International Journal of Web Services Research

April-June 2011, Vol. 8, No. 2

Table of Contents

Editorial Preface

Research Articles

Using Markov Decision Process Model with Logic Scoring of Preference Model to Optimize HTN Web Services Composition

Jiuyun Xu, China University of Petroleum and Beijing University of Posts and Telecommunications, China

Kun Chen, China University of Petroleum, China

Stephan Reiff-Marganiec, University of Leicester, UK

ABSTRACT

Automatic Web services composition can be achieved using AI planning techniques. HTN planning has been adopted to handle the OWL-S Web service composition problem. However, existing composition methods based on HTN planning have not considered the choice of decompositions available to a problem, which can lead to a variety of valid solutions. In this paper, the authors propose a model of combining a Markov decision process model and HTN planning to address Web services composition. In the model, HTN planning is enhanced to decompose a task in multiple ways and find more than one plan, taking into account both functional and non-functional properties. Furthermore, an evaluation method to choose the optimal plan and experimental results illustrate that the proposed approach works effectively. The paper extends previous work by refining a number of aspects of the approach and applying it to a realistic case study.

Keywords: *HTN Planning, Logic Scoring of Preference, Markov Decision Process, Services Composition, Services Selection*

INTRODUCTION

Web services are software "components" at an abstraction level suitable for business level reuse which are combined into larger systems, often dynamically and only when need arises. This is possible as their functionality is described in a way that can be machine interpreted (e.g., through a WSDL file or a Semantic Web description). These combinations, which are usually referred to as compositions, allow for services that are part of different applications, reside on different platforms, are developed using diverse programming languages and are possibly owned by different business partners to cooperate smoothly. The need for composition arises as usually a single service cannot fulfill the requirements of a user. Web service

DOI: 10.4018/jwsr.2011040103

composition provides a mechanism to combine different services to handle a complex business process. Automated Web service composition allows to combine services without human involvement in the planning and is valuable in many domains, e-commerce is a typical example. However, with the rapid increase of Web services, increasingly complex requirement of business process in the real world automatic service composition requires a flexible mechanism to deal with changing service availability. AI planning has often been adopted for automated Web services composition, as exemplified by the methods presented in for example Sirin, Wu, Hendler, and Nau (2004), Schuschel and Weske (2004), Paik, Maruyama, and Huhns (2006), and Akkiraju, Srivastava, Ivan, Goodwin, and Syeda-Mahmood (2006) to handle this issue.

In Sirin et al. (2004), an HTN planning method has been suggested to handle automatic Web services composition. This method translates OWL-S Web service descriptions to a SHOP2 domain and then a given business plan is achieved by decomposing complex tasks using operators from the SHOP2 domain. Considering the procedure of task decomposition, this method mainly is concerned with the *feasibility* of task decomposition; that is can *one* plan be found? However, a plan may fail for various reasons, service instances may no longer exists when they are about to be invoked, feature interaction in Web services (Weiss, Esfandiari, & Luo, 2007) may lead to undesired behavior or the specific plan might simply not be the best available for a user. There usually are several possible plans which can solve one specific high-level business process, so there is a natural redundancy to avoid these problems, one only needs to go a step further than just finding one plan. For example, if a user wants to attend an exhibition in another city in a few days. On the condition of satisfying user's requirements, he can make a choice of taking a flight or a train to the city and then attend the exhibition. In this situation, the user always wants to know what options he has and which is of the best quality (that is satisfying his non-functional criteria such as cost considerations or time saving).

This paper addresses the aspect of finding multiple composition plans and then selecting the most appropriate for a user. We propose an enhanced approach for Web services composition based on the combination of HTN planning and a Markov decision process model. With this approach, several highly suitable Web service plans will be obtained providing different solutions to a business process using Web services composition and hence offering much more flexible solution to the customer. To make sure these plans are indeed some of the best solutions available we use an evaluation mechanism to illustrate the optimal solution amongst those multiple solutions using a Markovian decision process. In this way, the optimal solution not only meets the requirements of the business process in its functional aspects, but also satisfied the expectations that the solution is of the best quality based on requirements considering the non-functional aspects.

This paper is an extension of our work presented at ICWS 2009 (Chen, Xu, & Reiff-Marganiec, 2009). The paper extends the previous work by addressing a number of issues queried at the conference as well as on some aspects which were planned as future work. Specifically we present a more (1) realistic case study with a more complete (2) analysis of the approach in terms of its complexity. We have also enhanced the (3) method to evaluate the non-functional (or QoS) properties and addressed the issue of choosing appropriate (4) values for the threshold used in the control strategy.

The rest of the paper is organized as follows: in the next section, an overview of Web services composition using HTN planning is provided. The following sections describe Markov Decision Process with Logic Scoring for Preference model for HTN Web services composition and detail the process of model solving. A case study is introduced and experi-

mental results are presented. Finally, we discuss related work, conclude and provide an outline of further research.

AN OVERVIEW OF WEB SERVICES COMPOSITION USING HTN PLANNING

HTN (Hierarchical Task Network) is a technique of AI planning based on control knowledge with a closed world assumption (informally, that means that all "building blocks" are known a-priori). HTN planning provides hierarchical abstraction with a powerful strategy to deal with the complexity of large and complicated real world planning domains. The purpose of an HTN planner is to produce a sequence of actions that perform some activity or task.

As with any planning approach, there is a need to express the terms of the application domain in the language understood for the planner. In order to adopt HTN planning as an approach to web service composition the planning domain, planning problem and the process of planning need to be married to the relevant concepts of the web service domain. The description of a planning domain includes a set of operators (which will be web service operations), and also a set of methods, each of which is a prescription for how to decompose a task into its subtasks (smaller tasks). The description of a planning problem will contain an initial state which in classical planning is a goal formula but here it will be the problem specification. The problem specification is expressed as a partially ordered set of tasks to accomplish.

The process of HTN planning proceeds by using the methods to decompose tasks recursively into smaller and smaller subtasks, until the planner reaches primitive tasks that can be performed directly using the planning operators. For each non-primitive task, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates methods to decompose the subtasks even further. When the constraints on the subtasks or the interactions among them prevent the plan from being feasible, the planning system will backtrack and try alternative methods. More details on HTN planning can be found in Nau, Au, Ilghami, Kuter, Murdock, Wu, and Yaman (2003).

OWL-S (World Wide Web Consortium, 2004) is a set of ontologies for describing the properties and capabilities of Web services. Currently, OWL-S is used to describe web services since it supports effective automation of various web services related activities including service discovery, composition, execution, and monitoring (it provides a richer framework than WSDL). Especially, the structure of OWL-S is propitious to exploit AI planning techniques for automatic service composition by treating service composition as a planning problem. In OWL-S, services can be described as composite or atomic processes with preconditions and effects. The concept of composite process decomposition in OWL-S process ontology is very similar to the concept of task decomposition in HTN planning. Hierarchical modelling is the core of the OWL-S process model to the point where the OWL-S process model constructs can be directly mapped to HTN methods and operators. Thus, HTN planning is especially promising for OWL-S Web services composition, which has been shown in Sirin et al. (2004), Kuter, Sirin, Parsia, Nau, and Hendler (2005), and Lin, Kuter, and Hendler (2007).

FORMAL MDP-LSP MODEL FOR HTN WEB SERVICES COMPOSITION

Current automatic web services composition techniques including those based on HTN planning are usually concerned with finding one solution that functionally fulfills the demands. However, with many functionally equivalent services being available work in other areas has considered selecting the best service for a given situation based on non-functional properties. Furthermore, having a choice also means that there is normally more than one possible

solution to address given requirements. Our approach addresses these, but requires extensions to the composition model used by SHOP2 in Sirin et al. (2004). Our approach of combining HTN with MDP-LSP (Markov Decision Process Model with Logic Scoring of Preference) identifies a choice of Web services composition plans and considers the non-functional aspects of Web services, which enhances the flexibility of automatic Web services composition. The approach is supported by a formal model as follows.

