Future Generation Computer Systems 27 (2011) 627–636

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A workflow framework for intelligent service composition

Xudong Song [a,b,*], Wanchun Dou [a,b,*], Jinjun Chen [a,c]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, PR China

[b] Department of Computer Science and Technology, Nanjing University, Nanjing 210093, PR China

[c] Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia

ARTICLE INFO

Article history:
Received 11 June 2010
Accepted 28 June 2010
Available online 4 July 2010

Keywords:
Workflow framework
Service composition
AI planning
CSP

ABSTRACT

In practice, service composition and its evaluation are initiated by participant web services' functional and non-functional attributes. It often consists of functional property-based service location, non-functional property-based quality evaluation, and then a finial executive plan. This scheduling process is often unfolded in a workflow framework. Generally, selecting qualified services and composing them manually are time-consuming and error-prone. In view of this challenge, a workflow framework is presented in this paper for intelligently navigating service composition. This workflow framework consists of two primary processing modules, i.e., Planning Module and CSP (Constraint Satisfaction Problems) Solving Module. Taking advantage of services' functional attributes, Planning Module aims at producing referred composite plans that play as general patterns. CSP Solving Module focuses on selecting qualified services to implement the composite plans, based on their non-functional attributes' evaluation. Finally, a case study is presented to demonstrate the validity of framework.

1. Introduction

A web service is a functionality module which is realized by web-accessible programs, databases, sensors, and a variety of other physical devices on the web and identified by URIs [1]. Web services can be advertised, discovered, located and invoked on the web. Generally, a service is always specified by its functional attributes, including inputs, outputs, preconditions and effects, and its non-functional attributes, e.g. time, price, availability, etc. [2]. Widely available and standardized Web services make it possible to realize Business-to-Business Inter-operability by inter-connecting Web services provided by multiple business partners that own demanded functional properties [3,4]. Generally, service composition is often identified by two features [1]. The first issue is typically called composition synthesis, which is about how to combine and coordinate the component services to fulfill a client's request in terms of a specification. The second issue is about service execution. In service execution, supervising and monitoring a composite service are often referred to as orchestration which concerns how to effectively coordinate services engaged in a service composition. The service orchestration could be promoted in the form of workflow paradigms, such as checkpoint selection and dynamic verification of fixed-time constraints [5,6]. This paper concentrates on the problem of composition synthesis.

With fast growing number of web services, locating appropriate services and composing them manually become tedious, time-consuming and error-prone [3]. To make service composition automatic, semantic web technology [7] has been exploited in service description to make information about services formal and machine-understandable, which is called Semantic Web Service.

The Ontology Web Language for Services (OWL-S) [8] is the most direct outcome of the development of Semantic Web Service. It provides web service advertisers with a core set of markup language constructs to describe services' properties and capabilities in a formal form. It facilitates the automation of Web services, including automated Web service discovery and composition. It divides the semantic description of a web service into three components: service profile, process model and grounding. The service profile describes what a service does by its input parameters, output parameters, preconditions and effects. The process model describes how a service works. It is either an Atomic Process that is directly executed or a Composite Process that is a combination of sub-processes. Sub-processes are either Atomic Processes or Composite Processes connected by a set of control constructs including Sequence, Unordered, Choice, If-Then-Else, Iterate, Repeat-Until, Repeat-While, Split and Split+Join. The grounding contains the details of how to access a service by specifying a communication protocol, parameters to be used in the protocol and the serialization techniques to be employed for the communication.

In practice, the service profile of OWL-S primarily describes a service's functional attributes. It lacks effective ways in dealing with a service's non-functional attributes. In practice, the services

* Corresponding author at: State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, PR China.
E-mail addresses: xudong0110@yahoo.cn (X. Song), douwc@nju.edu.cn (W. Dou).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.06.008

that have the same functions could be distinguished by its non-functional attributes. [9,2,10,11] present some ways to describe QoS criteria which could be incorporated into OWL-S. Although they are not perfect in some aspects, all those efforts aimed at making QoS description more flexible and extensible from different perspectives. These QoS descriptions, combined with the service profile, provide a complete formal description of a Web service. This paper mainly utilizes the service profile and process model to investigate an intelligent service composition. More specifically, service profile is used to describe functional attributes of a service, and process model aims at constructing a planning repository.

