresponding indicator. If a cell contains '✕', it means that the planner does not support the indicator under inspection. If a cell is empty, it denotes that we were not able to extract the information for the respective indicator from the literature. *Service description* provides the language for describing Web services assumed by the study being analysed, while *translation* gives the dual information. First, it indicates how well and exactly the translation process is described, and second, which format is the Web service description translated to. *HTN model* tells whether state-based HTN or plan-based HTN planning is employed, and which HTN planner is used for the implementation of the taken approach. Beside the extent to which it is supported, *sensing* may indicate whether the execution of a sensing action blocks the planning process, and whether sensing actions are performed during planning or they may be interleaved with world-altering ones during execution. *Assumptions* concern the degree of assumptions made to guarantee a successful composition with respect to composing, sensing and executing actions. *Contingencies* refers to unexpected behaviour of a composition at execution time, including Web service failures or time outs, and events or information changes made by some external agents. Each approach is evaluated with respect to the extent to which the support is implemented, and the type of contingency the approach can handle.

In Table 12, we group the studies into two categories. In the upper part, we analyse approaches that employ HTN planning exclusively in the attempt to solve the problem of Web service composition. In the lower part, we observe approaches that combine HTN planning with another technique, such as description logic and constraint satisfaction, to compose services.

Most of the approaches assume OWL-S description of Web services, and provide sound translation algorithms to appropriate internal representation. With respect to the HTN model, all approaches but one employ state-based HTN planning. From the state-based HTN approaches, one uses the SIADEX planner, one the SH planner, while the rest exploit SHOP, its Java version (JSHOP), or its successor SHOP2. From the plan-based HTN approaches, I-X/I-Plan is employed, which is an

Table 12
Summary of HTN-based approaches to Web service composition.

Study	Service description	Translation (Representation)	HTN model (Planner)	Sensing (Properties)	Assumptions	Contingencies (Types)
[116,29,32]	DAML-S [131] OWL-S	★★★ (SHOP2)	state-based (extended SHOP2)	★★ (blocking/non-blocking, during planning)	A1–A4	× (not discussed)
[132]		★ (SHOP)	state-based (extended SHOP)	★ (interleaving)	A1, A2	★★ (replanning for failures, time outs)
[117]	OWL-S	★★ (HPDL)	state-based (SIADEX)	★★ (during planning)		★★ (replanning for failures, time outs)
[133]	OWL-S	×	state-based (extended SHOP2)	× (not discussed)	A4, other	× (not discussed)
[134,135]	OWL-S	★★★ (SHOP2,PDDL,LTL)	state-based (extended SHOP2)	★★ (during planning)	A1–A3	× (not discussed)
[126]	REST	★ (HPDL)	state-based (SH)	× (not discussed)	A1–A4	× (not discussed)
[115]	OWL-S		plan-based (I-X/I-Plan)	★ (during planning)	A1	× (not discussed)
[136,137]	OWL-S	★★★ (SHOP,DL)	state-based (JSHOP + Pellet [138])	× (not discussed)		× (not discussed)
[36]		★ (SHOP2,CSP)	state-based (JSHOP2 + CSP Solver)	× (not discussed)		× (not discussed)
[139]	OWL-S	★★★ (SHOP,DL,PDDL)	state-based (extended JSHOP + Pellet)	× (not discussed)		× (not discussed)

HTN planner based on O-Plan2. Most of the approaches give actual contribution to HTN planning by extending the existing algorithms or providing new algorithms on top of the existing planners. With respect to sensing, only few approaches devote appropriate attention to it and provide clear description. We can observe and conclude that sensing is done during planning and, in some cases, in a non-blocking manner. While planning, sensing and possibly executing Web services, several approaches make some of the restricting assumptions, at least those that we were able to identify from the descriptions provided. Finally, little attention is devoted to execution monitoring and handling of contingencies at execution time.

Fig. 10 gives another perspective of approaches that assume OWL-S description of Web services and provide clear translation to the planning-level representation. The lower part specifies the studies that employ HTN planning only. Sirin et al. [29] appears to be the most influential and inspiring study. Two of them are a direct extension of the study, while the other two draw inspiration from the study with respect to the translation process. The upper part depicts the studies that combine HTN planning with DL reasoning. All studies are a continuation of the work presented in the first paper on HTN planning for Web service composition [116].

5. A broader view

While we covered many and diverse aspects of HTN planning, there are several research areas related to HTN planning that we have intentionally left out of the frameworks. These are important proposals of HTN planning that though focus on specific aspects of hierarchical planning. To provide a broader view of HTN planning, we consider them briefly here and also explain their relation with our frameworks.

Distributed planning refers to a setting in which the process of planning is distributed across several agents [140]. HTN planning is particularly suitable for cooperative distributed planning, which is based on the concept of task decomposition. In the cooperative distributed HTN planning, agents formulate and/or execute a plan by interacting between each other and decomposing their tasks based on the information that they acquire from other agents. Some efforts are made to develop distributed versions of several state-of-the-art HTN planners, namely NOAH [141], SIPE-2 [15], and SHOP2 [142,143]. There are also attempts to distribute HTN planning that do not directly relate to state-of-the-art HTN planners, such as [16,144]. We exclude distributed HTN planning from further examination specifically because we assume, contrary to the principles of distributed planning, that there is a single planning agent. In addition, many of distributed HTN planning approaches employ state-of-the-art HTN planners at their core which we cover in two of our frameworks. Generally, the problem of distributed planning is important in itself, therefore it can be considered as a separate framework.

