Task Selection of OWL-S Services During HTN Planning

Evren Sirin and Bijan Parsia
Department of Computer Science,
University of Maryland, College Park, MD 20742, USA,
evren@cs.umd.edu, bparsia@isr.umd.edu

Abstract

Hierarchical Task Network (HTN) planning has been used to compose services described in the OWL-S framework. However, traditional HTN task matching based on method names and signature is too simple to match methods contributed by diverse parties working in a decentralized, uncoordinated manner, as we expect on the Semantic Web. To address this problem, we present an extension of HTN planning, HTN-DL, which replaces the traditional task matching function with description logic based matchmaking.

1 Introduction

In HTN planning, when the planner is searching for the possible decompositions of a given task, the methods are matched based on the name of the task and its functional signature, i.e. the number of the parameters and their types. This simple matching criteria will obviously fail in a distributed and decentralized environment as separate developers cannot be expected to use the same names for their Web Service descriptions. In order to achieve interoperability in a Web context, tasks should have more expressive descriptions of their functionality such that diverse developers working in various situations are likely to produce compatible (or at least mappable) descriptions. The obvious suggestion is that these descriptions should be formulated in an expressive knowledge representation (KR) language against commonly available ontologies, e.g., Web based OWL ontologies. The OWL-S ontologies are the natural foundation for these descriptions, since they already provide a reasonable basic modeling of the Web Services domain and support the largest set of existing service descriptions.

1.1 HTN-DL Tasks, Methods and Operators

The main idea behind HTN-DL is to separate the task descriptions from method and operator definitions. There are two advantages of the separation. First, tasks can be described in a completely different and more expressive language. Second, tasks can be completely abstract. There are two components of a HTN-DL domain. The first component domain contains the operator and method descriptions. The second component is a DL knowledge base that contains task descriptions and the information about which task an action accomplishes. HTN-DL operators and methods are defined in the first component as follows:

Definition 1 (HTN-DL Operator) *An HTN-DL operator is described as* o = (name, DL-pre, DL-effects) *similar to original HTN operators with the difference that the expressions (*DL-pre *and* DL-effects*) consist of (possibly unground) ABox atoms.*

Definition 2 (HTN-DL Method) *An HTN-DL method is a tuple* m = (name, DL-pre, network) *with no task information. Similar like HTN-DL operators* DL-pre *consists of (possibly unground) ABox atoms.* network *is a set of HTN-DL tasks and a partial ordering on the tasks.*

Tasks are described as class descriptions in the TBox component of a DL KB and action symbols (operator and method names) are described as individuals in the ABox component.

Definition 3 (HTN-DL Task) *An HTN-DL task is simply a DL class expression. There also exists two special-purpose properties, namely* $hasInput$ *and* $hasOutput$*, that can be used to describe the parameters of the task.*

For example, a task with no inputs would be described as $\leq_0 hasInput$ whereas a task with input type $Address$ is described as $\exists hasInput.Address$.

Definition 4 (HTN-DL Task Ontology) *A* task ontology T_{ont} *is a DL knowledge base. TBox component of* T_{ont} *contains HTN-DL task descriptions and their relations with each other (subclass hierarchy). The ABox component of* T_{ont} *contains one individual for each operator and method in the domain. The individual is identified with the* name *of the operator or method. If an operator* o *(or method* m*) accomplishes a task* T *then this is expressed with the ABox type assertion* T(name) *(or* T(name)*).*

1.2 HTN-DL Algorithm

A planning problem in HTN-DL is the tuple (s, T, D) where s is a DL KB that represents the current state, T is the initial task network, and D is an HTN-DL domain as defined before. The domain information includes the task ontology T_{ont} that gives the definitions and task hierarchy. s contains the information about the state of the world.