Definition 1 (MDP-LSP Model for HTN Web Services Composition): An OWL-S Web services composition problem is defined by a quintuple <*S, T, D, Q, P*>, where

- *S* is the initial state of the problem.
- *T* is the task list, which contains the tasks that the user needs to solve.
- *D* is the description of a planning domain including a set of operators and a set of decomposition methods, and D can be derived from a collection of OWL-S process models.
- *Q* is the context information about services quality, which covers a wide range of non-functional properties.
- *P* is a set of optimal solutions which are available in the solution space.

On the basis of the above definition, solving the 5-tuple can return an optimal plan $P_{optimal} = (o_1 o_2 \ldots o_n)$, that is a sequence of instantiated operators that will achieve *T* from *S* in *D*, with the best quality with respect to the non-functional aspects *Q*.

The model solving consists of three main steps. First, the initialization of the description of the planning domain; second, the search for the best plans in the solution space based on HTN planning and thirdly, the evaluation of the optimality of the found plans. The next section will describe these in detail.

THE APPROACH FOR SOLVING THE MDP-LSP MODEL

This section describes the approach of finding optimal plans, which in technical terms can be seen as finding solutions to the models presented in the previous section. As this section is quite lengthy due to describing all 5 major steps in the process, we want to present an overview first before going into the details. The first step is to initialize the planning domain and is based on work by Sirin et al. (2004). Essentially the knowledge about available operators and methods is extracted from the OWL-S service descriptions to create a model of the planning domain. While this step does need to be repeated frequently to obtain the latest available service methods, it does not have to be executed before each planning process, or even as part of each planning process. The next three steps are concerned with finding several plans, calculating the quality of a branch (its immediate reward) and controlling how much of the state space should be explored. The respective sections describe the approach for planning with multi-decomposition, the method for evaluating non-functional properties and the control strategy. Finally we turn our attention to finding the best plan.

Initialization of the Description of the Planning Domain

As stated earlier, we will be using HTN planning which has proven successful in earlier work (Sirin et al., 2004). The first step in the approach is to translate the description of OWL-S services to a description of the planning domain.

There are many planning domain description languages, examples are PDDL (Gerevini & Long, 2005) and SHOP2 (Nau, Muñoz-Avila, Cao, Lotem, & Mitchell, 2001) descriptions. PDDL is used by many classical planers as standard description language, but for HTN SHOP2 is more suitable because the control

knowledge base for HTN planning consists of operators and methods, which are naturally expressed in the SHOP2 domain description (Nau et al., 2003; Sirin et al., 2004).

Recall that operators are basic building blocks out of which the final plan would be build, while methods describe possible decompositions of larger tasks. The definitions of operators and methods are equivalent to what was defined in Sirin et al. (2004) and are as follows:

Definition 2 (Operator). An operator is an expression of the form ($h(v\rightarrow)$ *Pre Del Add*) where

- $h(v\rightarrow)$ is a primitive task with a list of input parameters $v\rightarrow$,
- *Pre* represents the operator's preconditions,
- *Del* represents the operator's delete list which is described as a conjunction of logical atoms that will become false after operator's execution, and
- *Add* represents the operator's add list which is described as a conjunction of logical atoms that will become true after operator's execution.

Definition 3 (Method). A method is an expression of the form ($h(v\rightarrow)$ Pre_1 T_1 Pre_2 T_2 ...) where

- $h(v\rightarrow)$ is a compound task with a list of input parameters ($v\rightarrow$),
- each Pre_i is a precondition expression, and
- each T_i is a partially ordered set of subtasks.

Before using HTN planning to compose Web services, we need to translate all OWL-S Web services descriptions into SHOP2 descriptions. In order to achieve this, the profile description of each service is translated to an element in the task ontology[1] and the service process model is translated to a set of methods and operators.

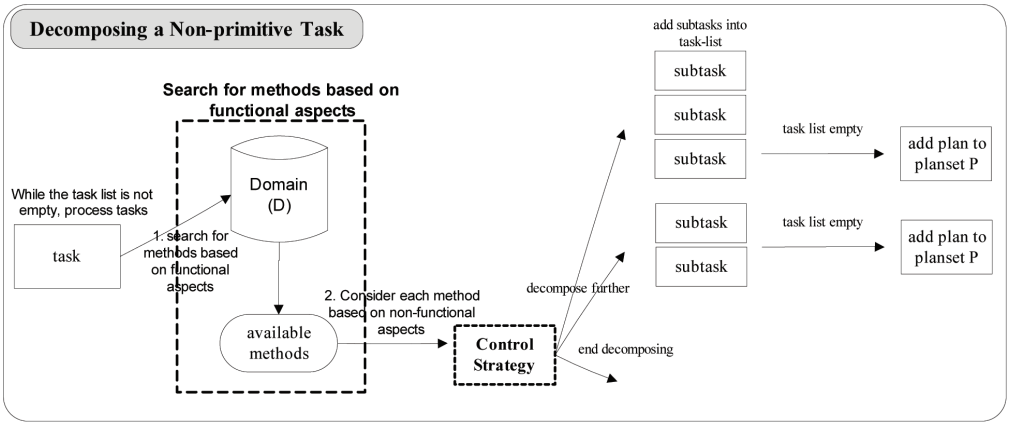
In the OWL-S process ontology, operations are modeled as processes, which can have two sorts of non-exclusive effects. First, an operation can generate and return some new information based on information it is given and the world state; Information production is described by the inputs and outputs of the process. Second, it can produce a change in the world; this transition is described by the preconditions and effects of the process.

There are three types of processes in OWL-S, including *atomic* processes, *composite* processes and *simple* processes. An *atomic* process is a model of a single-step Web service that can be executed to accomplish some task directly. A *composite* process is a compound Web service which can be decomposed into other *atomic* processes, *composite* processes or *simple* processes. The decomposition of a *composite* process is specified through its control constructs. A *simple* process is not invocable and not associated with a service grounding -- simple processes are used as elements of abstraction. They may be used either to provide a view of (a specialized way of using) some atomic process, or a simplified representation of some composite process (World Wide Web Consortium, 2004). The following will give an introduction to the translation algorithm as introduced in Sirin et al. (2004).

Let $K = \{K_1, K_2, \ldots, K_m\}$ be a collection of OWL-S process models. Then, we define the description of the planning domain *D* to be the results of the *TRANSLATE_PROCESS_MODEL(K)* operation defined by Sirin et al. (2004). Details of the translation and assumptions the translation based on are all kept unchanged and we will not describe the translation algorithm in detail here. Briefly, the process translates *atomic* processes into operators and translates *composite* processes or *simple* processes into domain methods respectively. Especially, the *composite* processes are translated according to their control constructs like *Sequence*, *If-Then-Else*, *Repeat-While* and so on. Each control construct corresponds to a sub-translating algorithm.

After the completion of this process, the element *D* needed for our model is complete and provides us with the needed set of operators and decomposition methods. Each operator is

Figure 1. Decomposing a non-primitive task



a description of what needs to be done to accomplish some primitive task, and each method tells us how to decompose some compound task into a set of partially ordered subtasks.

Clearly this initialization phase does not have to be executed every time a solution is sought – in general we can assume that the set of available services changes much less frequently then there is a need to find a new plan for a specific problem. Note that the initialization phase does need to be executed regularly to ensure that any change in the OWL-S process models and also new service arrivals are reflected in the domain description.

Planning with Multi-Decomposition for Tasks

In this paper, the process of HTN planning is improved in the second step, which is searching for plans. Initial ideas for this have been presented in Chen et al. (2009). This enhancement means that we are able to produce more than one good solution within the available solution space. Specific details of the improvement focus on decomposition for non-primitive tasks when a task can be decomposed by more than one method.

