

Template-based Composition of Semantic Web Services

Evren Sirin and **Bijan Parsia** and **James Hendler**
{evren@cs.umd.edu, bparsia@isr.umd.edu, hendler@cs.umd.edu }
MINDSWAP Research Group,
University of Maryland,
College Park MD 20742, USA

Abstract

Workflow templates are necessary for various different Web Service related tasks such as encoding business rules in a B2B application, specifying domain knowledge in a scientific Grid application, and defining preferences for users that interact with Web Services. Abstract activities in templates can be used to specify the features of a required service and concrete services can be discovered and used to generate executable workflows. In this paper, we examine how Web ontologies can be used to write such template descriptions that will allow flexible matchmaking of services. We discuss the importance of expressing preferences in templates and provide a ranking algorithm based on DL inference services. We then present the HTN-DL formalism – an extension to the HTN planning formalism to generate compositions of Web Services based on these templates. Finally we present the experimental evaluation for the composition system we proposed.

Introduction

Web Services are an emerging paradigm in which very loosely coupled software components are published, located, and invoked on the Web as parts of distributed applications. The main focus of the Web Services effort is to achieve interoperability between heterogeneous, decentralized and distributed applications. Dynamic integration of the Web Services to fulfill the requirements of the task at hand is one of the most important objectives.

Using workflow templates for Web Service composition provides the means to achieve such functionality. A workflow template describes the outline of activities that need to be performed to solve a problem. Generally, templates are parameterized with respect to some variables so the generic template can be configured and customized for a specific instance of the problem based on the users' current requirements and preferences. Such a perspective on composition has been realized in different systems, such as HTN planning for Web Services (Sirin *et al.* 2004), Golog's "generic procedures" (McIlraith & Son 2002), and configuration based on parametric design (ten Teije, van Harmelen, & Wielinga 2004).

Workflow templates are useful in many domains and applications. For example, in B2B applications business rules can be encoded using templates. Often, most of the steps in a business process are fixed. However, partners that will be used in the transaction might be chosen at run-time depending on the specific request at hand. Similarly in Grid applications, workflows represent the standard procedures for accomplishing a task, but the exact services/resources that will be used for a specific instance of the task may vary.

There are various considerations about how a workflow template can be written for Web Services. The activities in the workflow may be bound to an existing concrete service at design-time or these activities may be defined in terms of abstract descriptions that will be matched with the available concrete services at run-time. On one hand, we want such abstract descriptions to be generic enough so all the relevant services can be found at execution time. On the other hand, we want some of the features to be fixed in all possible matches so the integration and the execution of the dynamically discovered service can be automatically accomplished without human intervention. Moreover, the abstract service descriptions should be able to specify the preferences of the template generator so the matching services can be ranked and selected based on these preferences at run time.

In this paper, we investigate how workflow templates can be written to describe expressive and flexible compositions. We are mainly motivated by the problem of *template instantiation*, i.e. finding the concrete services that constitute a valid execution of the original workflow. In doing so, we focus on workflows that are controlled by a single party, in other words we are looking at templates that are oriented toward Web Service orchestration not choreography. As a Web Service orchestration language we focus on OWL-S (OWL-S Coalition 2004) which is the most mature and probably the most widely deployed comprehensive Semantic Web Service technology.

In the remaining of the paper, we first highlight the most important features related to describing workflow templates, then discuss how OWL-S can be extended to define such templates. We then describe how algorithms for matching and ranking potential services can be used with these descriptions, and finally show how to combine all together with HTN planning formalism to generate compositions of the

Web Services.

Motivating Examples

Let us illustrate couple of real-world examples that will make our motivation and proposed solutions clearer. As a starting point, consider the problem of obtaining books on the Web. There are various online stores that sell new or used books, book rental clubs that let subscribers check out several books at a time, and libraries from where books can be borrowed. Such services will typically be published in a Web Service registry that uses a technology such as UDDI.

Web Service registries store information about Web Services, the businesses that provide the service and a taxonomy which is the categorization of available services. Figure 1 shows a portion of such a categorization for book selling services. For each category in the taxonomy, there might be several different Web Services registered. Each of these services could have different structures and different interaction models. Some services may involve a single message exchange: send the order request with credit card information and delivery address. Other services may have more complicated structures: create a new account (or login to an existing account), add the items to a shopping cart and proceed to checkout.