The HTN-DL planning procedure operates similarly to the HTN algorithm in spirit. The tasks in the initial network are

```
procedure HTN-DL(s, T, D)
    if T is empty then return empty plan
    Let t be a task in T with no predecessors
    Nondeterministically choose an individual a from
        the KB such that T_ont ⊨ t(a)
    if a is an operator name defined in D then
        Let o = (a, Pre, Effects) in D
        if s ⊭ Pre then return failure
        Let s' be s after applying Effects
        Let T' be T after removing t
        return [o, HTN-DL(s', T', D)]
    else if a is a method name defined in D then
        Let m = (a Pre network)
        if s ⊭ Pre then return failure
        Let T' be T after replacing t with network
        return HTN-DL(s, T', D)
    end if
end HTN-DL
```

Figure 1: HTN-DL planning procedure.

matched with operator and method definitions. When an operator matches the selected task, task is removed from the network and the operator is added to the plan (provided that its precondition is satisfied). When a method is matched with the task, task is replaced with the subtasks defined in the method.

The main difference of the HTN-DL algorithm is the task matching mechanism. In HTN-DL, tasks at hand are simply class descriptions in T_{ont} and actions (operators and methods) are individuals in T_{ont}. Task selection is reduced to the well-known *instance retrieval* algorithm in DL; given a class expression C find all the individuals $\{i_1, i_2, \ldots, i_n\}$ such that $\forall k, 1 \leq k \leq n$ it is true that $(i_k)^I \subseteq C^I$.

Note that, we do not have any assumptions about the expressivity of the DL being used with HTN-DL. Different DLs with varying complexity, $\mathcal{ALC}$, $\mathcal{SHF}$, or $\mathcal{SHIQ}$, can be used in HTN-DL. It is preferable to use a decidable DL for the task matching but unfortunately this would not ensure the decidability of the planning procedure as HTN planning is undecidable in general [1].

2 Implementation and Evaluation

As an initial attempt to investigate the applicability of our ideas, we have extended the JSHOP planner, the Java version of the HTN planner SHOP2 [2], with OWL DL reasoner Pellet. This new planner uses the DL reasoner to evaluate HTN-DL preconditions and matches the tasks with methods based on the descriptions in the task ontology.

We have created a planning domain about buying books. For the task ontology, we have augmented the OWL version of the North American Industry Classification System (NAICS) ontology with some additional definitions. NAICS ontology contains definitions about 1800 categories for classifying business establishments. The planning problem is to buy one or more books where there are different restrictions for each book, e.g. for a certain book we may be interested in only unused copies sold by a high rated service.

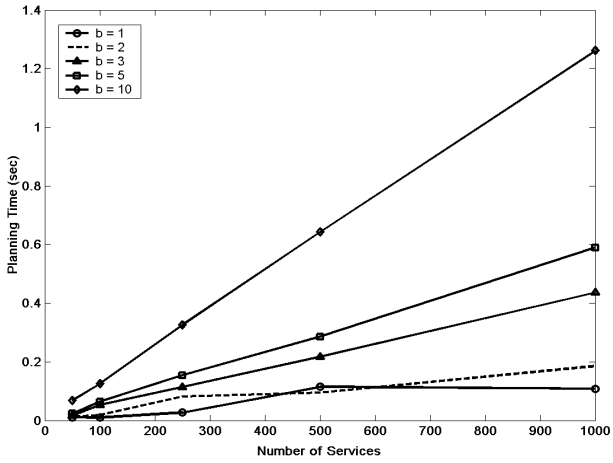

Figure 2: Performance of HTN-DL planner where each line shows the time spent when the planning problem

Note that this is a relatively simple problem from a planning perspective but has quite different characteristics than a usual planning problem. Classical planning benchmark problems, e.g. famous blocks world problem, has generally dealt with a small number of predefined actions, e.g. move block, in the presence of large number of objects, e.g. hundreds of blocks. However, interesting Web Composition problems generally deal with a large number of actions with varying properties, e.g. hundreds of book selling services, but with limited number of objects involved, e.g. buying a couple of books. Therefore, we believe this is a good starting example that shows the characteristics of a Web Services domain.