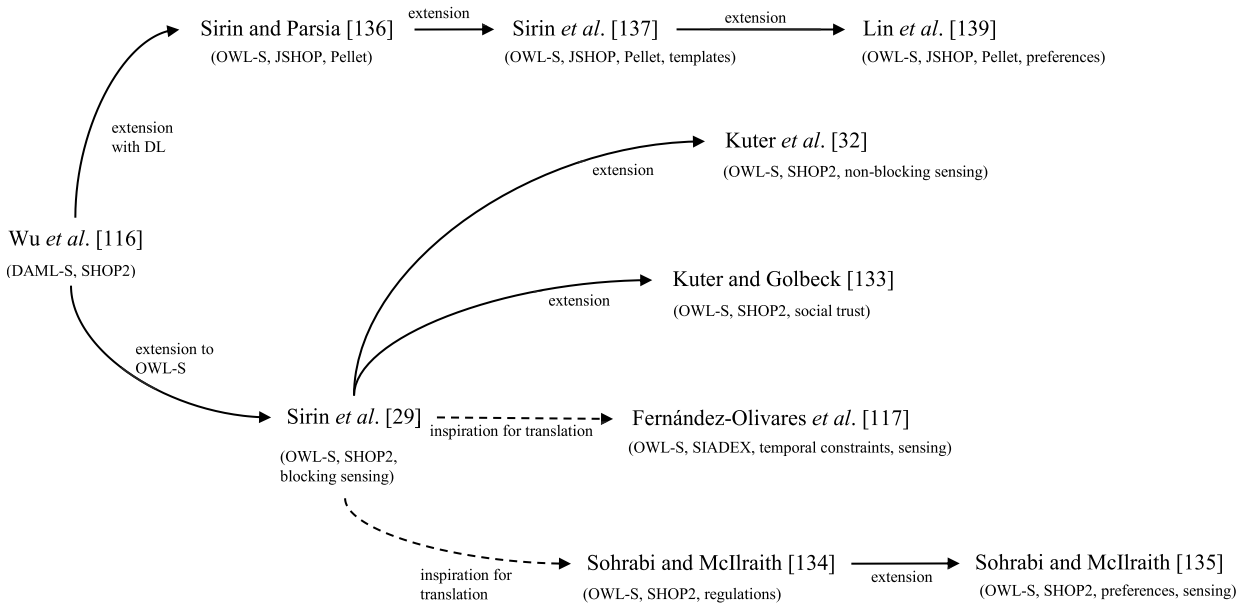


Fig. 10. Relations between studies that employ HTN planning for Web service composition.

Landmarks are facts that must be true in some state of every plan that is a solution to a planning problem [145]. In HTN planning, landmarks are abstract tasks that are mandatory and must be performed by any plan that solves a problem [17]. The landmarks are used to guide and speed up the hierarchical planning process [146]. In order to extract landmarks, a planning problem is pre-processed by systematically analysing the ways in which relevant tasks can be decomposed [147]. We do not discuss landmarks in HTN planning in details due to two factors. First, knowledge acquisition by pre-processing a planning problem is a proven and effective way to reduce the planning effort in classical planning. However, in HTN planning, the pre-processing of the domain knowledge so as to prune the search space is still a novel and emerging topic. The second and decisive factor is that landmarks, or domain exploitation in general, do not fit into our pie of frameworks. That is, theoretical contributions on landmarks can be seen as extensions rather than as postulates of our theoretical and conceptual frameworks. Moreover, the frameworks abstract away and analyse state-of-the-art HTN planners, and landmarks, however, are not applied to any of these planners.

Learning domain knowledge deals with techniques used to alleviate the burden of encoding domain knowledge needed by planners by automatically learning HTNs. There is a number of systems that use different approaches to learn domain knowledge from examples: one approach learns preconditions of SHOP-like methods given the method structure as input to the system [18], another one learns incrementally approximate preconditions [148], an approach learns very general HTNs by learning from expert traces [149], or another one that learns HTNs with a better balance between generality and specificity [150]. Some approaches acquire methods by analysing a set of planning problems together with their solutions and a set of annotated tasks in a given deterministic domain [19], and others may learn knowledge for domains that include primitive tasks with multiple possible outcomes [151]. Some recent approaches learn tasks by using a set of partially observed plan traces and a set of annotated tasks together with some constraints [152], or learn probabilistic HTNs that capture user preferences on plans by observing the user behaviour [153]. We acknowledge that learning domain knowledge is a relevant and crucial aspect for HTN planning, but this research area has its own specific aspects, such as learning algorithms and annotations, that are independent of HTN planning, thus we believe it would be more meaningful and useful to be analysed in a separate study.

Utility-based planning provides a framework for solving planning problems according to a given risk attitude [154]. For each such attitude, there is a utility function that transforms costs into real values with the purpose of maximising the expected utility. Utility-based HTN planning as a framework is proposed only recently [20]. A primitive task is associated with some function of consumption, while a compound task has a utility function to express the perceived utility of the task. The task network with the best estimated value is chosen for further processing. There are studies that take only costs of primitive tasks into account to search for the "best" plan [13,155]. In some studies, such as [16], the costs of compound tasks seem to have some decision-making role. Finally, a study that might fit into the utility-based framework is [133], where ratings of primitive tasks (*cf.* atomic services) are taken into account when estimating the trust of compound tasks (*cf.* composite services). Many of these studies do not deal with the state-of-the-art HTN planners, and most of them provide only minor theoretical foundations as a potential contribution to our theoretical framework. Therefore, we exclude this research area from further consideration.

6. Conclusions

While the composition of Web services is yet another example of the hierarchical organisation of the world, HTN planning provides the means to represent such hierarchies and to compose Web services into value-added functionalities. Beside appearing practically convenient, hierarchical planning is a long-lived technique characterised with controversy and confusion related to both theory and its use in the practice. We have shown that such a situation can be remedied. First of all, HTN planners are categorised based on the space they search in. We find that plan-based HTN planners need to search more complex spaces than state-based HTN planners. Then, several concepts that affect the planners' search space are identified. Most plan-based HTN planners lack the clarity and purpose of their concepts, especially the task decomposition and task interactions. We show that those concepts can be simplified but clarified, and logically interconnected. On the other hand, the concepts in state-based HTN planners are simple and coherent, but with some limitation – these planners lack the non-determinism in the task decomposition and commit early to variable bindings. However, the planners avoid most of task interactions and the confusion they may cause. Above all, we provide two formal models of HTN planning that comprise general ideas of the two categories of hierarchical planners.