With unambiguous descriptions of web services, AI Planning technology was investigated and used in service composition in [12,13]. However, [12,13] did not consider QoS of composed services. These composed services can be executed directly. A typical planning technology is HTN (Hierarchical Task Network) planning [14]. SHOP2 (Simple Hierarchical Ordered Planner 2) [15], a typical HTN planner, is used in this paper to produce several plans to fulfill a user's functional requirements, i.e. what steps should be done. In the produced plan, each instantiated operator corresponds to a set of services which have the same function described in the same service class. Compared with [12,13], our plans cannot be executed and need to be processed further.

The plans only tell what kinds of services can be used to accomplish a functionality required by a user without any further detailed non-functional information such as time, availability and so on. These non-functional requirements can help find appropriate services to fill in the corresponding instantiated operators in the plans to produce real composite services, so several rules have been designed to transfer a plan to a CSP (Constraint Satisfaction Problems) [16]. After solving CSP, appropriate services can be picked out and the corresponding composite services can be generated from the plans.

CSP-based service composition methods are used in [17–19]. But in [17,18], some evaluations about QoS such as time cannot be solved well in some circumstances because of intrinsic characteristics of SHOP2. In [19], only CSP is used to accomplish service composition. Because it lacks domain knowledge, CSP is less efficient than SHOP2 in some cases [16]. In our paper, SHOP2 concentrates on services' functional attributes and CSP on services' non-functional attributes. So, SHOP2 does not need to take QoS into account while it is executed to avoid intrinsic limits of SHOP2. CSP only solves QoS problems, which reduced the size of the problem in [19].

To the best of our knowledge, there is little work to incorporate SHOP2 and CSP into intelligent service composition. In this paper, we focus on investigating how to develop a workflow framework for intelligent service composition using SHOP2 and CSP. More specifically, SHOP2 takes advantage of functional attributes of services in service composition, and CSP solving aims at service selection by non-functional attributes of services.

The paper is organized as follows: In Section 2, an overview of the workflow framework is put forward for intelligent service composition. In Section 3, the components of the framework are investigated in detail. Furthermore, a case study is presented to demonstrate the feasibility of the framework in Section 4. In Section 5, related works and comparison analysis is presented. At last, some conclusions are proposed in Section 6.

2. Overview of workflow framework for intelligent service composition

The framework is illustrated in Fig. 1. It has two processing modules (Planning Module and CSP Solving Module) and three

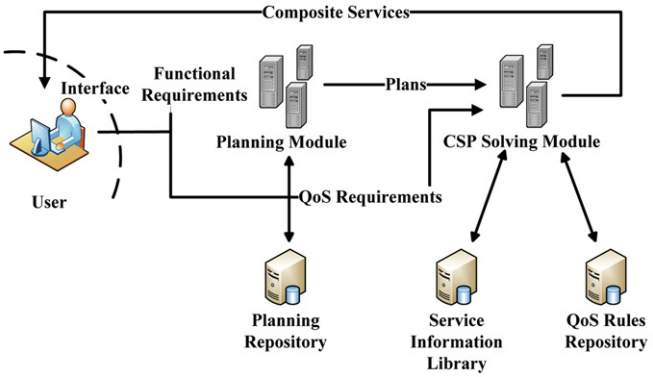

Fig. 1. An overview of workflow framework for intelligent service composition.

repositories (planning repository, service information library and QoS rules repository).

Planning Module is carried out by SHOP2 [15], a widely used HTN planner. SHOP2 generates the steps of each plan in the same order that those steps will later be executed, so it knows the current state at each step of the planning process. This reduces the complexity of reasoning by eliminating a great deal of uncertainty about the state of the world. And also it has a substantial expressive power to describe the corresponding functional attributes of service classes.