Figure 2 shows the planning time for solving different versions of this problem. We have randomly generated planning domains with 50, 100, 250, 500 and 1000 services and planning problems that involved buying 1, 2, 3, 5, and 10 books. For each setting, we used 10 different problems and reported the average planning time. Note that buying 10 books using 1000 services takes only 1.4sec demonstrating the feasibility of HTN-DL approach. Although reasoning with OWL DL ontologies have theoretically very high complexity (NEXPTIME), highly optimized instance retrieval algorithm used in task matching show a linear behavior even for the cases where we have 1000 instances (i.e. services) and 2000 concepts (i.e. categories) in the task ontology.

References

[1] Kutluhan Erol, James Hendler, and Dana S. Nau. Htn planning: complexity and expressivity. In *Proc. of the 12th National Conference on Artificial Intelligence*, 1994.

[2] D. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of AI Research*, 20:379–404, 2003.

Demo explanation

The demonstration is planned to show the implementation of HTN-DL algorithm and how it can be used to compose real-world Web Services. Demonstration will present the relationship between HTN-DL concepts, e.g. task ontology, method, operator, and Web Service concepts, e.g. service taxonomy, composite service, atomic service. We aim to show how reasonably expressive OWL-S descriptions can be build starting from pure syntactic WSDL descriptions and then HTN-DL planner can be used to generate executable compositions based on abstract composite descriptions.

Demo will be composed of several sections that show

- how WSDL descriptions are converted to OWL-S
- how these descriptions can be enriched using precondition/effect descriptions and service taxonomies
- how such descriptions are mappable to HTN-DL domains
- how HTN-DL planner will generate composition based on such domains

Demo will start with a brief review of WSDL, and the relation between WSDL and OWL-S, i.e. "grounding" ontology of OWL-S. We show our WSDL to OWL-S conversion tool and explain the limits of automatic conversion. We also explain how OWL-S profile hierarchies can be build to provide a capability oriented taxonomy of services. The automatically generated OWL-S descriptions are linked to these taxonomies to create a library of Web Services.

The second step in the demonstration is to use an interactive composition tool [3] which will allow people to explore matchmaking using a DL reasoner, namely OWL DL reasoner Pellet [4], in a service composition framework. This part shows how simple forward and backward chaining planners can be used to build up composite services (plans). We also discuss more complex composite services which cannot be easily found with current planners (but can serve as input to HTN planners).

The next step in the demonstration is to discuss the relation between OWL-S descriptions and HTN-DL concepts. Using examples, we show the mapping from OWL-S profile hierarchies to HTN-DL task ontologies, OWL-S composite processes to HTN-DL methods, and from OWL-S atomic processes to HTN-DL operators. We explain the why OWL-S language is limited and is not suitable to describe the HTN-DL tasks in their full generality. The extensions to OWL-S are described using the examples.

The last part of the demo is to show the HTN-DL system in action where the OWL-S templates are fed into the planner along with a task ontology and a library of available services. The planner dynamically matches the abstract process descriptions in the template with concrete service descriptions and finds an executable flow out of many different possibilities.

In the demo, there will be mainly two planning domains used. First is the book buying example as explained above and the second is the problem of making travel arrangements for a conference, e.g. finding the available transportation, booking accommodation and so on.

Software and tools that will be used in the demo are:

[1] OWL-S API, http://www.mindswap.org/2004/owl-s/api/

[2] HTN-DL, http://www.mindswap.org/2004/owl-s/api/download/HTN-DL-0.1.zip

[3] Web Service Composer, http://www.mindswap.org/evren/composer

[4] Pellet: OWL DL reasoner, http://www.mindswap.org/2003/pellet

[5] SWOOP: A Hypermedia-based OWL Ontology Browser and Editor, http://www.mindswap.org/2004/SWOOP

Additional resources:

[6] Sample OWL-S files, http://www.mindswap.org/2004/owl-s/services

[7] SHOP2 system, http://www.cs.umd.edu/projects/shop/