Further analysis enabled us to interpret several practical issues of HTN planners. On an exemplifying domain, we found that plan-based HTN planners require smaller domain knowledge than state-based HTN planners, but that a prerequisite to encode the knowledge for the former ones is the comprehension of their underlying mechanisms. We know that HTN planning is more expressive than STRIPS-like planning in theory, but none of the hierarchical planners can achieve such expressiveness in practice. We show that both categories of planners have similar levels of expressiveness: they support a fragment of first-order logic, quality constraints are only partially addressed, and resource and temporal constraints are supported by few planners up to a certain level. The underlying problem-solving mechanisms of plan-based HTN planners are more complex than the ones of state-based HTN planners, and that only few planers show flexibility for incorporating alternative techniques. Furthermore and based on the systems of three hierarchical planners, we propose a general architecture for execution monitoring and replanning suitable for modern planners. With respect to scalability and efficiency of HTN planners, we stress that little is reported on their performance and pairwise comparison. As for the applicability, we find that HTN planning has been used in at least 50 domains, and SHOP2 is the most applied HTN planner, while O-Plan2 is the most applied plan-based HTN planner.

From the set of real-world applications, we select Web service composition and demonstrate its strong relationship with HTN planning. We show that HTN-based approaches to service composition must pre-process the external service representation in order to create a possibly correct internal presentation suitable for HTN planners. A large majority of approaches employ state-based HTN planners to compose Web services. Although suitable, all approaches appear to be limited in addressing the properties and issues of Web service environments.

Our theoretical framework, upon which we defined the two formal models, can support new models and algorithms for HTN planning, such as those recently proposed in [50]. New concepts, such as landmarks in HTN planning, can be later easily plugged into the framework of concepts as well. Our analysis shows that a common syntax and semantics for specifying HTN domains and problems is missing. A single description language for both categories of HTN planners seems illusory ambitious, but each category may make use of a description language à la PDDL. In this way, research improvements and performance evaluation of HTN planners can be stimulated, a direct comparison of the planners on possibly standardised set of problems can be enabled, and finally, help in understanding the expressive power of HTN planners can be provided. Furthermore, we recognise that HTN planners can be improved in the area of goals, such as extended goals, hybrid goals [156], and active goal reasoning [157].

The composition of Web services witness the need for 1) planners able to deal better with uncertainty, and 2) using planning before and during plan execution. The second point is in-line with the principles of continual planning [158]. Future HTN planners should monitor, refine, modify and repair their plans during the whole time of acting [159]. Many interesting research questions related to this subject remain open, such as a formalisation of execution semantics under the framework of HTN planning that define valid repairs of a solution plan at execution time, and development of an algorithm for plan repair (*e.g.*, similar to the orchestration algorithm in [160]) that is sound and complete with respect to the execution semantics.

Acknowledgements

We are deeply in debt to Austin Tate for the proactive and attentive stance toward this work, and for valuable comments on Nonlin and O-Plan2. We thank Luis Castillo Vidal for the constructive feedback on SIADEX. We are grateful to Luigia Carlucci Aiello for the helpful comments on the paper. We thank Eirini Kaldeli for the useful feedback on an earlier version of the paper. The work is supported by the Dutch National Research Council under the NWO Smart Energy Systems program, contract no. 647.000.004.

References

[1] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning: Theory & Practice, Morgan Kaufmann Publishers Inc., 2004.
[2] D.S. Nau, Y. Cao, A. Lotem, H. Muñoz Avila, SHOP: simple hierarchical ordered planner, in: International Joint Conference on Artificial Intelligence, IJCAI'99, Morgan Kaufmann Publishers Inc., 1999, pp. 968–975.