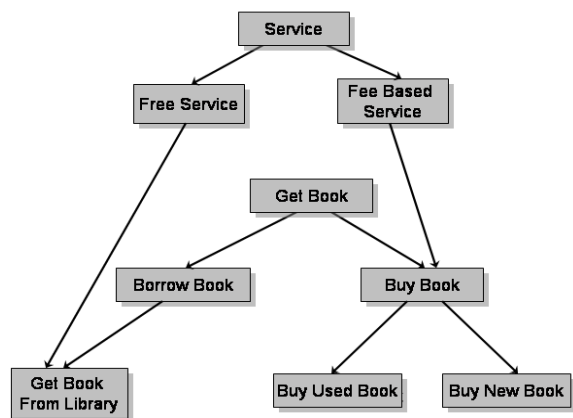


Figure 1: A simple ontology that describes book related services

In an extension to this standard setting, we can also publish and discover workflow templates. For example, a conference organizer can provide a workflow template that describes the steps for making travel arrangements for the conference. Such a template would describe steps to register to the conference, arrange transportation to the conference venue, make hotel reservations, and possibly register for additional events provided during the conference. However, the conference organizer may not want to specify the exact hotel or travel agent service in the template. Instead template just describes the abstract services and when someone wants to use this template, concrete services are selected based on the given user input.

Users might also have some pre-defined workflow templates customized to their needs, e.g. at the end of a transac-

tion the amount spent is converted to a certain currency and the expenses are recorded in the user's financial software. Such templates can be generated using a semi-automated composition tool (Sirin, Parsia, & Hendler 2004) or could be saved from a previous similar transaction or generated using workflow mining techniques (van der Aalst *et al.* 2003).

The goal of template-based composition is to find a sequence of services that will be a valid execution of the given workflow template. Note that this is a more generalized version of workflow instantiation problem because we are considering recursive decomposition of templates, e.g. an abstract step in a template can be achieved with another template that needs to be further decomposed, and the steps of the workflow cannot always be considered in isolation, e.g. for the conference example above we do not want to buy the plane tickets if we cannot reserve the hotel room. Therefore, it is required that executability of the whole plan should be considered possibly by looking at different choice points in the workflow, e.g. choose traveling by train if no plane tickets can be found.

Workflow Templates

A workflow specifies what activities need to be done, in what order (sometimes linearly, sometimes in parallel) to accomplish a certain task. The execution ordering of these activities are defined through different constructors, which permit the flow of execution control, e.g. sequence, loop and concurrency. A workflow not only specifies the control flow but also details the data flow, i.e. what are the parameters of each activity, how the data is carried between each activity and so on. Workflows in general can describe real-world activities that are carried out by humans but we focus on workflow of Web Services where each atomic concrete activity is defined as a Web Service.

A workflow template is a generalized workflow where some of the steps are defined as *abstract* activities. Of course there are various different ways to define an abstract service. The idea is not to fix each step of the workflow at design time and have the ability to discover and substitute services at run time. Such a template is more suitable in a dynamic environment where the available services and task requirements are constantly changing.

The most straight-forward way to define an abstract service is using the notion of an *interface*, as provided by WSDL. Similar to the interfaces in a programming language, a WSDL interface defines an operation and its functional signature but no execution detail. Any service that implements the interface is guaranteed to safely work at run time. However, this approach does not provide any flexibility, i.e. another service which provides the same functionality through a different interface can not be used.

Furthermore, in a template we need the ability to specify preferences so the possible matches found at execution time can be ranked and selected. Preferences could include choices and/or restrictions about the concrete service that can be used in place of the abstract process, e.g. use only services that are certified by some authority, do not use any service that does not adhere to the required security and privacy policies, try to find non-fee services and use a fee-based

service if there are no other choices available, etc. Some of these choices can be defined global, e.g. never use an uncertified service, and some others could be custom to the template element, e.g. for this step require additional encryption.

Templates in OWL-S

OWL-S (OWL-S Coalition 2004), a set of ontologies written in Web Ontology Language (OWL) (Dean & Schreiber 2004) for describing Web Services. The OWL-S ontology includes three primary sub-ontologies: the service profile, process model, and grounding. The profile is used to describe the service capabilities defined through inputs, outputs, preconditions and effects as well as qualitative properties such as the quality of service provided, security guarantees made, etc. Process model describes the pattern of interaction with the Web Service and grounding defines the execution details of the Web Service by linking the process definition to a WSDL operation.