SHOP2 is a partial-order planner that produces total-order plans and uses domain-dependent knowledge to generate plans quickly. The domain-dependent knowledge, including operators, methods and tasks, used to tell how to decompose a task, is put in a planning repository which is designed by the experts in that field. Task is divided into two types, primitive tasks which can be accomplished by choosing one instantiated operator and non-primitive tasks which should be decomposed by choosing an appropriate method and instantiating it. The objective of SHOP2 is to produce a sequence of operators using methods to decompose tasks into smaller and smaller tasks recursively until primitive tasks which can be accomplished directly by instantiated operators. In the framework, an instantiated operator corresponds to a service class, which accomplishes the same function, so the plans produced are abstract. Also, the plans only tell how to carry out a task with no other constraints, such as QoS requirements. They cannot be executed directly after planning until appropriate service instances are selected. The number of plans is so large and some of them have strong similarity, so they can be combined and specified by Petri Net [20] for further analysis. The resulted number is much smaller and the plans are easier to be processed by a unified method.

The next step is to choose appropriate service instances for instantiated operators in the plans. This is accomplished by CSP Solving Module. The inputs of CSP Solving Module are plans produced by Planning Module and QoS requirements of users. The CSP Solving Module needs the QoS rules repository to translate plans into CSPs and then take advantage of the service information library to select appropriate service instances. Here, the functional requirements of corresponding services will not be taken into account because they are considered in the plans after planning.

The QoS rules repository consists of a set of QoS rules of different QoS criteria that can map a plan described by Petri Net to a set of CSP constraints. The service information repository has a store of information about services on the web and keeps information consistent with the Web services by periodically updating.

In CSP Solving Module, all satisfactory solutions are actual composite services that can accomplish a user's functional and QoS requirements and can be executed directly. All composite services should be ranked by exceptional probability before submitted to the user.

```
Input: plan (O, R)
Output: dependence relationship R'
Procedure:
Begin
R' = empty
Foreach oi in O do
   If oi happens before oj and oj depends on oi
    Then add (oi, oj) into R';
EndForeach
Return R'
End
```

Fig. 2. Dependence detection algorithm.

```
Input: plan (O, R')
Output: dependence relationship Rd
Procedure:
Begin
Rd = R';
Foreach oi in O do
   If oi depends on oj
      If exists o k that oi depends on ok and ok depends on oj
        Then remove (oi, oj) from Rd;
EndForeach
Return Rd
End
```

Fig. 3. Dependence reduction algorithm.

3. Component developments

The main components composed in the workflow framework and their functionalities have been briefly introduced in Section 2. In this section, these components and their development will be explored in detail to demonstrate the intelligent composition mechanism of the workflow framework.

3.1. Planning module

Definition 1 (*Instantiated Operator*)**.** An Instantiated Operator corresponds to a service class in a particular domain, and can be represented as:

$o \equiv \langle D, SC, In, Out, Pre, E \rangle$.

It means that an instantiated operator o, corresponding to a service class SC in the domain D, has an input set of In, output set of Out, precondition set of Pre and effect set of E. In, Out, Pre and E correspond to input parameters, output parameters, preconditions and effects of a service described by service profile in OWL-S.

Definition 2 (*Happens Before*)**.** Given a set of instantiated operators O and a binary relation set $R \subseteq O \times O$, for each o_1 and o_2, where $o_1 \neq o_2 \wedge o_1, o_2 \in O$, o_1 happens before o_2 iff

(1) $(o_1, o_2) \in R$;
(2) $\exists o \in O$, o_1 happens before o and o happens before o_2.

Any plan can be represented as a pair (O, R), where O is a set of instantiated operators and R is a Happens Before relationship on O. This representation only includes structural information of an original plan.

Definition 3 (*Dependence*)**.** In a plan (O, R), an instantiated operator o_i is dependent on another instantiated operator o_j if and only if:

1. $o_i.Pre \cap o_j.E \neq \emptyset$; or
2. $o_i.In \cap o_j.Out \neq \emptyset$.

If o_i does not depend on o_j and neither does o_j, then the relationship between o_i and o_j is called concurrency.

o_1 happens before o_2, which means o_1 is executed first, then o_2. However, o_2 may happen before o_1, which means o_1 and o_2 can be executed concurrently, or o_1 must be executed first, which means o_2 is dependent on o_1. The original sequential plans generated from SHOP2 only contain the relationship of Happens Before from which Dependence can be inferred.

According to Definition 3, it is obvious to detect the partial-order relationship implied by the plan and it is easy to get the partial-order plans from the total-order ones by the algorithm illustrated in Fig. 2. After the algorithm of Fig. 2 is applied, plan (O, R′) is obtained.