The improved decomposition method changes the way of decomposing when a task can be decomposed by multiple methods. The method chooses each method to decompose a non-primitive task instead of choosing any one of the ones applicable in the current state. Also, a control strategy is embedded into the planning process to decide whether a branch will be decomposed further. Figure 1 presents an overview of the improved non-primitive tasks decomposition.

The improved decomposition is superior to the decomposition presented in Sirin et al. (2004) on the strategy of searching for solutions as it does not just find one solution but finds many already pre-filtering for quality with a view to allowing to finally choose the best solution. For decomposing a non-primitive task with every available method, the current state (S) and task list (T) must be copied, and the number of the replications is the same as the number of available methods. After this, every branch can be considered by the planning method. If one branch cannot be decomposed further, that is all the subtasks are primitive tasks, the found plan will be added to the set of plans (P). In the subsequent recursive process, a similar situation that a subtask may have more than one available method to be decomposed will occur. With the number of such situations increasing, the solution space that will be searched is growing and the planning process will be more and more complex. So, we apply

a control strategy to decide whether a branch will be decomposed further.

Before the definition of the control strategy, the concept of immediate reward needs to be introduced.

An immediate reward is a utility value to measure the quality of a decomposition method. A method decomposes a task into primitive subtasks or non- primitive subtasks. A primitive task can be performed directly using a service operation (or planning operator in planning terms). Clearly, operations suggested by a decomposition method have a direct impact on the overall quality of the solution. On the basis of this, the immediate reward of a decomposition method can be calculated by using the service QoS details (Q), and the corresponding Web services are mapped into operators produced on the certainty branch, which does not have a subtask that can be decomposed by more than one method in the remaining decomposition process until planning is completed.

A Method for Calculating Immediate Reward

In our former work (Chen et al., 2009), we used a fitness function using an average weight mechanism to calculate the immediate reward based on standard QoS criteria including cost, response time, availability and reliability (Canfora, Di Penta, Esposito, & Villani, 2005). However, a UDDI repository usually does not contain the information about standard QoS data. Even if it does, the data is stored for human consideration rather than in the machine readable form required for automatic services selection. In any approach where decisions have to be made in decomposition whether to include a particular service into the set of solutions we clearly encounter a service selection problem.

As we said before, in our previous work we relied on a simplified QoS model assuming data for these to be available, however one of the significant extensions in this paper is the adoption of a service selection method (Reiff-Marganiec, Yu, & Tilly, 2009; Yu & Reiff-Marganiec, 2008) used to obtain data about non-functional properties automatically considering user context information. We use the scores provided through this framework and then calculate the immediate reward for choosing which branches to extend.

The development of the selection method was motivated by the fact that the complexity of business processes and the dynamic nature of the co-operations make it difficult for the business modeler or planner to select appropriate services, manage the compositions efficiently and understand requirements within a dynamic context correctly. The method considers that a service's suitability depends largely on the user's context and does change over time with changes to the user's context as well as the currently available services. For this paper, choosing this selection method is mainly based on three merits. Firstly, it combines evaluation and selection activities, which is consistent with the purpose of choosing a decomposition branch. Secondly, it can deal with a wider range of non-functional attributes than the basic QoS non-functional attributes, in fact it is open so that any non-functional aspect that makes sense for a group of services can be included and not all services need to have the same non-functional properties. Thirdly, this method incorporates the Logic Scoring of Preferences (LSP) method (Dujmovic & Larsen, 2004) which captures the logic relations between criteria rather than just simply using an average weight mechanism thus ensuring that even if large numbers of criteria are considered critical criteria are never overruled by high scores of others and the like (these are termed simultaneity and replacability).

The process using the enhanced selection method to calculate immediate rewards involves 4 steps and is as follows:

Step 1: Obtain relevant non-functional properties

The first step is concerned with gaining values for the relevant non-functional properties. These values are gained from the context information and which non-functional properties are relevant is based on details about the service operations. This had been discussed in

detail in (ICWS09). What is relevant here is to understand that all non-functional criteria are defined through a tuple {*Name, Type, Weight, Value*}, where *Name* is a unique string for identifying the criteria, *Type* is the data type of the criterion (e.g., Boolean, Integer, Set) and is crucial in automatically selecting the right evaluation function (see step 2). The *Weight* reflects the importance of the criterion and has an initial value created at the same time as the criteria, but the actual value might have been modified by the end-users. The *Value* is the current evaluation value of the service. *Values* are obtained from the context information (or in some cases can be directly queried from the service, in which case a query expression would be given here) and might be more or less static (e.g., for a printing service the fact that it is "black and white" is quite static, while the length of the print queue is highly dynamic).

Step 2: Metrics for non-functional properties

Having obtained the current values for each non-functional criterion, we need to calculate the evaluation score E for each criterion for every service. Because there may be many types of values there is a number of evaluation functions dependent on the data type of the criterion.

For example, if the type of the criterion is numerical the evaluation function shown in formula (1) is used.

$$E = \begin{cases} 1 - \left(\dfrac{v_{max} - v}{v_{max} - v_{min}} \right) & if\ w \geq 0 \\ \left(\dfrac{v_{max} - v}{v_{max} - v_{min}} \right) & otherwise \end{cases} \quad (1)$$

In formula (1), v_{min} and v_{max} are the minimum and maximum value of all services which are options based on the values gathered in step 1, v is the value for the current service and w is the weight of the current criterion. A negative weight would mean that a smaller value is desirable (e.g., for cost).

If the criterion is of Boolean type, then an exact match will be used as seen in formula (2):

$$E = \begin{cases} 1 & if\ criteria\ is\ met \\ 0 & otherwise \end{cases} \quad (2)$$

If the criterion is a set type, then the size of the overlapping subset is of interest (see formula (3)):

$$E = {(e_1 + e_2 + \cdots + e_n)}\Big/{n}, \quad with\ e_i\ being\ a\ score\ for\ each\ element \quad (3)$$

Step 3: Calculate all aspects of criteria

After step 2 we have scores for all criteria of all services. These need to be aggregated into a score for each service, which is achieved using a global preference calculation function L:

$$L = \left(\left|w_1\right| E_1^r + \left|w_2\right| E_2^r + \cdots + \left|w_n\right| E_n^r\right)^{1/r}, \quad with\ 0 \leq E \leq 1\ and \sum_{i=1}^{n} \left|w_i\right| = 1 \quad (4)$$

In this formula each E_i is one of the individual scores obtained in step 2 (with $1 < i < n$, and n being the total number criteria for this service). w is the weight of each criteria. r is the logic power value adopted from the LSP method and obtained automatically by a method introduced in Yu and Reiff-Marganiec (2008) – r captures the logical relations between the criteria.

Step 4: Calculate the reward value for each decomposition method

The fourth and final step calculates the immediate reward for evaluating the quality of a decomposition method which is determined by the services coming out in a plan. The immediate reward function is shown as formula (5):

$$R = \left(L\left(S_1\right) + L\left(S_2\right) + \cdots + L\left(S_n\right)\right) / n \quad (5)$$

In formula (5), $L(S_i)$ is a global evaluation value for a service S_i as obtained in step 3. S_1, S_2 ... S_n are the services which are options for decomposition at the current point in the plan. As R captures a normalized score for all services involved in the decomposition, a higher value of R reflects that all services are more desirable and hence the overall solution involving these services is more desirable.

Control Strategy for Planning Process

Because of the complexity of business process, especially the availability and suitability of services, planning processes to find feasible composition plans may do much unnecessary work. One of the extremes is to find just one plan (as in most traditional approaches). The other extreme would be to compute all plans – that is to fully extend all decomposition methods. However, this does not benefit the user as many plans might not be suitable for the current situation, or at least not of sufficient quality and in addition the process to finding them takes possibly quite long. So, it is crucial to get a balance between the number of plans found and the computational effort of finding them by introducing a control strategy.