[3] F. Bacchus, The AIPS '00 planning competition, AI Mag. 22 (3) (2001) 47–56.
[4] D.S. Nau, Current trends in automated planning, AI Mag. 28 (4) (2007) 43–58.
[5] E.D. Sacerdoti, A structure for plans and behavior, Ph.D. thesis, Standford University, AI Center, 1975, aAI7605794.
[6] E.D. Sacerdoti, The nonlinear nature of plans, in: International Joint Conference on Artificial Intelligence – Volume 1, IJCAI'75, Morgan Kaufmann Publishers Inc., 1975, pp. 206–214.
[7] A. Tate, Project planning using a hierarchic non-linear planner, Tech. Rep. 25, Department of Artificial Intelligence, University of Edinburgh, 1976.
[8] A. Tate, Generating project networks, in: International Joint Conference on Artificial Intelligence – Volume 2, IJCAI'77, Morgan Kaufmann Publishers Inc., 1977, pp. 888–893.
[9] D.E. Wilkins, Can AI planners solve practical problems?, Comput. Intell. 6 (1991) 232–246.
[10] K. Currie, A. Tate, O-Plan: the open planning architecture, Artif. Intell. 52 (1) (1991) 49–86.
[11] A. Tate, B. Drabble, R. Kirby, O-Plan2: an open architecture for command, planning and control, in: Intelligent Scheduling, Morgan Kaufmann Publishers Inc., 1994, pp. 213–239.
[12] K. Erol, Hierarchical task network planning: formalization, analysis, and implementation, Ph.D. thesis, Computer Science Department, University of Maryland, 1996.
[13] D.S. Nau, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, F. Yaman, SHOP2: an HTN planning system, J. Artif. Intell. Res. 20 (1) (2003) 379–404.
[14] L.A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, F. Palao, Temporal enhancements of an HTN planner, in: Conference of the Spanish Association for Artificial Intelligence, Current Topics in AI, Springer, 2005, pp. 429–438.
[15] M. desJardins, M. Wolverton, Coordinating a distributed planning system, AI Mag. 20 (4) (1999) 45–53.
[16] F. Amigoni, N. Gatti, C. Pinciroli, M. Roveri, What planner for ambient intelligence applications?, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 35 (1) (2005) 7–21.
[17] M. Elkawkagy, B. Schattenberg, S. Biundo, Landmarks in hierarchical planning, in: European Conference on Artificial Intelligence, IOS Press, 2010, pp. 229–234.
[18] O. Ilghami, D.S. Nau, CaMeL: learning method preconditions for HTN planning, in: International Conference on AI Planning and Scheduling, 2002, pp. 131–141.
[19] C. Hogg, H. Muñoz Avila, U. Kuter HTN-MAKER, Learning HTNs with minimal additional knowledge engineering required, in: National Conference on Artificial Intelligence – Volume 2, AAAI, 2008, pp. 950–956.
[20] I. Georgievski, A. Lazovik, Utility-based HTN planning, in: European Conference on Artificial Intelligence, 2014, pp. 1013–1014.
[21] M.P. Papazoglou, D. Georgakopoulos, Introduction: service-oriented computing, Commun. ACM 46 (10) (2003) 24–28.
[22] S. Bleul, T. Weise, K. Geihs, The Web Service Challenge – a review on Semantic Web service composition, in: Service-Oriented Computing, SOC'2009, 2009, pp. 1–12.
[23] The Web Services Challenge, Online: accessed Jan. 2014, http://www.wschallenge.org/.
[24] M.B. Blake, W. Cheung, M.C. Jaeger, A. Wombacher, WSC-06: The Web Service Challenge, in: CEC/EEE, 2006, pp. 1–2.
[25] M. Aiello, N. van Benthem, E. el Khoury, Visualizing compositions of services from large repositories, in: IEEE CEC/EEE, 2008, pp. 359–362.
[26] M. Aiello, E. el Khoury, A. Lazovik, P. Ratelband, Optimal QoS-aware Web service composition, in: IEEE CEC/EEE, 2009, pp. 491–494.
[27] V. Degeler, I. Georgievski, A. Lazovik, M. Aiello, Concept mapping for faster QoS-aware Web service composition, in: IEEE Conference on Service Oriented Computing and Applications, 2010, pp. 1–4.
[28] M. Aiello, M.P. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serafini, P. Traverso, A request language for Web services based on planning and constraint satisfaction, in: International Workshop on Technologies for E-Services, Springer, 2002, pp. 76–85.