In some cases, R′ contains redundant information. For example, if (o_1, o_2), (o_2, o_3), $(o_1, o_3) \in R'$, then (o_1, o_3) can be removed from R′ without loss of information about dependence, because (o_1, o_2), (o_2, o_3) implies (o_1, o_3). In terms of this transition relationship, dependence set of R′ could be reduced to R_d which contains core information about dependencies among instantiated operators. The algorithm of reduction is illustrated in Fig. 3.

Given two plans (O_1, R_d^1) and (O_2, R_d^2), it is easy to get Happens Before relationship R_1 and R_2 on O_1 and O_2 respectively from R_d^1 and R_d^2.

Definition 4 (*Path*)**.** A path P in a plan (O, R) is $(o_1, o_2, \ldots, o_n)$ where $o_i \in O$ $(1 \leq i \leq n)$ and $(o_1, o_2, \ldots, o_n) \in \underbrace{R \times R \times \cdots \times R}_{n-1}$.

n is the length of a path. o_i is a direct precedent of o_{i+1} and o_{i+1} is a direct successor of o_i. o_1 is the start of the path P and o_n the end. A path P_i is said to be covered by P_j iff

$$\forall o_k((o_k \in P_i \rightarrow \exists o_l(o_l \in P_j \wedge o_l \equiv o_k)) \wedge \forall o_i(1 \leq i \leq k)(o_i \in P_j)).$$

A path is called a complete path, if it can only be covered by itself.

Definition 5 (*Combinable Plans*)**.** Two plans (O_1, R_1) and (O_2, R_2) are called combinable if and only if for each complete path of (O_1, R_1), there exists a complete path of (O_2, R_2) such that the start and the end of the paths are identical, and vice versa.

Definition 5 tells that two plans that have the same start conditions and end conditions are combinable. In order to combine multiple plans into one, two important constructs are introduced in Definitions 6 and 7.

Definition 6 (*AND Split/Join*)**.** That all successors will be executed only after all the precedents finish is called AND Split/Join.

Definition 7 (*OR Split/Join*)**.** That only one of the successors will be executed after all the precedents finish is called OR Split. That all successors will be executed after only one of the precedents finishes is called OR Join. Each path between OR Split and its corresponding OR Join has different preconditions of the first instantiated operator.

In order to present the two constructs explicitly and precisely, Petri net [20] is used to describe combined plans. A Petri net (PN) has two types of nodes, namely, places and transitions. A place is represented by a circle and a transition by a bar. Places and transitions are connected by arcs directed and connected either from a place to a transition or from a transition to a place. The number of places is finite and nonzero. The number of transitions is also finite and nonzero. Places may contain tokens. In the classic Petri Net, a place can include up to 1 token. If all places directly

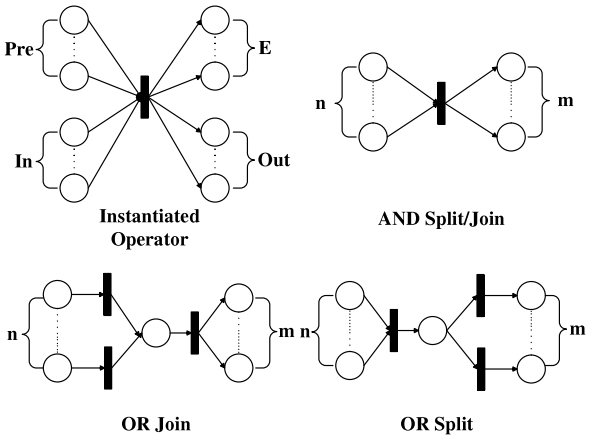

Fig. 4. Instantiated operator, AND Split/Join, and OR Split/Join.

connected to a transition contain one token, the transition is enabled and can be fired.

In this framework, transitions are used to present instantiated operators of the plans. Places are used to present In, Out, Pre, E of instantiated operators. Control constructs are showed via combining places and transitions. Instantiated operators, AND Split/Join and OR Split/Join represented by Petri Net are illustrated in Fig. 4. In Fig. 4, places do not always refer to Pre, In, E or Out and moreover transitions do not always mean instantiated operators. The combination of the two types of nodes can describe AND Split/Join and OR Split/Join constructs. This kind of description is formal and easier for computer processing.