OWL-S process ontology is a quite extensive orchestration language that provides control flow elements such as *Perform*, *Sequence*, *Any-Order*, *Choice*, *If-Then-Else*, *Iterate*, *Repeat-While*, and *Repeat-Until*. A *CompositeProcess* in OWL-S can be built from other processes using these constructs along with the data flow between the components. Processes that do not have any internal structure (or whose internals are hidden) are defined as *AtomicProcesses*.

However, OWL-S does not have a control construct to describe abstract processes in a flexible way. The *SimpleProcess* construct was originally envisaged for a similar purpose but in the latest release of OWL-S (version 1.1) this construct is quite under-specified. Technical overview states that *SimpleProcess* is primarily designed to provide an abstract view for an existing atomic or composite process. That is, it is mainly used when the concrete process that corresponds to the abstract view is known a priori at design time.

More importantly, an abstract process description in OWL-S should be able to refer to profile descriptions. A profile description include information about various different aspects, e.g. information about the service provider, the rankings and certifications associated with the service. This information is required to differentiate among candidate services for finding the best possible choice.

Extensions to OWL-S

Our solution to above problems is to have a new process type named *AbstractProcess* to define functional requirements such as input, output types and annotate the *Perform* control construct with preference descriptions. This separation allows us to reuse the same *AbstractProcess* with different preferences in different contexts.

AbstractProcess is similar to an *AtomicProcess* description and has inputs, outputs, preconditions and effects. The difference is that *AbstractProcesses* are not connected to any specific *Profile* or *Grounding* description. And unlike *SimpleProcess*, it does not have any links to existing processes. *AbstractProcess* can be used with *Perform* construct as an ordinary *Process*. This allows us to have *partially abstract*

templates, e.g. we can have a sequence where first step is a concrete *AtomicProcess* that is executable and second step is and *AbstractProcess* that needs to be instantiated. The following is an example¹ that describes an *AbstractProcess* that sells books and accepts credit cards Visa or Mastercard.

```
(define abstract process BuyBookProcess
  inputs: ( RequestedBook - books:Book,
            ShipmentAddress - loc:Address,
            Payment - :VisaOrMastercard )
  outputs:( Receipt - :PurchaseReceipt )
  result: (
    shippedTo(RequestedBook,
              ShipmentAddress) )
);

:VisaOrMastercard a owl:Class;
  owl:oneOf ( :Visa :Mastercard ) .
```

The preference descriptions are added to the perform description as additional statements. The base preferences can be combined through a partial ordering that defines the prioritization of preferences, i.e. specifying that one preference is more important than the other. We now describe our preference schema which is inspired by the Preference XPATH (Kießling *et al.* 2001), a query language designed to encode various selection conditions in e-commerce applications.

In its simplest form, a base preference says that the profile of the sought service should belong to a class in some service ontology. The class here is not restricted to be a named concept in the ontology. Ordinary OWL constructs for conjunction (*owl:intersectionOf*), disjunction (*owl:unionOf*), negation (*owl:complementOf*) can be used to describe more complex descriptions. The following example prefers all profiles that are either *NewBookBuyingService* or *UsedBookBuyingService* but not *BorrowBookService*.

```
:BuyBookPref a pref:PreferProfile ;
  pref:profile [
    a owl:Class;
    [ owl:intersectionOf (
      [ owl:unionOf (
        :NewBookBuyingService
        :UsedBookBuyingService ) ]
      [ owl:complementOf
        :BorrowBookService ] ) ] ] .
```

The preference can be defined for certain attributes of a profile. Such attributes in OWL-S are described using an extensible set of *ServiceParameters*. A *ServiceParameter* can also be defined to have values restricted to a class but we can also have numerical *ServiceParameters*. For example, average response time in milliseconds for a service can be defined as a parameter that takes integer values. Preferences about such numerical values can be to get the minimum or maximum values, i.e. prefer services with minimum average

¹For brevity, process descriptions are written in OWL-S presentation syntax and all other ontological definitions are given in N3 syntax. Both formats can easily be translated to verbose RDF/XML syntax.

response time or maximum reliability metric. These preferences are expressed as follows

```
:RespTimePref a pref:PreferMinimum ;
  pref:serviceParam :AvgResponseTime .

:ReliabilityPref a pref:PreferMaximum;
  pref:serviceParam :ServiceReliability .
```