[29] E. Sirin, B. Parsia, D. Wu, J. Hendler, D.S. Nau, HTN planning for Web service composition using SHOP2, J. Web Semant. 1 (2004) 377–396.
[30] A. Lazovik, M. Aiello, M. Papazoglou, Associating assertions with business processes and monitoring their execution, in: International Conference on Service-Oriented Computing, ICSOC '04, ACM, 2004, pp. 94–104.
[31] S. Dustdar, W. Schreiner, A survey on Web services composition, Int. J. Web Grid Serv. 1 (1) (2005) 1–30.
[32] U. Kuter, E. Sirin, B. Parsia, D. Nau, J. Hendler, Information gathering during planning for Web service composition, J. Web Semant. 3 (2005) 183–205.
[33] B. Medjahed, A. Bouguettaya, A multilevel ccmposability model for Semantic Web services, IEEE Trans. Knowl. Data Eng. 17 (2005) 954–968.
[34] M. Klusch, A. Gerber, Semantic Web service composition planning with OWLS-XPlan, in: International AAAI Fall Symposium on Agents and the Semantic Web, AAAI, 2005, pp. 55–62.
[35] S. Sohrabi, N. Prokoshyna, S.A. Mcilraith, Web service composition via generic procedures and customizing user preferences, in: International Semantic Web Conference, ISWC'06, Springer, 2006, pp. 597–611.
[36] I. Paik, D. Maruyama, Automatic Web services composition using combining HTN and CSP, in: IEEE International Conference on Computer and Information Technology, IEEE Computer Society, 2007, pp. 206–211.
[37] E. Kaldeli, A. Lazovik, M. Aiello, Extended goals for composing services, in: International Conference on Automated Planning and Scheduling, ICAPS'09, AAAI, 2009, pp. 362–365.
[38] E. Kaldeli, A. Lazovik, M. Aiello, Continual planning with sensing for Web service composition, in: AAAI Conference on Artificial Intelligence, AAAI, 2011, pp. 1198–1203.
[39] K. Erol, J. Hendler, D.S. Nau, UMCP: a sound and complete procedure for hierarchical task-network planning, in: International Conference on AI Planning Systems, Morgan Kaufmann Publishers Inc., 1994, pp. 249–254.
[40] T. Geier, P. Bercher, On the decidability of HTN planning with task insertion, in: International Joint Conference on Artificial Intelligence – Volume 3, IJCAI'11, AAAI, 2011, pp. 1955–1961.
[41] M. Stefik, Planning with constraints (MOLGEN: Part 1), Artif. Intell. 16 (2) (1981) 111–140.
[42] D.S. Weld, An introduction to least commitment planning, AI Mag. 15 (4) (1994) 27–61.
[43] D.E. Foulser, M. Li, Q. Yang, Theory and algorithms for plan merging, Artif. Intell. 57 (1992) 143–181.
[44] Q. Yang, A theory of conflict resolution in planning, Artif. Intell. 58 (1) (1992) 361–392.
[45] D.E. Wilkins, Practical Planning: Extending the Classical AI Planning Paradigm, Morgan Kaufmann Publishers Inc., 1988.
[46] K. Erol, J. Hendler, D.S. Nau, Semantics for hierarchical task network planning, Tech. Rep. UMIACS-TR-94-31, University of Maryland, Institute for Advanced Computer Studies, 1994.
[47] A. Nareyek, E.C. Freuder, R. Fourer, E. Giunchiglia, R.P. Goldman, H. Kautz, J. Rintanen, A. Tate, Constraints and AI planning, IEEE Intell. Syst. 20 (2005) 62–72.
[48] I. Georgievski, A. Lazovik, M. Aiello, Task interaction in an HTN planner, Tech. rep, arXiv:1111.7025 [CoRR], 2011.
[49] R.E. Fikes, N.J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, in: International Joint Conference on Artificial Intelligence, IJCAI'71, Morgan Kaufmann Publishers Inc., 1971, pp. 608–620.
[50] R. Alford, V. Shivashankar, U. Kuter, D.S. Nau, HTN problem spaces: structure, algorithms, termination, in: Annual Symposium on Combinatorial Search, AAAI, 2012, pp. 2–9.
[51] I. Georgievski, M. Aiello, An overview of hierarchical task planning, Tech. rep., arXiv:1403.7426 [CoRR], 2014.
[52] R. Tsuneto, K. Erol, J. Hendler, D.S. Nau, Commitment strategies in hierarchical task network planning, in: National Conference on Artificial Intelligence – Volume 1, AAAI'96, AAAI, 1996, pp. 536–542.