According to Definitions 5–7, combinable plans might be combined by AND Split/Join and OR Split/Join constructs. The algorithm is illustrated in Fig. 5.

In the algorithm, the termination condition is that all plans in A are maximally extended using input plans, so the plans obtained from the algorithm of Fig. 5 are called extended plans. Classifying R_k into different path sets is the key to the algorithm. The paths in the same set have different starts and ends. However, the precedents of the starts of different paths in the same set must be same and the successors of the ends of the paths are identical too. Two important points should be highlighted. The first is that according to Definition 7, the difference in starts and ends is related to the difference in preconditions of the starts and effects of the ends respectively. The second is that the extended plans might have paths that are never executed. This problem cannot affect the effectiveness of the extended plans, because each path between OR Split and OR Join is associated with a probability P, which will be introduced later.

Definition 8 (*Composite Service*)**.** In an extended plan, if each instantiated operator o is substituted by a service instance in the o.SC, then the result is a composite service, denoted as CS.

Taking advantage of services' functional attributes, Planning Module can produce referred composite plans that play as general patterns. In the following sections, we focus on investigating how to select qualified services to implement the composite plans, i.e. producing CS, based on their non-functional attributes' evaluation.

3.2. QoS rules repository

QoS rules repository is used to store various QoS rules about different QoS criteria. There are QoS criteria in the framework, including five issues of time, availability, successful execution rate, reputation and price of a service.

```
Input: plans (O_i, R_i)
Output: extended plans (O_k, R_k)
Procedure:
Begin
Put all plans into an array A;
While
  If all plans in A have OR and AND constructs
    Then return A;
  EndIf
  get the first one (O_i, R_i) in A;
  If exists (O_j, R_j), and (O_i, R_i) and (O_j, R_j) are combinable
    Then R_k = R_i ∪ R_j;
         classify paths in R_k into different sets;
         Foreach set found above
           insert OR split/Join constructs into R_k;
         EndForeach
         O_k = O_i ∪ O_j ;
         delete (O_i, R_i) and (O_j, R_j);
          insert (O_k, R_k) into the end of A;
  EndIf
  If no (O_j, R_j) exists
     If plan (O_i, R_i) only has OR constructs or no OR construct
         add AND Split/Join into (O_i, R_i);
         put (O_i, R_i) to the end of A;
      Else put (O_i, R_i) to the end of A;
     EndIf
EndIf
EndWhile
End
```

Fig. 5. Extended plan formation algorithm.

(1) *Time issue.*

Any service consumes some time for its execution, which is called duration. Some services could start immediately after the preceding ones finish. While others may have a little delay because its start time may not be equal to the preceding one's finish time, which is the general case. According to the structures of extended plans, three temporal relationships between two services are very important to analyze time characteristics about a composite service. Given a service s, the start time, finish time and duration are all expected values.

Definition 9 (*Meet, Before and After*)**.**

$\text{Meet}(s_1, s_2) \Leftrightarrow s_1.\text{finish} = s_2.\text{start};$

$\text{Before}(s_1, s_2) \Leftrightarrow s_1.\text{finish} < s_2.\text{start};$

$\text{After}(s_1, s_2) \Leftrightarrow \text{Before}(s_2, s_1);$

where s_1, s_2 denote services.

According to Definition 9, it is not difficult to obtain the following QoS rules about time.

(1) o_i happens before $o_j \Longrightarrow$ Meet (o_i.SC.s_i, o_j.SC.s_j) $\vee$Before (o_i.SC.s_i, o_j.SC.s_j);
(2) o_i happens before $o_j \wedge$ Undefined (o_j.SC.s_j.start) $\wedge$ Undefined (o_j.SC.s_j.finish) $\Longrightarrow$ Meet (o_i.SC.s_i, o_j.SC.s_j);
(3) o_i.SC.s_i.finish = o_i.SC.s_i.start + o_i.SC.s_i.duration;
(4) Undefined (o_i.SC.s_i.start) $\wedge$ Undefined (o_i.SC.s_i.finish) $\wedge$ Undefined (o_i.SC.s_i.duration) $\equiv \perp$;
(5) Undefined (o_i.SC.s_i.duration) $\Longrightarrow$ Defined (o_i.SC.s_i.start) $\wedge$ Defined (o_i.SC.s_i.finish).