In our former work, we used a control strategy by comparing a threshold value λ (with $\lambda \geq 0$) to the immediate reward value R of a decomposition method. If $R \geq \lambda$, the planner uses the method to decompose further, else if $R < \lambda$ the planner stops to decompose this branch. This approach requires the user to set the value for λ, allowing them to control how many plans would be retrieved. The case of $\lambda = 0$ would mean that all the branches will be fully extended and hence all possible plans would be found. If a too large value is chosen, it might be that no plans are found.

One of the difficulties with that approach was choosing an appropriate value for λ. Furthermore, choosing the right value for λ in dynamic settings, such as the one proposed here where the reward function is based on dynamic data and context information becomes even more difficult, as it involves a rich reward calculation. We are proposing a reviewed strategy here and will later on discuss choosing good values for λ.

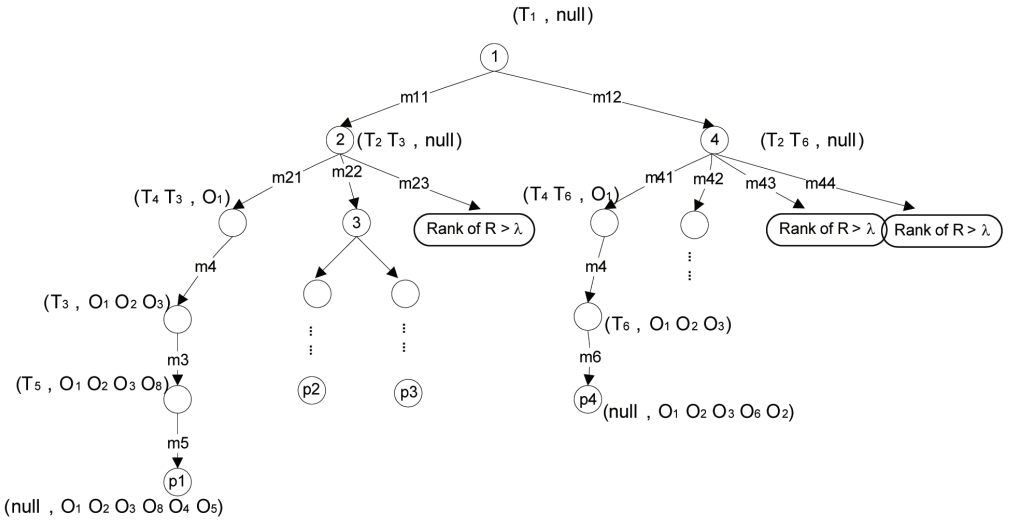
Definition 4 (Control Strategy). There is a threshold value λ with $\lambda \in \mathbf{N}$ which determines the number of decomposition branches that can be extended at a specific decomposition point.

At every decomposition point, that is a node in the tree which has to be decomposed further, a decision has to be made as to how many of the possible methods we wish to decompose. At each such point the planer calculates the immediate reward value for all feasible decomposition methods and ranks them in descending order of immediate reward values. The ranks will be indexed starting from 1. If a decomposition method rank index is i and $i \leq \lambda$ the planner will decompose the method further. Otherwise, that is if $i > \lambda$, the planner will not decompose the branch further. More colloquially, the first λ branches will be decomposed.

Figure 2 shows an example search tree for a planning problem. Let us assume $\lambda = 2$. Now, let's consider node 2, where we identify 3 possible methods to decompose the node: m_{21}, m_{22} and m_{23}. These are already conveniently ranked by their immediate reward, that is $R(m_{21}) \geq R(m_{22}) \geq R(m_{23})$. As $\lambda = 2$ we will only decompose the first two, which equal to the best two choices at this point. m_{23} will not be decomposed further. A similar situation arises at node 4, where again there are 4 methods for further decomposition, and this time m_{43} and m_{44} will not be decomposed further. Node 3 is another case: there are two options here and both would be decomposed further (albeit details are not shown in the figure).

The control strategy ensures that we reduce the size of the solution space that is searched in such a way that the explored solutions will be better suited to demands of the users. In the control strategy previously proposed λ was directly compared to the immediate reward of a method, which meant that it was difficult (if

Figure 2. A search tree for a planning problem



not impossible) to judge what a good value of λ was without knowing the values of the immediate reward calculations. Furthermore, the immediate reward might vary from method to method so that some nodes might have had a many methods further expanded while other might have had very few expanded, leading to a very unbalanced situation. This is even more critical if the immediate rewards are calculated using a framework such as the one presented in this paper where the reward values are much more dynamically gathered.

In Figure 2 methods are shown as labels on branches and nodes are labeled with both a number and a text at the side $n = (T, \pi)$. The number is for convenience, the term in brackets is the list of tasks T which needs to be solved and π is the current partial plan (essentially a sequence of operators) – note that once T is empty, π is a feasible solution and hence a final plan. A node can be reached from the initial state S following the operators in the current (partial) plan π. The algorithm for HTN planning used is as shown in Figure 3.

The new strategy provides greater transparency to the user in that it is clear how many methods will be expanded at each place (assuming that at least that number exists, otherwise all options will be explored). However, it still ensures that the best options are expanded further, as the explored branches will be those with the highest value to the user due to the ranking by immediate reward gained. It may be possible that branches which could lead to better utility further down the line will be cut away prematurely, but that has little consequence to the better quality plans found. In view of the reliability of plans during actual execution, the partial plan composed of the operators which are found on an anterior branch is more important than the one found on the posterior branch. Consequently the plans produced by the HTN planning algorithm for complete decomposition are ensuring best quality.

Since the immediate reward value R measures the quality of a decomposition method, it can also be used to evaluate the quality of plans, and we will come back to this in the next step when we decide on best plans.

Optimality Evaluation Using MDP

After the completion of the HTN planning step, several good plans can be provided to the user, but it is the optimal plan that users are most concerned about. Hence, we proposed

Figure 3. HTN planning algorithm for complete decomposition

```
Procedure CompleteDecomposition_HTN
Input:  S,T D,Q,P, plan ( a sequence of Operators).
Output: true or false
begin
     if T! = ∅
          get the first task t from T
     if t is a non-primitive task
          Find all the available methods M in D.
             M ← {(m,θ): m is an instance of a Method in D, as m = (h,Pre,T )
                        θ unifies { h(m),t}, Pre(m)is true in S.}
          if  M = Φ, return false.
          else
               make N copies of current < S,N >, N = no of (m,θ) in M.
               choose a pair (m,θ) ∈ M to decompose from copies of < S,T >.
               Loop N:
                    compute the immediate reward value R of m by Q.
               rank all R values in descending order
                    and index starting from 1
               remove t from T
               for all methods were index is less than or equal to λ
                    add all the elements in T(m) to T
               else return false.
     else if t is a primitive task
          Find an Operator o = (h Pre Del Add) in D,
               such that h unifies with t and S satisfies Pre.
          if no such o exists, return false.
          else
               modify S by deleting Del and adding Add.
               modify T by removing t.
```

a method to evaluate the optimality using a Markov decision process (MDP) is proposed. MDPs provide a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of the decision maker. MDPs are useful for solving a wide range of optimization problems.

In the process of HTN planning, the choice of multiple decomposition methods can be seen as a decision-making process and the decision-making only connects with the current state. So we construct an MDP model by introducing the probability and reward value for choosing a decomposition method and solve the model to find the optimal plan. The time to choose a method is at decision-making time in the planning process, such as the nodes $(1,2,3,4)$ in Figure 2. First, a list of four objects in MDP should be described as (*S, A,* $P_a(\cdot,\cdot)$, $R_a(\cdot,\cdot)$), where:

- *S* is the state space,
- *A* is the available action set (which is identical to the available decomposition methods set),
- $P_a(s,s')$ is the probability that action *a* in state *s* will lead to state *s'*.

- $R_a(s,s')$ is the immediate reward received after transition to state s' from state s.

Calculation of Transition Probability and Reward

In the MDP process, the calculation of the transition probability and reward is core. The probability for choosing a decomposition method in HTN planning is related to the pre-conditions of the method. Fewer constraints in terms of fewer preconditions will lead to a smaller risk of failure in the actual execution process. Hence, a less restrictive method has a higher probability of being selected.