These preferences can be combined through *cumulation*, which treats each preference equally important, or through *prioritization*, which treats some preferences more important than others. Both, cumulation and prioritization forms a partial order and these complex preferences can be applied orthogonally.

```
:QOSPref a pref:PreferAll ;
  pref:preference :RespTimePref ;
  pref:preference :ReliabilityPref .

:OverallPref a pref:PreferOrdered;
  pref:preference (
    :BuyBookPref
    :QOSPref ) .
```

Finally these preferences are attached to the *Perform* statement to specify the preferences for the given abstract process as follows:

```
:PerformBookBuy a p:Perform ;
  p:process :BuyBookProcess ;
  p:withPreference OverallPref .
```

Multiple preferences in the same *Perform* are simply treated as elements of a *PreferAll*, i.e. no prioritization is assumed. Preferences can be used only with *AbstractProcess* performs.

Matching and Ranking

Given a *Perform* statement with an *AbstractProcess* description and some preferences, we want to find the matching concrete services and then order the possible choices based on the preferences. We exploit standard reasoning services for both computing possible matches and deciding if a service satisfies given preferences. The subsumption relations defined in the ontologies allow us to generate more flexible matches.

We reduce the matching on functional attributes (inputs, outputs, preconditions and effects) to *query containment*. Given a process description we generate two queries; one for input requirements and one for output requirements. The expression *InQ* denotes the input requirement is the conjunction of atoms in the precondition expression with the input type restrictions. Similarly the output requirement expression *OutQ* is the conjunction of atoms in the effect expression with the output type restrictions². Consider the *BuyBookProcess* example we defined in the previous section. Then we would get the following expressions:

$$\begin{aligned} InQ &= Book(RequestedBook), \\ &Address(ShipmentAddress), \\ &VisaOrMastercard(Payment) \\ OutQ &= PurchaseReceipt(Receipt), \\ &shippedTo(RequestedBook, \\ &ShipmentAddress) \end{aligned}$$

A concrete process *C* matches with an abstract process *A* if $InQ(A) \sqsubseteq InQ(C)$ and $OutQ(C) \sqsubseteq OutQ(A)$ where \sqsubseteq is the query containment relation. Formally, a query *q1* is contained in *q2* if all the answers of *q1* are included in the answers of *q2*. This relation between the concrete and abstract process ensures two conditions 1) When the precondition of the abstract service is true then the precondition of concrete service is also true 2) After the effects of the concrete service is applied the effects of the abstract is also satisfied. It is easy to see that when there are no precondition or effect expressions associated with the process this matching algorithm reduces to standard techniques developed for Semantic Web Service matchmaking (Paolucci *et al.* 2002; Li & Horrocks 2003) that simply checks subsumption relation between parameter types.

The second step in matching is to compute the rank of the services based on the given preferences. We adopt the standard rank computation methods used in multi-attribute decision problems. The preferences on different attributes are treated equally important, i.e. the well-known Pareto optimality principle that avoids the “empty-result”-effect and the flooding-effect. The prioritization of preferences are applied orthogonally to rank the possible matches.

HTN-DL: Extended HTN Formalism

Hierarchical Task Network (HTN) planning for Web Service composition (Sirin *et al.* 2004) is an example of template-based composition approach. An HTN planning problem is formulated in terms of tasks to be accomplished where a task represents an abstract activity. A task is defined with its name and the number of parameters it has (no types associated with parameters). Tasks at hand are matched against *method* descriptions, which is a prescription for how to decompose a task into *subtasks*. There are also *operators*, which are single-step atomic actions with no internal structure. The tasks that are accomplished by operators are named *primitive tasks*.

An HTN planning problem starts with one or more tasks, methods are matched with the tasks and recursively decomposed until each step is reduced to an operator (For a more detailed description of HTN planning reader is referred to (Ghallab, Nau, & Traverso 2004)). The resulting plan is a sequence of operators. The HTN view of planning fits naturally with composition of OWL-S services as composite OWL-S processes can be directly translated to HTN method descriptions and atomic processes can be modeled as operators. As shown in (Sirin *et al.* 2004) this process can be automated when the Web Services are described in OWL-S.

However, it is not possible to use HTN planning directly with the kinds of abstract process descriptions explained above. The notion of a task in HTN planning aligns with the *AbstractProcess* concept but HTN planning makes some assumptions that are too restricted.