[53] R. Tsuneto, J. Hendler, D.S. Nau, Analyzing external conditions to improve the efficiency of HTN planning, in: National Conference on Artificial Intelligence/Conference on Innovative Applications of Artificial Intelligence, AAAI, 1998, pp. 913–920.
[54] A. Tate, B. Drabble, J. Dalton, The use of condition types to restrict search in an AI planner, in: National Conference on Artificial Intelligence – Volume 2, AAAI'94, AAAI, 1994, pp. 1129–1134.
[55] I. Bratko, Prolog: Programming for Artificial Intelligence, 3rd ed., Addison-Wesley Longman Publishing Co., Inc., 2001.
[56] Python, Online: accessed Jan. 2014, http://www.python.org/getit/releases/3.4.0/.
[57] D.E. Wilkins, M. Desjardins, A call for knowledge-based planning, AI Mag. 22 (1) (2001) 99–115.
[58] D. Long, M. Fox, The 3rd International Planning Competition: results and analysis, J. Artif. Intell. Res. 20 (2003) 1–59.
[59] S.V. Chenoweth, On the NP-hardness of blocks world, in: National Conference on Artificial Intelligence, AAAI'91, 1991, pp. 623–627.
[60] K. Erol, J.A. Hendler, D.S. Nau, HTN planning: complexity and expressivity, in: National Conference on Artificial Intelligence – Volume 2, AAAI'94, AAAI, 1994, pp. 1123–1128.
[61] S. Kambhampati, A comparative analysis of partial order planning and task reduction planning, SIGART Bull. 6 (1995) 16–25.
[62] D.S. Nau, S.J. Smith, K. Erol, Control strategies in HTN planning: theory versus practice, in: National Conference on Artificial Intelligence/Conference on Innovative applications of Artificial Intelligence, AAAI'98/IAAI'98, AAAI, 1998, pp. 1127–1133.
[63] M. Lekavý, P. Návrat, Expressivity of STRIPS-like and HTN-like planning, in: KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications, KES-AMSTA'07, Springer, 2007, pp. 121–130.
[64] K. Erol, J. Handler, D.S. Nau, Complexity results for HTN planning, Ann. Math. Artif. Intell. 18 (1) (1996) 69–93.
[65] M. Ghallab, C.K. Isi, S. Penberthy, D.E. Smith, Y. Sun, D. Weld, PDDL – The Planning Domain Definition Language, Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
[66] M. Fox, D. Long, PDDL2.1: an extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (1) (2003) 61–124.
[67] A. Gerevini, D. Long, Preferences and soft constraints in PDDL3, in: ICAPS Workshop on Planning with Preferences and Soft Constraints, 2006.
[68] J.M. Agosta, Constraining influence diagram structure by generative planning: an application to the optimization of oil spill response, in: International Conference on Uncertainty in Artificial Intelligence, UAI'96, Morgan Kaufmann Publishers Inc., 1996, pp. 11–19.
[69] M. Aarup, M.M. Arentoft, Y. Parrod, I. Stokes, H. Vadon, J. Stader, OPTIMUM-AIV: a knowledge-based planning and scheduling system for spacecraft AIV, in: Conference on Knowledge Based Scheduling, Morgan Kaufmann Publishers Inc., 1994, pp. 451–469.
[70] S.J.J. Smith, K. Hebbar, D.S. Nau, I. Minis, Integrating electrical and mechanical design and process planning, 1997.
[71] E. Kaldeli, E.U. Warriach, A. Lazovik, M. Aiello, Coordinating the Web of services for a smart home, ACM Trans. Web 7 (2) (2012) 10:1–10:40.
[72] D.S. Nau, T.C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Muñoz Avila, J.W. Murdock, Applications of SHOP and SHOP2, IEEE Intell. Syst. 20 (2) (2005) 34–41.
[73] V. Shivashankar, U. Kuter, D.S. Nau, R. Alford, A hierarchical goal-based formalism and algorithm for single-agent planning, in: International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 981–988.
[74] B. Schattenberg, Hybrid planning and scheduling, Ph.D. thesis, Institute of Artificial Intelligence, Ulm University, 2009.
[75] K.L. Myers, Strategic advice for hierarchical planners, in: International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann Publishers Inc., 1996, pp. 112–123.
[76] K.L. Myers, Planning with conflicting advice, in: International Conference on Artificial Intelligence Planning Systems, AAAI, 2000, pp. 355–362, poster paper.
[77] S. Sohrabi, S.A. Mcilraith, On planning with preferences in HTN, in: International Workshop on Non-Monotonic Reasoning, 2008, pp. 241–248.
[78] S. Sohrabi, J.A. Baier, S.A. McIlraith, HTN planning with preferences, in: International Joint Conference on Artificial Intelligence, IJCAI'09, Morgan Kaufmann Publishers Inc., 2009, pp. 1790–1797.
[79] J. Stillman, R. Arthur, A. Deitsch, Tachyon: a constraint-based temporal reasoning model and its implementation, SIGART Bull. 4 (3) (1993) 1–4.
[80] F. Yaman, D.S. Nau, Timeline: an HTN planner that can reason about time, in: AIPS'02 Workshop on Planning for Temporal Domains, 2002, pp. 75–81.
[81] R.P. Goldman, Durative planning in HTNs, in: International Conference on Automated Planning and Scheduling, ICAPS'06, AAAI, 2006, pp. 382–385.
[82] J.F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.
[83] J.F.A.K. van Benthem, The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, 2nd edition, Kluwer Academic Publishers, Dordrecht, 1991.
[84] D.S. Nau, Y. Cao, A. Lotem, H. Muñoz Avila, SHOP and M-SHOP: planning with ordered task decomposition, Tech. Rep. CS-TR-4157, Computer Science Department, University of Maryland, 2000.
[85] S. Andrews, B. Kettler, K. Erol, J. Hendler, UM Translog: a planning domain for the development and benchmarking of planning systems, Tech. Rep. CS-TR-3487, Computer Science Department, University of Maryland, 1995.
[86] L.A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, F. Palao, Efficiently handling temporal knowledge in an HTN planner, in: International Conference on Automated Planning and Scheduling, ICAPS'06, AAAI, 2006, pp. 63–72.
[87] T.J. Lee, D. Wilkins, Using SIPE-2 to integrate planning for military air campaigns, IEEE Expert 11 (6) (1996) 11–12.
[88] A. Tate, J. Dalton, J. Levine, Generation of multiple qualitatively different plan options, in: International Conference on AI Planning Systems, AAAI, 1998, pp. 27–35.
[89] S. Khan, W. Gillis, C. Schmidt, K. Decker, A multi-agent system-driven AI planning approach to biological pathway discovery, in: International Conference on Automated Planning and Scheduling, ICAPS'03, AAAI, 2003, pp. 246–255.
[90] A. González-Ferrer, J. Fernández-Olivares, L. Castillo, From business process models to hierarchical task network planning domains, Knowl. Eng. Rev. 28 (2) (2013) 175–193.
[91] B. Drabble, J. Dalton, A. Tate, Repairing plans on-the-fly, in: NASA Workshop on P&S for Space, 1997.
[92] D. Wilkins, R.V. Desimone, Applying an AI planner to military operations planning, in: Intelligent Scheduling, Morgan Kaufmann Publishers Inc., 1992, pp. 685–709.
[93] A. Tate, B. Drabble, J. Dalton, O-Plan: a knowledge-based planner and its application to logistics, in: ARPI, 1996, pp. 259–266.
[94] J. Kingston, N. Shadbolt, A. Tate, CommonKADS models for knowledge based planning, in: National Conference on Artificial Intelligence, AAAI'96, AAAI, 1996, pp. 477–482.
[95] A. Tate, J. Dalton, J. Levine, O-Plan: a Web-based AI planning agent, in: National Conference on Artificial Intelligence/Conference on Innovative Applications of Artificial Intelligence, AAAI'00/IAAI'00, AAAI, 2000, pp. 1131–1132.
[96] A. Tate, J. Dalton, O-Plan: a common Lisp planning Web service, in: International Lisp Conference, 2003.
[97] H. Muñoz-Avila, D.W. Aha, L. Breslow, D.S. Nau, HICAP: an interactive case-based planning architecture and its application to noncombatant evacuation operations, 1999.
[98] J. Gancet, G. Hattenberger, R. Alami, S. Lacroix, Task planning and control for a multi-UAV system: architecture and algorithms, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, pp. 1017–1022.
[99] M. de la Asunción, L. Castillo, J. Fernádez-Olivares, O. García-Pérez, A. González, F. Palao, SIADEX: an interactive artificial intelligence planner for decision support and training in forest fire fighting, AI Commun. 18 (4) (2005) 257–268.
[100] L. Castillo, L. Morales, A. González-Ferrer, J. Fernández-Olivares, D. Borrajo, E. Onaindía, Automatic generation of temporal planning domains for e-learning problems, J. Sched. 13 (4) (2010) 347–362.