Assume that a task can be decomposed by k methods M. Each method $m_i \in M$ (with $1 < i < k$) has n_i parameters in its set of preconditions Pre_i. Then, the transition probability from s to s' is defined by formula (6):

$$P_a\left(s \mid s'\right) = P_{m_i}\left(s \mid s'\right) = \frac{\grave{e}\sum_{j=1}^{k} n_j - n_i}{\sum_{j=1}^{k} n_j} with\, \grave{e} = \begin{cases} 1 & if\ Pre_i \subset s \\ 0 & otherwise \end{cases}$$

$$P_a\left(s \mid s'\right) = P_{m_i}\left(s \mid s'\right) = \frac{\grave{e}\sum_{j=1}^{k} n_j - n_i}{\sum_{j=1}^{k} n_j} with\, \grave{e} = \begin{cases} 1 & if\ Pre_i \subset s \\ 0 & otherwise \end{cases} \quad (6)$$

Obviously considering non-functional properties helps in selecting among services with the same functionality and allows for evaluation of alternative execution paths for process adaptation. Moreover, non-functional properties can be used as a basis for cost models that drive process optimization (Garcia & de Toledo, 2006). This motivates us to use the same immediate reward function introduced earlier, (formula (1)) here. This comes with the added advantage that we do not calculate yet a different value.

Solving the MDP by Way of Policy Iteration

The solution to a Markov Decision Process can be expressed as a policy π, a function from states to actions. The standard family of algorithms to calculate the policy calculates two variables repeatedly: one is value V, the utility value of state s, and the other is the policy π which contains actions A. s' is the next state achieved by executing an action $a \in A$ from the current state s. The two variables are calculated by formulae (7) and (8):

$$V\left(s\right) = R\left(s\right) + \tilde{a}\sum_{s'} P_{\eth(s)}\left(s,s'\right)V\left(s'\right),$$

$$where\, \tilde{a}\, is\, a\, discounting\, factor \quad (7)$$

$$\eth\left(s\right) = \arg\max_a \sum_{s'} P_a\left(s,s'\right)V\left(s'\right) \quad (8)$$

After completing the second step of the HTN planning process, the plan set P has n plans. In this stage policies, the set of available actions A and state space S required for the MDP can be determined. To obtain the policies, we simply assume each plan to be a policy. For example the plan p1 from Figure 2 can be expressed as a policy π_1: {(s_1,m11),(s_2,m21)}, and the expected utility of a policy reflects the quality of the plan, and is calculated by formula (9):

$$E_{\eth_i}\left(s\right) = R\left(s\right) + \tilde{a}\sum_{s'} P_{\eth_i(s)}\left(s,s'\right)E\left(s'\right), \quad (9)$$

$$E_{\eth_i}\left(s\right) = R\left(s\right) + \tilde{a}\sum_{s'} P_{\eth_i(s)}\left(s,s'\right)E\left(s'\right),$$

where s is the state in policy $\pi 1$. Formula (9) calculates all the rewards on non-primitive tasks decompositions during the production of a whole plan. As each layered decomposition is considered, the high layers gain more impact for the plan than the low ones.

The policy iteration algorithm is used to find the optimal policy, details are shown in Figure 4. The process is known to converge in

Figure 4. Policy iteration algorithm for MDP

```
Policy Iteration:
start with an arbitrary policy π
for i=1,2,…
   compute E_{π_s}(s) for every s:
```

$$E_{\pi_i}(s) = R\big(s, \pi_i(s)\big) + \gamma \sum_{s' \in S} P_{\pi_i(s)}(s, s') E(s')$$

```
   for every s:
```

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_a(s, s') V(s')$$

```
   if π_{i+1} = π_i then break
rof
   π_{i+1} is the optimal policy
```

a finite number of iterations and ends with the optimal policy (for a proof, we refer to Ke, 2004).

Policy Iteration:

start with an arbitrary policy πfor i=1,2,… compute $E_{\delta_s}(s)$ for every s:

$$E_{\delta_i}\left(s\right) = R\left(s,\delta_i\left(s\right)\right) + \tilde{a}\sum_{s'\in S} P_{\delta_i(s)}\left(s,s'\right) E\left(s'\right)$$

for every s:

$$\delta\left(s\right) = \arg\max_{a\in A} \sum_{s'\in S} P_a\left(s,s'\right) V\left(s'\right)$$

if $\delta_{i+1}=\delta_i$ then breakrof

δ_{i+1} is the optimal policy

Evaluation

A Reality-Based Scenario

To demonstrate the feasibility of our composition approach, we use a reality-based scenario which illustrates an online shopping process.

Supposed that a customer wants to buy a digital camera online and he has several requirements which are: the brand and model (Canon IXUS85), the price (it should be lower than 1400￥) and the place of delivery (the Economic Development Zone of Qingdao, China). There will clearly be many feasible plans after services composition as we expect many shops to sell and deliver cameras. The number of plans will immediately reduced, because the reward function will not explore options where the specific model is not stocked, where it is too expensive or where carriers to not deliver to Qingdao.

Looking at the case study from a more technical level, we have a several items. First of all we have the definition of the goal and initial state, with the first two elements being the initial state and buy camera being the initial task in the task list:

```
(defproblem problem shopping
((toHasGood camera) (hasMoney 2500))
((buy camera 2500)))
```

The first step of the planning process was concerned with extracting domain knowledge from the service descriptions; recall that we distinguished between operators (mapping

to atomic processes which can be executed directly) and methods (describing how to decompose composite processes). In our domain description file, we have 19 operators and 29 methods. Examples of these are:

```
(:operator (!using_TaoBao ?x) ((toHas-
Good     ?x)) () ((environment ?z)))
(:operator (!using_eBay ?x) ((toHas-
Good ?x)) () ((environment ?z)))
(:operator (!shop_1 ?x) () () (inShop
?z))

(:method (chooseEnvirionment ?x ?y)
    ((toHasGood ?x))
    ((!using_TaoBao ?x) (doshop_TaoBao
?x ?y))
)
(:method (buy ?x ?y)
    ((toHasGood ?x) (hasMoney?y))
    ((chooseEnvirionment ?x ?y) (de-
liver ?x ?y) (!evaluate ?x))
)
```

Operators have a name, a precondition and if that is fulfilled an effect on the environment. For example the operator using_TaoBao has the precondition (toHasGood ?x), and if the precondition is satisfied, it will add the (environment ?z) into the state .

Methods similarly do have a name, a precondition and a list of subtasks. So for example (buy ?x ?y) can decompose the task named 'buy' if the conditions (toHasGood ?x) and (hasMoney ?y) are satisfied into the subtasks (chooseEnvirionment ?x ?y), (deliver ?x ?y) and (!evaluate ?x).