²Here we do not consider conditional effects so we only have conjunctions of atoms for outputs

- An HTN task is a very specific type of an abstract activity, where the only information associated with it is its name and the number of parameters. There is no way to associate preferences about possible decompositions. For example, consider an abstract activity whose preference specifies the category *BuyNewBook* from the Figure 1. The service hierarchy implies that any service that belongs to categories *BuyBook* or *GetBook* would also be acceptable. More expressive HTN task descriptions are needed to represent preferences related to such taxonomies. Moreover, different services may use different kind of vocabularies (ontologies) to describe their functionality. It should also be possible to easily establish or infer the similarities between these descriptions.
- HTN planning makes a strict separation between *primitive* and *nonprimitive* tasks which makes it impossible to say that both a composite service and an atomic service achieves the same task. In other words, a task cannot be established both by a method or by an operator which means that task is both primitive and nonprimitive (consider the example of a single step book buying service vs. multi-step book buying service).
- Operator and primitive task symbols are unique so there can be at most one operator that accomplishes a primitive task. Clearly, this assumption is too restricted for Web Services because there are many different atomic Web Services that would accomplish the same task. The task symbols and operators names should be separate entities in order to overcome this limitation.
- The precondition and effects of the OWL-S are expressed in some dialect of OWL which is a fairly expressive KR language that makes the open world assumption. Such an expressivity cannot be directly handled with the existing HTN planners which at most allows axiomatic inference.

We now describe HTN-DL, that extends HTN formalism to overcome these problems.

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 with no structure (no primitive or nonprimitive task notion). Note that there are still primitive and nonprimitive actions, i.e. operators and methods.

There are two components of a HTN-DL domain. The first component describes the planning domain and contains the operator and method descriptions. The second component is a Description Logic (DL) knowledge base that contains task and preference descriptions. These components are called HTN component and DL component, respectively.

We chose the Description Logic as our knowledge representation language due to the direct correspondence between DLs and OWL. OWL can be viewed as an expressive Description Logic, where an OWL ontology is equivalent to a Description Logic knowledge base. A DL knowledge

base typically comprises two components: a “TBox” and an “ABox”. The TBox contains intensional knowledge in the form of a terminology and the ABox contains extensional knowledge that is specific to the individuals of the domain of discourse. OWL facts (type assertions, property assertions, individual equality and inequality) corresponds to ABox assertions and OWL axioms (subclass axioms, subproperty axioms, domain and range restrictions, etc.) correspond to TBox knowledge.

HTN-DL operators and methods are defined in the HTN component as follows:

Definition 1 (HTN-DL Operator) *An HTN-DL operator is described as $o = (\text{name}, \text{DL-pre}, \text{DL-effects}, \text{DL-profile})$ similar to original HTN operators. name is a syntactic expression $n(x_1, \dots, x_n)$ where n is the operator name and x_1, \dots, x_n are input parameters. DL-pre and DL-effects are expressions that consist of (possibly unground) ABox atoms describing the preconditions and the effects of the action, respectively. DL-profile refers to an instance in the DL component that describes the non-functional aspects of the operator (similar to an OWL-S profile).*

Definition 2 (HTN-DL Method) *An HTN-DL method is a tuple $m = (\text{name}, \text{DL-pre}, \text{network}, \text{DL-profile})$ where name, DL-pre, and DL-profile is defined as before. network is a set of HTN-DL tasks and a partial ordering on the tasks that describes how the method is decomposed into smaller steps.*

Note that, unlike OWL-S processes, HTN-DL operators and methods do not have types associated with the parameters. However, typed variables are merely syntactic sugar as they can be expressed in the preconditions by adding atoms (such conditions are also referred as knowledge preconditions).

Tasks descriptions are divided into two parts. Functional description of the task (parameters, preconditions, effects) are described in the HTN component whereas the preference description (base preferences and ordering relations) are described in the DL component.

Definition 3 (HTN-DL Task) *An HTN-DL task is described as $t = (\text{name}, \text{DL-pre}, \text{DL-effects}, \text{DL-preferences})$ where name, DL-pre, DL-effects are defined as before and DL-preferences refers to a preference description in the DL component.*

Definition 4 (HTN-DL Task Ontology) *A task ontology T_{ont} is a DL knowledge base where the profiles of actions and preferences related to tasks are defined. DL knowledge typically contains the profile hierarchy (subclass relations between service categories). Preferences are described with respect to DL-profiles of the actions.*

Note that, the elements of HTN-DL has a very close relationship to the extended version of OWL-S described in the previous sections. The main advantage of representing the services in HTN-DL (compared to just using HTN) is the ability to refer to the profiles of the services in addition to the process descriptions. The preference descriptions about such profiles are stored in the DL component which means

we can directly use the OWL definitions without a translation.