[101] J. Fernández-Olivares, J.A. Cózar, L. Castillo, Automating oncology therapy plans by means of temporal hierarchical task networks planning, in: ECAI Workshop on Knowledge Management for Healthcare Processes, 2008.
[102] I. Sánchez-Garzón, J. Fernández-Olivares, L. Castillo, An approach for representing and managing medical exceptions in care pathways based on temporal hierarchical planning techniques, in: R. Lenz, S. Miksch, M. Peleg, M. Reichert, D. Riano, A. Teije (Eds.), Process Support and Knowledge Representation in Health Care, in: Lecture Notes in Computer Science, vol. 7738, Springer, 2013, pp. 168–182.
[103] D.E. Wilkins, T. Lee, P. Berry, Interactive execution monitoring of agent teams, J. Artif. Intell. Res. 18 (2003) 217–261.
[104] M.E. Drummond, K.W. Currie, A. Tate, O-Plan meets T-SAT: first results from the application of an AI planner to spacecraft mission sequencing, Tech. rep., Artificial Intelligence Applications Institute, University of Edinburgh, 1988.
[105] A. Tate, J. Levine, P. Jarvis, J. Dalton, Using AI planning technology for army small unit operations, in: Artificial Intelligence Planning and Scheduling Systems Conference, AAAI, 2000, pp. 379–386, poster paper.
[106] N. Blaylock, J. Allen, Generating artificial corpora for plan recognition, in: International Conference on User Modeling, UM'05, Springer, 2005, pp. 179–188.
[107] N. Blaylock, J. Allen, Hierarchical instantiated goal recognition, in: AAAI Workshop on Modeling Others from Observations, 2006, pp. 8–15.
[108] A. Tate, Key concepts in the O-Plan2 knowledge based plan representation, in: UK Planning and Scheduling, 1994.
[109] H. Muñoz-Avila, K. Gupta, D.W. Aha, D.S. Nau, Knowledge-based project planning, in: R. Dieng-Kuntz, N. Matta (Eds.), Knowledge Management and Organizational Memories, Springer, 2002, pp. 125–134.
[110] L. Castillo, E. Armengol, E. Onaindía, L. Sebastiá, J. González-Boticario, A. Rodríguez, S. Fernández, J.D. Arias, D. Borrajo, Samap: an user-oriented adaptive system for planning tourist visits, Expert Syst. Appl. 34 (2) (2008) 1318–1332.
[111] A. Tate, Planning and doing things, AISB Q. (2007) 7–8.
[112] M. Weser, D. Off, J. Zhang, HTN robot planning in partially observable dynamic environments, in: International Conference on Robotics and Automation, 2010, pp. 1505–1510.
[113] D. Di Marco, R. Janssen, A. Perzylo, M.J. Van de Molengraft, P. Levi, A deliberation layer for instantiating robot execution plans from abstract task descriptions, in: International Conference on Automated Planning and Scheduling, Workshop on Planning and Robotics, 2013, pp. 12–19.
[114] A. Tate, D. Lesley, A retrospective on the 'Planning: A Joint AI/OR Approach' project, Tech. Rep. Working paper 125, Department of Artificial Intelligence, University of Edinburgh, 1982.
[115] A. Uszok, J.M. Bradshaw, R. Jeffers, A. Tate, J. Dalton, Applying KAoS services to ensure policy compliance for Semantic Web services workflow composition and enactment, in: International Semantic Web Conference, Springer, 2004, pp. 425–440.
[116] D. Wu, B. Parsia, E. Sirin, J. Hendler, D.S. Nau, Automating DAML-S Web services composition using SHOP2, in: International Semantic Web Conference, ISWC'03, Springer, 2003, pp. 195–210.
[117] J. Fernández-Olivares, T. Garzón, L. Castillo, O. García-Pérez, F. Palao, A middle-ware for the automated composition and invocation of Semantic Web services based on temporal HTN planning techniques, in: D. Borrajo, L. Castillo, J. Corchado (Eds.), Current Topics in Artificial Intelligence, in: Lecture Notes in Computer Science, vol. 4788, Springer, 2007, pp. 70–79.
[118] Y.-G. Ha, J.-C. Sohn, Y.-J. Cho, H. Yoon, Towards a ubiquitous robotic companion: design and implementation of ubiquitous robotic service framework, ETRI J. 27 (6) (2005) 666–676.
[119] F. Marquardt, C. Reisse, A. Uhrmacher, T. Kirste, A two-way approach to service composition in smart device ensembles, in: Advanced Topics in Telecommunication, 2008.
[120] S. Song, S.-W. Lee, A goal-driven approach for adaptive service composition using planning, Math. Comput. Model. 58 (1–2) (2013) 261–273.
[121] E. Hidalgo, L. Castillo, R.I. Madrid, O. García-Pérez, M. Cabello, J. Fdez-Olivares, ATHENA: smart process management for daily activity planning for cognitive impairment, in: J. Bravo, R. Hervás, V. Villarreal (Eds.), Ambient Assisted Living, in: Lecture Notes in Computer Science, vol. 6693, 2011, pp. 65–72.
[122] I. Sánchez-Garzón, G. Milla-Millán, J. Fernández-Olivares, Context-aware generation and adaptive execution of daily living care pathways, in: International Conference on Ambient Assisted Living and Home Care, Springer, 2012, pp. 362–370.
[123] J.-P. Kelly, A. Botea, S. Koenig, Offline planning with hierarchical task networks in video games, in: Artificial Intelligence and Interactive Digital Entertainment Conference, 2008.
[124] T.J. Lee, The air campaign planning knowledge base, Tech. Rep. Version 4.2, Artificial Intelligence Center, SRI International, 1999.
[125] J.M. Agosta, Formulation and implementation of an equipment configuration problem with the (SIPE-2) generative planner, in: AAAI-95 Spring Symposium on Integrated Planning Applications, 1995, pp. 1–10.
[126] I. Georgievski, T.A. Nguyen, M. Aiello, Combining activity recognition and AI planning for energy-saving offices, in: International Conference on Ubiquitous Intelligence and Computing, 2013, pp. 238–245.
[127] L. Richardson, S. Ruby, Restful Web Services, 1st edition, O'Reilly, 2007.
[128] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D.L. Mcguinness, E. Sirin, N. Srinivasan, Bringing semantics to Web services with OWL-S, World Wide Web 10 (3) (2007) 243–277.
[129] I. Horrocks, P.F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: the making of a Web Ontology Language, J. Web Semant. 1 (1) (2003) 7–26.
[130] S. Sohrabi, Customizing the composition of Web services and beyond, Ph.D. thesis, Depart. of Computer Science, Univ. of Toronto, 2013.
[131] M.H. Burstein, J.R. Hobbs, O. Lassila, D. Martin, D.V. McDermott, S.A. McIlraith, S. Narayanan, M. Paolucci, T.R. Payne, K.P. Sycara, DAML-S: web service description for the Semantic Web, in: International Semantic Web Conference on the Semantic Web, ISWC '02, Springer, 2002, pp. 348–363.
[132] T. Madhusudan, N. Uttamsingh, A declarative approach to composing Web services in dynamic environments, Decis. Support Syst. 41 (2) (2006) 325–357.
[133] U. Kuter, J. Golbeck, Semantic Web service composition in social environments, in: International Semantic Web Conference, ISWC '09, Springer, 2009, pp. 344–358.
[134] S. Sohrabi, S.A. Mcilraith, Optimizing Web service composition while enforcing regulations, in: International Semantic Web Conference, ISWC '09, Springer, 2009, pp. 601–617.
[135] S. Sohrabi, S.A. McIlraith, Preference-based Web service composition: a middle ground between execution and search, in: International Semantic Web Conference on the Semantic Web – Volume Part I, ISWC'10, Springer, 2010, pp. 713–729.
[136] E. Sirin, B. Parsia, Planning for Semantic Web services, in: Semantic Web Services Workshop at 3rd ISWC, 2004.
[137] E. Sirin, B. Parsia, J. Hendler, Template-based composition of Semantic Web services, in: AAAI Fall Symposium on Agents and the Semantic Web, AAAI, 2005, pp. 85–92.
[138] Pellet: OWL DL reasoner, Online: accessed Jan. 2014, http://clarkparsia.com/pellet/.
[139] N. Lin, U. Kuter, E. Sirin, Web service composition with user preferences, in: European Semantic Web Conference on the Semantic Web: Research and Applications, ESWC'08, Springer, 2008, pp. 629–643.
[140] M. desJardins, E. Durfee, C.J. Ortiz, M. Wolverton, A survey of research in distributed, continual planning, AI Mag. 20 (4) (1999).
[141] D.D. Corkill, Hierarchical planning in a distributed environment, in: International Joint Conference on Artificial intelligence – Volume 1, IJCAI'79, Morgan Kaufmann Publishers Inc., 1979, pp. 168–175.