The final piece of domain description available is the information about the non-functional properties of the services. Examples are:

```
!using_TaoBao
time 25 availability 0.9 cost 1.5 lan-
guage   English,Chinese
!using_eBay
time 35 availability 0.95 cost 2.5
language       Chinese
!shop_1
safty high cost 20 bankcard
ICBC,CCB,ABC   privacy high
```

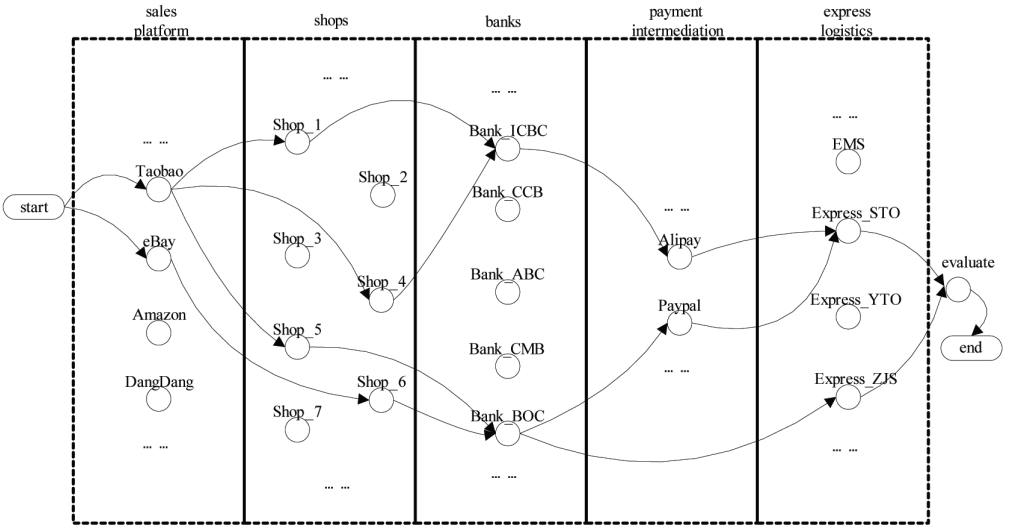
While in a real operating environment these might be obtained in a more dynamic fashion, we provide these as a description file for our experiments. The file, of which the above is an extract, contains two lines per service: the first line is the service name(operator), the second line contains the non-functional attributes formulated as name value pairs. So for example the operator using_TaoBao has non-functional attributes 'time', 'availability', 'cost' and 'language' with respective values of '25', '0.9', '1.5' and 'English,Chinese'.

We have design a system to simulate the services composition by using our method. Using the system involves three steps. First, before planning starts we initialize the system by providing the SHOP2 domain description and the services quality context information (an example was shown above). After that, users submit their requirements and the planning begins which is the core process for services composition. This stage includes the planning, service evaluation and the expected utility calculation. As this system is intended for experimental use, the plans are shown when planning has finished and the expected utility value and execution sequence for each plan can be viewed. Obviously the plan with the highest utility value is the best plan. An example of services composition plans based on the case study is shown in Figure 5, and we can see that there are six plans satisfying the customer's requirements.The whole planning process is shown in Figure 6.

Analysis

There are a number of areas to be analyzed. Of course there is the issue of gathering and evaluating the criteria value for the non-functional properties: The actual calculations are quite straight forward, but of course for each service each criteria needs to be looked at. However, this has been shown to be quite feasible and efficient in Yu (2009). Furthermore, for this

Figure 5. A shopping example of services composition in E-Commerce



specific aspect it can be commented that taking a little longer but identifying the right services will still be much faster than executing services which form part of process, especially if we realize during execution that a chosen service is simply not suitable.

More crucially, and much closer related to the key aspect of this paper is the complexity of the planning task. It is obvious that tasks with different complexity have different sizes of solution space. Users would expect that sufficient and better plans can be provided – naively assuming that all possible solutions would be considered. However a task might be so complex that the search time will be extremely long and there will be many redundant plans. We can control the search space by adjusting the threshold value λ which by allowing or hindering decomposition of certain branches controls the maximum number of extended decomposition branches and hence the overall search space.

We have conducted some experiments to show the effect of control strategy and gain an understanding of the complexity involved – while this could be done theoretically we have chosen the more practical approach as we feel that it provides a good understanding and is more aligned with the aim of the overall work: to be suitable for us in reality.

The hardware environment for the experiment was a standard PC with a Pentium 4 CPU running at 2.8 GHZ with 512MB Memory. The setup was such that we have searched the whole solution space as a reference value and then gradually reduced the threshold value λ.

The results for the shopping example presented earlier can be found in Table 1. Obviously, the threshold value λ is directly proportional to the planning time *t* and the number of found plans *n*. In each set of plans found, we had indicated the optimal plan according to its expected utility value. We observed that each set contained the same optimal plan both in terms of services selected and their ordering (note that kept the environment stable, that is the non-functional properties are always evaluated to the same scores; changing that would obviously lead to different plans). The optimal plan is the service sequence:

```
"!using_DanDang!shop_1!bank_
ICBC!Alipay!sendBy_Express_
STO!evaluate".
```

Figure 6. The planning process for services composition

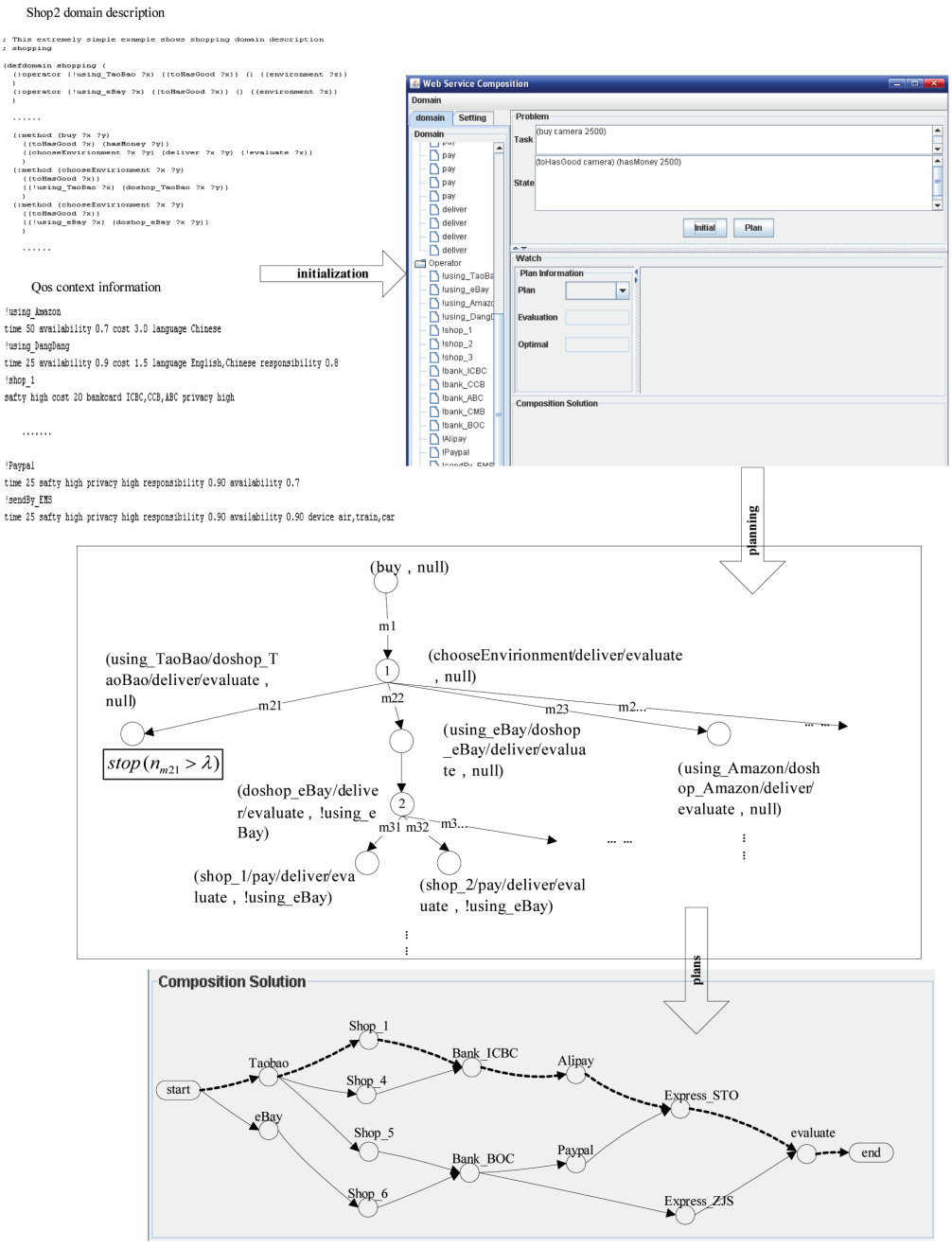


Fig. 6 The planning process for services composition

Table 1. Planning results under different threshold (a)

λ	*n*	*t*(ms)	optimal plan
8	128	250	plan_98
7	112	234	plan_86
6	96	219	plan_74
5	80	203	plan_62
4	64	188	plan_50
3	27	141	plan_20
2	8	78	plan_5
1	1	31	plan_1
0	0	-	-

To identify suitable values for λ, we need to consider the tradeoff between finding sufficient plans and being economical on the time used. Figure 7 shows two lines depicting these issues. One line indicates the relation between threshold λ and the number *n* of plans found; the other indicates the relation between threshold λ and the reciprocal of the planning time 1/*t*. Clearly these two lines will intersect (one will always increase, the other decrease) and the intersection point will fall into an interval $[\lambda_i,\lambda_{i+1}]$. In the given scenario the interval is [2,3]. Manually analyzing the result sets, we determined that setting λ to a boundary value of the interval determined by the interaction point, a good number of plans with little redundancy is found while maintaining a good search time. We conclude that the intersection point of the two graphs determines good values for λ. In general we found from this and a number of other case studies that values of 2 or 3 are generally best for λ.