Finally a planning domain in HTN-DL is defined as follows:

Definition 5 (HTN-DL domain) An HTN-DL domain D is a triple (O, M, T_{ont}) where O is a list of operators, M is a list of methods, and T_{ont} is the task ontology.

HTN-DL Algorithm

We now briefly describe the HTN-DL formalism, an extension to HTN planning where task descriptions and task matching is done using OWL-DL ontologies. The idea is to keep the previous translation algorithm to encode OWL-S processes as HTN methods but leave the matching of abstract activities to the OWL-DL reasoner. The DL reasoner also handles the evaluation of preconditions and simulates the effects of the services.

```

procedure HTN-DL( $s, T, D$ )
  if  $T$  is empty then return empty plan
  Let  $t$  be a task in  $T$  with no predecessors
  Let  $a = \text{match-and-rank}(t, D)$ 
  if  $a$  is an operator then
    Let  $o = (a, Pre, Effects)$  in  $D$ 
    if  $s \not\models Pre$  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 then
    Let  $m = (a, Pre, network)$ 
    if  $s \not\models Pre$  then return failure
    Let  $T'$  be  $T$  after replacing  $t$  with  $network$ 
    return HTN-DL( $s, T', D$ )
  end if
end HTN-DL

```

Figure 2: HTN-DL planning procedure.

The HTN-DL planning procedure (see Figure 2) operates similarly to the HTN algorithm in spirit. The main difference of the HTN-DL algorithm is the task matching mechanism. The function *match-and-rank* nondeterministically returns the matching actions for the given task in the order that is specified by the preferences. If a plan cannot be generated by the selected action, planning algorithm will backtrack and select the next available actions.

Implementation and Evaluation

As an initial attempt to investigate the applicability of our ideas, we have implemented the HTN-DL algorithm by extending the JSHOP planner, the Java version of the HTN planner SHOP2 (Nau *et al.* 2003), with OWL DL reasoner Pellet (Pellet 2003). We wanted to investigate two different points in our experiments: 1) How does the integrated system compare to the original planning system 2) How do the new task matching/ranking mechanism scale to a large

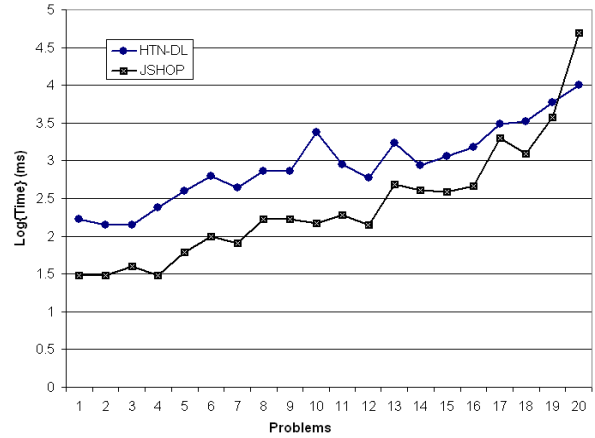


Figure 3: Comparison of HTN-DL with JSHOP system

number of services and complex ontologies. We ran our experiments on a Windows machine with a Pentium Centrino 1.6 GHz CPU and 512MB memory.

The objective of the first experiment is to see the overhead of using a DL reasoner for evaluating preconditions and effects of the services compared to the original planning system where the state of the world is represented simply as a relational database. In order to have a planning problem that would run on both systems, we took a standard planning problem (Rover domain which was used in the 2002 International Planning Competition (Fox & Long 2002)) and encoded this domain in HTN-DL similar to described in (Sirin & Parsia 2004). There were 20 different problems of increasing complexity and we ran both systems 20 times on each problem and computed the average time spent.

Figure 3 shows (in logarithmic scale) the total planning time spent by both systems for the different problems in the suite. As seen in the figure, the total planning time for HTN-DL is generally slightly greater than JSHOP. However, for the hardest problem, where there are large number of resources, HTN-DL performance is better than JSHOP. This is due to the fact that DL reasoner Pellet used in HTN-DL is optimized to handle large number of instances whereas the JSHOP implementation is not. This experiment shows that even though the expressivity of the knowledge representation language increases dramatically the reasoning time does not necessarily increase.