[142] J.D. Dix, H. Muñoz Avila, D.S. Nau, L. Zhang, IMPACTing SHOP: putting an AI planner into a multi-agent environment, Ann. Math. Artif. Intell. 37 (4) (2003) 381–407.
[143] S. Lu, DSHOP: distributed simple hierarchical ordered planner, Master's thesis, University of Windsor, 2004.
[144] S. Magnenat, M. Voelkle, F. Mondada, Planner9, a HTN planner distributed on groups of miniature mobile robots, in: International Conference on Intelligent Robotics and Applications, ICIRA'09, Springer, 2009, pp. 1013–1022.
[145] J. Porteous, L. Sebastia, J. Hoffmann, On the extraction, ordering and usage of landmarks in planning, in: European Conference of Planning, 2001, pp. 37–48.
[146] M. Elkawkagy, P. Bercher, B. Schattenberg, S. Biundo, Improving hierarchical planning performance by the use of landmarks, in: AAAI Conference on Artificial Intelligence, 2012, pp. 1763–1769.
[147] M. Elkawkagy, P. Bercher, B. Schattenberg, S. Biundo, Landmark-aware strategies for hierarchical planning, in: Workshop on Heuristics for Domain-independent Planning (HDIP 2011) at ICAPS 2011, 2011, pp. 73–79.
[148] O. Ilghami, D.S. Nau, D.W. Aha, Learning preconditions for planning from plan traces and HTN structure, Comput. Intell. 21 (4) (2005) 388–413.
[149] N. Nejati, P. Langley, T. Konik, Learning hierarchical task networks by observation, in: International Conference on Machine Learning, ACM, 2006, pp. 665–672.
[150] N. Nejati, T. Könik, U. Kuter, A goal- and dependency-directed algorithm for learning hierarchical task networks, in: International Conference on Knowledge Capture, ACM, 2009, pp. 113–120.
[151] C. Hogg, U. Kuter, H. Muñoz Avila, Learning hierarchical task networks for non-deterministic planning domains, in: International Joint Conference on Artificial Intelligence, IJCAI'09, Morgan Kaufmann Publishers Inc., 2009, pp. 1708–1714.
[152] H.H. Zhuo, H. Muñoz Avila, Q. Yang, Learning hierarchical task network domains from partially observed plan traces, Artif. Intell. 212 (0) (2014) 134–157.
[153] N. Li, W. Cushing, S. Kambhampati, S. Yoon, Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences, ACM Trans. Intell. Syst. Technol. 5 (2) (2014) 29:1–29:32.
[154] S. Koenig, R.G. Simmons, Risk-sensitive planning with probabilistic decision graphs, in: International Conference on Principles of Knowledge Representation and Reasoning, 1994, pp. 363–373.
[155] J. Luo, C. Zhu, W. Zhang, Z. Liu, Messy genetic algorithm for optimum solution search of HTN planning, J. Inf. Comput. Sci. 10 (5) (2013) 1303–1313.
[156] T.A. Estlin, S.A. Chien, X. Wang, Hierarchical task network and operator-based planning: two complementary approaches to real-world planning, J. Exp. Theor. Artif. Intell. 13 (4) (2001) 379–395.
[157] V. Shivashankar, R. Alford, U. Kuter, D. Nau, Hierarchical goal networks and goal-driven autonomy: going where AI planning meets goal reasoning, in: Goal Reasoning: Papers from the ACS Workshop, 2013, pp. 95–110.
[158] M. Brenner, B. Nebel, Continual planning and acting in dynamic multiagent environments, Auton. Agents Multi-Agent Syst. 19 (3) (2009) 297–331.
[159] M. Ghallab, D. Nau, P. Traverso, The actor's view of automated planning and acting: a position paper, Artif. Intell. 208 (2014) 1–17.
[160] E. Kaldeli, Domain-independent planning for services in uncertain and dynamic environments, Ph.D. thesis, Faculty of Mathematics and Natural Sciences, University of Groningen, 2013.