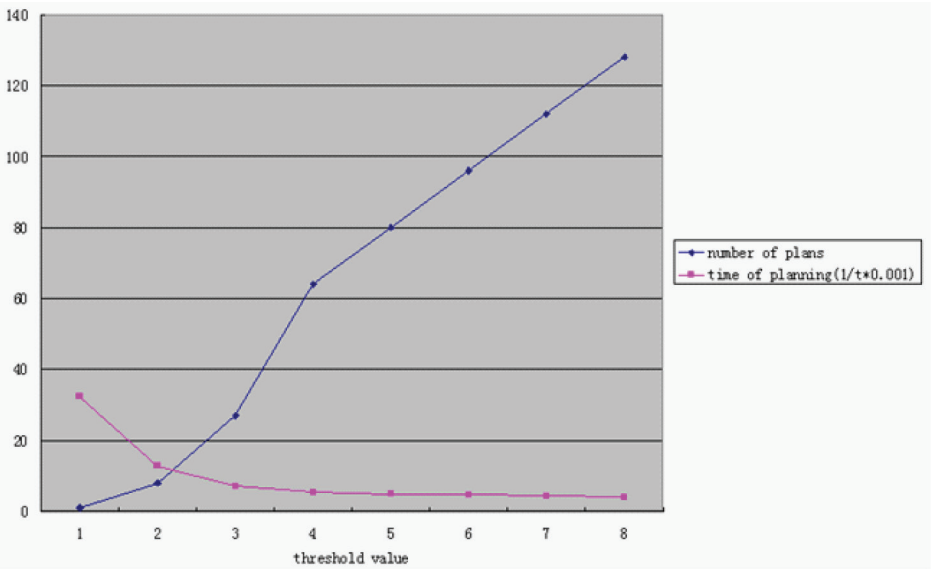
RELATED WORK

Kuter et al. (2005) present an HTN planning algorithm, ENQUIRER, designed for planning domains and in which the information about the initial state of the world may not be complete. By using ENQUIRER, information is discoverable through planning-time information gathering queries. In ENQUIRER, some limitations in their previous work (Sirin et al., 2004) are overcome, which makes service composition sound and complete. Our work is based on that presented by Sirin et al. (2004). Our approach improves the composition method to provide multiple plans and also to consider the non-functional properties of Web services in the planning process in addition to provide the best solution for each user in their given situation.

The work by Lécué (2009) and Lécué, Delteil, and Leger (2008) focuses on casual links, that is the functional dependencies between services. This is formulated as an Integer Programming problem and the solver is entrusted with finding a sequence of operations, that is one plan. The work presented here uses HTN planning rather than a constraint based solution, but more crucially enhances on two aspects: we are looking for the best plan in that the planning part of our work searches for multiple plans, not just one and furthermore does not consider non-functional properties at all.

Further, Zhang, Zhang, Cao, and Mou (2004) propose an enhanced HTN planning method combined with partial-order planning (POP) for service composition in which action decompositions are used as plan refinements in POP. Compared to the pure HTN planning, their approach can solve certain tasks, which are novel conjunctive goals. In our approach, we also focus on the decomposition in HTN planning, improving the decomposition for

Figure 7. Planning results under different threshold (b)



non-primitive tasks, but rather than trying to solve new types of goals, we wish to search more potential feasible solutions.

Paik et al. (2006) suggest a combined architecture, which consists of HTN planning and Constraint Satisfaction Problem (CSP) as an underlying problem-solving engine to automate Web service composition, especially for composition problems with many parameters. In the architecture, a complete semantic concept for CSP is defined using OWL, which allows for solver agents to automatically solve a given problem with greater flexibility and more intelligently. This work focuses using CSP for the semantic web. The CSP solver is part of the combined architecture, but not tightly integrated into the HTN planning.

Doshi, Goodwin, Akkiraju, and Verma (2004) model the workflow composition problem as an MDP, which handles non-deterministic behaviors of Web services in dynamic environments during the plan execution phase. A policy computed by MDP for generating workflows is capable of optimally recovering from Web service failures. MDPs have been used by other for related problems as well, for example for the selection of single services (Cai, Luo, Qian, & Gao, 2005). We use MDP to evaluate optimality of plans when selecting among multiple available plans.

Thiagarajan and Stumptner (2007) consider service composition as configuration tasks, assuming an abstract workflow (that is the structure of a composition). In their work they use constraint based generation of the plans representing the composition problem as a constraint based meta model. They also discuss the inclusion of cost-based optimisation and preferences. Similar efforts are presented by Hassine, Matsubara, and Ishida (2006) use a CSP solver to find a solution of instantiating an abstract workflow with concrete ones with the goal of satisfying the users' requirements at a global level. User requirements are expressed as constraints. Our work differs from these efforts in that we do not require an abstract workflow.

CONCLUSION AND FUTURE WORK

In this paper, a novel composition model based on HTN planning with MDP-LSP has been

proposed. With this model, more than one plan is found and the evaluation mechanism in the model provides the optimal plan based on non-functional aspects.

MDP is an efficient method to solve optimization problems, like choosing the best plan in service composition. In order to demonstrate the feasibility and validity for using MDP in conjunction with HTN to find the optimal plan, we experimented on an e-travel composition example with random QoS data in our former work (Chen et al., 2009). In that experiment, we have found that while this returned the right result and worked well, there were some aspects that could be improved. In this paper, we presented a new immediate reward function to adjust to dynamic context information for service evaluation. The general problem with identifying a cut-off threshold in the reward function and the further complication of that matter due to the more dynamic utility values led to the consideration of a new control strategy. This has been presented in this paper and has been applied to a larger, more realistic case study.

With a choice of Web services composition plans, users can be more flexible in accomplishing their tasks in the most suitable way. They can adopt the optimal plan that our method provides, but they can also choose freely according to their own preference from a number of alternatives. Moreover, when executing the selected plan results in failure, candidate plans can ensure the tasks will be completed without constraints slacking or premises increasing.

While our method can provide multiple plans for users, we will explore a re-planning mechanism to be used when plan execution results in failure. Using this mechanism, a process of plan execution can be continued automatically from an appropriate service node and the negative impact of a failure will be minimized.

ACKNOWLEDGMENT

This work is jointly supported by the National Natural Science Foundation of China (No.60672121), National Key Basic Research Program of China (973 Program) (2009CB320406) and the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No. 60821001).

REFERENCES

Akkiraju, R., Srivastava, B., Ivan, A.-A., Goodwin, R., & Syeda-Mahmood, T. (2006, September). SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition. In *Proceedings of the IEEE International Conference on Web Services*, Chicago, IL (pp. 37-44).

Cai, D., Luo, Z., Qian, K., & Gao, Y. (2005, November). Towards efficient selection of Web services with reinforcement learning process. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 05),* Hong Kong (pp. 372-276).