To test the performance of task matching/ranking in the presence of large ontologies, we have created a planning domain about obtaining books using Web Services. As mentioned in the “Motivating Example” section, we created different versions of these services that require different conditions, e.g. an online store may require registration before you place an order, a university library on the other hand will lend books only to its students and faculty. The planner needs to find a service for each book in the request list, verify the availability, ensure all the conditions of the service are met.

For describing abstract processes using service taxonomies 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. We have used the categories such as “Book stores” (NAICS code 451211), “Used Merchandise Stores” (NAICS code 453310), “Electronic Shopping” (NAICS code 454111) and specialized these classes for book buying services.

Workflow templates describe how to interact with the available services, e.g. some Web Services just provide functionality to order books but not search the content of the store in which case a separate service may be needed to get the ISBN number of the book to complete the order. 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.

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. For this reason, we believe this setting is a good starting example that shows the characteristics of a Web Services domain.

Figure 4 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 (NEXP-TIME), 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.

These are still preliminary results and we are now working on to build more domains from different application areas where we can evaluate the effectiveness and efficiency of workflow templates.

Related Work

The most relevant work to our is METEOR-S Web Service Composition Framework (MWSCF) (Sivashanmugam *et al.* Winter 2004 05). MWSCF describes Semantic Process Templates (SPT) that describe a workflow of abstract and concrete services. The templates may include QoS criteria that will be used for discovery. For ranking, discov-

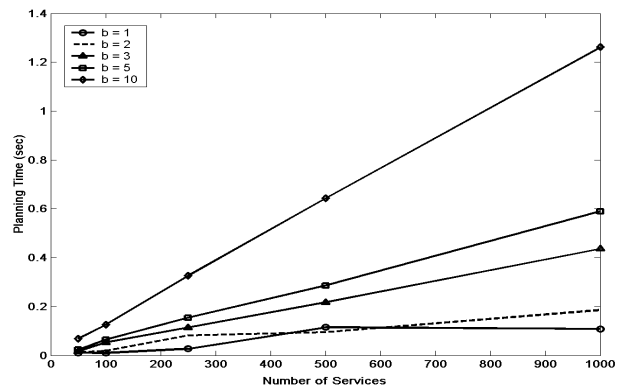


Figure 4: Performance of HTN-DL planner where each line shows the time spent with a different size of input task network

ered services each selection criteria is assigned a numerical score and a weighted combination of these scores are computed to rank the services. The main problem we see with assigning weights is to come up with good numerical values to encode the preferences. Especially when templates are generated by different developers, organizations and are being shared it is hard to ensure that these weights are normalized. Our choice of using qualitative preference specification by means of partially ordered preferences has been effective in e-commerce applications and more intuitive for users (Kießling *et al.* 2001). The other difference of our approach compared to MWSCF is the way we make use of non-deterministic choice constructs to encode possible different execution paths and consider recursive decomposition of templates. MSWCF on the other hand focuses on matching abstract services with atomic concrete services

Another approach for Web Service Composition using OWL-S, is described in (McIlraith & Son 2002). This work is based on the notion of generic procedures that are customized based on user constraints. This work extends the Golog language to enable programs that are generic, customizable and usable in the context of the Web. The Golog procedures can be seen as workflows as they can encode various different control flow elements. However, there is no notion of abstract services in this setting and only workflows of concrete actions and no flexibility to discover and use different services. Their approach provides the flexibility to change the conditions under which a service can be used (*desirability* conditions are defined in addition to preconditions). This lets users to customize the workflows to some extent but is useful for describing hard constraints rather soft constraints. Thus it is hard to write such constraints because it is easy to get into situations where either no plan is found or too many possible plans are generated.

AI planning techniques other than HTN planning have been applied to Web Service composition. These approaches range from simple classical STRIPS-style planning (Sheshagiri, desJardins, & Finin 2003) to extended estimated-regression planning (McDermott 2002), and from “Planning as Model Checking” (McDermott 2002) that deals with non-

determinism, partial observability, and complex goals, to planning based on “knowledge-level formulation” (Martinez & Lespérance 2004). The main difference of HTN planning is the use of domain knowledge, e.g. template structures, to guide the planner to find a composition. Other planning approaches mentioned above focuses on atomic services in isolation and tries to generate compositions using only the precondition and effect descriptions. Although this is useful for the cases where there are no known templates at hand for a task, in general it is hard for these approaches to scale to problems where there are large number of services.

Conclusions

Workflows play an important role in many different application areas such as B2B enterprise applications, scientific Grid computing applications, and even for users that interact with Web Services. The evolving requirements in these applications and the constantly changing availability of services makes it hard to build fixed workflows. Instead flexible workflow templates that describe the abstract functionalities are needed so concrete services to satisfy the requirements can be selected at run time rather than fixing the choice at design time.

In this paper we have looked at the problem of describing such workflow templates. Our approach is to use ontologies to describe abstract functionalities and encode qualitative preferences in workflow descriptions. This mixture allows us to encode both hard constraints about the functional parameters of the services and soft constraints related to non-functional attributes of the services. We explained how to extend the OWL-S language to support such abstract process definitions.

We also presented the HTN-DL formalism, an extension to HTN planning that is aimed to solve Web Service composition problems by using these expressive template descriptions. We have implemented a preliminary system that combines HTN planning with DL reasoning and our initial experimental results demonstrate the feasibility of our approach.

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA), Fujitsu Laboratory of America at College Park, Lockheed Martin Advanced Technology Laboratories, the National Science Foundation (NSF), the National Institute of Standards and Technology (NIST), and NTT Corp.

References

- Dean, M., and Schreiber, G. 2004. OWL Web Ontology Language Reference W3C Recommendation. <http://www.w3.org/tr/owl-ref/>.
- Fox, M., and Long, D. 2002. International planning competition. <http://www.dur.ac.uk/d.p.long/competition.html>.
- Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann. chapter Hierarchical Task Network Planning.
- Kießling, W.; Hafenrichter, B.; Fischer, S.; and Holland, S. 2001. Preference XPATH—a query language for e-commerce. In *Proceedings of the 5th International Conference Wirtschaftsinformatik*.
- Li, L., and Horrocks, I. 2003. A Software Framework For Matchmaking Based on Semantic Web Technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*.
- Martinez, E., and Lespérance, Y. 2004. Web service composition as a planning task: Experiments using knowledge-based planning. In *Proceedings of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services*, 62–69.
- McDermott, D. 2002. Estimated-regression planning for interactions with web services. In *AIPS*, 204–211.
- McIlraith, S., and Son, T. 2002. Adapting Golog for composition of semantic web services. In *Proc. of the 8th International Conference on Knowledge Representation and Reasoning*.
- Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of AI Research* 20:379–404.
- OWL-S Coalition. 2004. OWL-S: Semantic markup for web services. W3C Member Submission 22 November 2004 <http://www.w3.org/Submission/OWL-S/>.
- Paolucci, M.; Kawamura, T.; Payne, T. R.; and Sycara, K. 2002. Semantic Matching of Web Services Capabilities. In *The First International Semantic Web Conference*.
- Pellet. 2003. Pellet - OWL DL Reasoner. <http://www.mindswap.org/2003/pellet>.
- Sheshagiri, M.; desJardins, M.; and Finin, T. 2003. A planner for composing services described in daml-s. In *AA-MAS'03 Workshop on Web Services and Agent-based Engineering*.
- Sirin, E., and Parsia, B. 2004. Planning for semantic web services. In *Semantic Web Services Workshop at 3rd International Semantic Web Conference (ISWC2004)*.
- Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN planning for web service composition using SHOP2. *Journal of Web Semantics* 1(4):377–396.
- Sirin, E.; Parsia, B.; and Hendler, J. 2004. Composition-driven filtering and selection of semantic web services. *IEEE Intelligent Systems* 19(4):42–49.
- Sivashanmugam, K.; Miller, J. A.; Sheth, A. P.; and Verma, K. Winter 2004-05. Framework for semantic web process composition. *International Journal of Electronic Commerce* 9(2):71.
- ten Teije, A.; van Harmelen, F.; and Wielinga, B. 2004. Configuration of web services as parametric design. In *Proc. of the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW'04)*.
- van der Aalst, W. M. P.; van Dongen, B. F.; Herbst, J.; Maruster, L.; Schimm, G.; and Weijters, A. J. M. M. 2003. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.* 47(2):237–267.