Canfora, G., Di Penta, M., Esposito, R., & Villani, M. L. (2005, June). An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, Washington, DC (pp. 1069-1075).

Chen, K., Xu, J., & Reiff-Marganiec, S. (2009, July). Markov-HTN Planning Approach to Enhance Flexibility of Automatic Web Service Composition. In *Proceedings of the IEEE International Conference on Web Services*, Los Angeles, CA (pp. 9-16).

Doshi, P., Goodwin, R., Akkiraju, R., & Verma, K. (2005). Dynamic Workflow Composition: Using Markov Decision Processes. *International Journal of Web Services Research*, *2*(1), 1–17. doi:10.4018/jwsr.2005010101

Dujmovic, J. J., & Larsen, H. (2004). Properties and modeling of partial conjunction/disjunction. In *Current Issues in Data and Knowledge Engineering: Proceedings of the Eurofuse Workshop on Data and Knowledge Engineering* (pp. 215-224).

Garcia, D. Z. G., & de Toledo, M. B. F. (2006, November). Semantics-enriched QoS policies for web service interactions. In *Proceedings of the 12th Brazilian Symposium on Multimedia and the Web (WebMedia 2006)* (pp. 35-44).

Gerevini, A., & Long, D. (2005). *Plan Constraints and Preferences in PDDL3 (Tech. Rep.)*. Brescia, Italy: Department of Electronics for Automation, University of Brescia.

Hassine, A. B., Matsubara, S., & Ishida, T. (2006). A Constraint-Based Approach to Horizontal Web Service Composition. In *Proceedings of the 2006 International Semantic Web Conference (ISWC 2006)* (pp. 130-143).

Kuter, U., Sirin, E., Parsia, B., Nau, D. S., & Hendler, J. A. (2005). Information gathering during planning for Web Service composition. *Journal of Web Semantics*, *3*(2-3), 183–205. doi:10.1016/j.websem.2005.07.001

Lécué, F. (2009, October). Optimizing QoS-Aware Semantic Web Service Composition. In *Proceedings of the 2009 International Semantic Web Conference (ISWC 2009)*, Chantilly, VA (pp. 375-191).

Lécué, F., Delteil, A., & Leger, A. (2008). Optimizing Causal Link Based Web Service Composition. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)* (pp. 45-49).

Lin, N., Kuter, U., & Hendler, J. (2007, July). Web Service Composition via Problem Decomposition across Multiple Ontologies. In *Proceedings of the IEEE Conference on Services*, Salt Lake City, UT (pp. 65-72).

Nau, D., Muñoz-Avila, H., Cao, Y., Lotem, A., & Mitchell, S. (2001, August). Total-Order Planning with Partially Ordered Subtasks. In *Proceedings of the IJCAI 2001 Conference*, Seattle, WA.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Paik, I., Maruyama, D., & Huhns, M. N. (2006, September). A Framework for Intelligent Web Services: Combined HTN and CSP Approach. In *Proceedings of the IEEE International Conference on Web Services*, Chicago, IL (pp. 959-962).

Reiff-Marganiec, S., Yu, H. Q., & Tilly, M. (2009). Service Selection based on Non-Functional Properties. In *Proceedings of the ICSOC 2007 Workshops* (LNCS 4907, pp. 128-138).

Schuschel, H., & Weske, M. (2004, June). Automated Planning in a Service-Oriented Architecture. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Modena, Italy (pp. 75-80).

Sirin, E., Parsia, B., Wu, D., Hendler, J. A., & Nau, D. S. (2004). HTN planning for Web Service composition using SHOP2. *Journal of Web Semantics*, *1*(4), 377–396. doi:10.1016/j.websem.2004.06.005

Thiagarajan, R., & Stumptner, M. (2007). Service Composition With Consistency-based Matchmaking: A CSP-based Approach. In *Proceedings of the ECOWS 2007 Conference* (pp. 22-32).

Weiss, M., Esfandiari, B., & Luo, Y. (2007). Towards a classification of web service feature interactions. *International Journal of Computer and Telecommunications Networking*, *51*(2), 359–381.

World Wide Web Consortium. (2004). *OWL-S: Semantic Markup for Web Services.* Retrieved January 27, 2010, from http://www.w3.org/Submission/OWL-S/

Yu, H. Q. (2009). *Context Aware Automatic Service Selection*. Unpublished doctoral dissertation, University of Leicester, UK.

Yu, H. Q., & Reiff-Marganiec, S. (2008, July). A Method for Automated Web Service Selection. In *Proceedings of the 2nd International Workshop on Web Service Composition and Adaptation (WSCA-2008)* (pp. 513-520).

Zhang, J., Zhang, S., Cao, J., & Mou, Y. (2004, September). Improved HTN Planning Approach for Service Composition. In *Proceedings of the IEEE International Conference on Services Computing*, Shanghai, China (pp. 609-612).

ENDNOTE

[1] In this paper, HTN planning is based on the Close World assumption, which is means all the tasks for expressing users' requirements must be one of the elements in task ontology.

Jiuyun Xu is a professor at School of Computer & Communication Engineering located at China University of Petroleum. He obtained his PhD in Computer Science in 2004 from Beijing University of Posts & Telecommunications. He was as an honorary research fellow visiting University of Leicester, which is sponsored by China Overseas Scholarship Committee. In his PhD, Jiuyun investigated Feature Interactions in Next Generation Networks. His research interests are focused on Semantic Web Service Composition with a specific view of using Immune Algorithm, Markov Decision Process and other AI techniques which are not only to find suitable compositions, but to find the best possible plans based on non-functional properties. Another keen area of interest is that of Feature Interactions, where Jiuyun is looking at runtime detection and resolution mechanisms. Jiuyun is also interested in semi-automatic or automatic generate ontology, ontology mapping in Semantic Web and natural language understanding. Jiuyun has extensively published in these areas. Jiuyun is a session Chair of the third International Conference on Natural Computation. And Jiuyun is the member of Yocsef (China Computer Federation Young Computer Scientists & Engineers Forum) QingDao Academic Committee.

Kun Chen has obtained a Bachelor degree of Computer Science at China University of Petroleum in 2007. He is currently a master student under the supervision of Jiuyun Xu. His main research interests concern Semantic Web services, particularly investigating services composition and feature interaction among services. The type of problem he looks at is how to enhance the services composition based on AI planning techniques and how to solve the conflicts during the services execution.

Stephan Reiff-Marganiec is a Senior Lecturer in the Department of Computer Science at the University of Leicester, which he joined in 2003. He worked in the computer industry in Germany and Luxembourg for several years. From 1998 to 2001, he worked as a Research Assistant on the EPSRC HFIG project at the University of Glasgow, while at the same time reading for a PhD in Computing Science. The work performed at Glasgow investigated hybrid approaches to the feature interaction problem. From 2001 to 2003 Stephan worked as a Research Fellow on the EPSRC ACCENT project at the University of Stirling, investigating policies, emerging features and associated conflict resolution techniques. Stephan has since been working in the areas of services and features considering service selection based on non-functional properties as well as policies as mechanisms for system flexibility. Stephan has in the order of 40 publications in international conferences and journals in these areas. Stephan was responsible for organising the British Colloquium for Theoretical Computer Science in 2001 and again in 2004 and since 2004 has been treasurer of BCTCS. He was also co-Chair of the 8th and 10th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI05 and ICFI09), co-Chair of the second, third and fourth Young Researchers Workshop in Service Oriented Computing (YR-SOC 2007, 08 and 09) and is senior member of the steering committee for YR-SOC. Stephan was principal investigator of the project "Ad-Hoc Web Applications" funded by the Nuffield foundation and leader of workpackages and tasks in the EU funded projects Leg2Net, Sensoria and inContext focusing on automatic service adaption, context aware service selection, workflows and rule based service composition. He is co-editor of the Handbook on Non-functional properties for Service oriented Systems to appear in 2011.