One-arity predicate Undefined means that the corresponding expected time of a service is not defined or indicated by its service provider. If a service's start time is undefined, it can start on demand, which indicates its finish time. This is the meaning of rule 2. Rule 3 is an identity equation about a service's start time, finish time and duration. Rules 4 and 5 together show that any service should at least have or indicate its duration.

For OR Split/Join structures, only one successor will be chosen to execute. Moreover, through historical data analysis, the probabilities that which successors will be chosen in later execution can be obtained (i.e. p_i for each successor i). Let OR.duration denotes the expected duration from OR Split to the corresponding OR Join and OR_i.duration the ith branch's duration, then rule 6 is obtained.

(6) $\text{OR.duration} = \sum_i p_i \times \text{OR}_i.\text{duration}_i$.

Sometimes a user may put forward start time and finish time constraints on a single service. These requirements can be denoted as $[\text{Start}_{req1}, \text{Start}_{req2}]$ and $[\text{Finish}_{req1}, \text{Finish}_{req2}]$. And, the following two rules are obvious.

(7) $\forall o_i \in O,\ o_i.SC.s_i.\text{start} \in [\text{Start}_{req1}, \text{Start}_{req2}]$
(8) $\forall o_i \in O,\ o_i.SC.s_i.\text{finish} \in [\text{Fin}_{req1}, \text{Fin}_{req2}]$.

Definition 10 (*Start Set and End Set of a Plan*)**.** For a plan (O, R):
$\forall o_i \in O$, if $\nexists o_j \in O$, such that o_j happens before o_i, then the o_i belongs to Start Set (SS);
$\forall o_i \in O$, if $\nexists o_j \in O$, such that o_i happens before o_j, then the o_i belongs to End Set (ES).

According to Definition 10, in a CS, for each $o_i \in SS$ and its corresponding s_i, $\min\{SS.o_i.SC.s_i.\text{start}\}$ exists. And for all $o_j \in ES$, $\max\{ES.o_j.SC.s_j.\text{finish}\}$ exists.

(9) $\max\{ES.o_j.SC.s_j.\text{finish}\} - \min\{SS.o_i.SC.s_i.\text{start}\} \leq \text{Dua}_{req}$.

Dua_{req} is a user's requirement about the duration of a composite service. And let CS.duration denote the expression of $\max\{ES.o_j.SC.s_j.\text{finish}\} - \min\{SS.o_i.SC.s_i.\text{start}\}$.

(2) *Availability issue*.

The availability $\text{Avi}(s_i)$ of a service s_i is the probability that the service is accessible, ranging from 0 to 1. If all preconditions and inputs are available when a service is to be executed, then the service must be available. It is assumed that inputs of a service must be available if previous ones finish successfully, and cannot be available if previous ones fail. Because of uncertainties in preconditions of a service, the availability of a service is also uncertain. Preconditions may be interdependent. However, it is assumed that no precondition will depend on itself. Thus, a DAG (Directed Acyclic Graph) can be used to present the dependencies. If a precondition i depends on another one j, then an arc from i to j is added. This dependence has the probability of pa_{ij}. Then, the availability pa_i of a precondition is computed as:

$$pa_i = \sum_j pa_{ij} \times pa_j.$$

And the availability $\text{Avi}(s_i)$ of a service is computed as:

$$\text{Avi}\ (s_i) = \prod_j pa_j.$$

OR structure is dealt with as a whole, and its availability is computed by the following formula:

$$\text{Avi(OR)} = \sum p_i \times \text{Avi}\ (\text{OR.CS}_i).$$

OR.CS_i denotes the path from OR Split to the corresponding OR Join whose start service is service i. It considers the path as a composite service CS_i. This formula indicates recursive computations.

Thus, for any composite service CS in terms of a plan, the following formula can be applied for availability aggregation about CS.

$$\text{Avi(CS)} = \min(\text{Avi}(OR_k), \text{Avi}_{oi.SC.si \notin OR}(o_i.SC.s_i));$$

The rule about Avi(CS) is

(10) $\text{Avi(CS)} \geq \text{Avi}_{req}$

where Avi_{req} is a user's requirement about availability.

(3) *Issue of successful execution rate*.

The successful execution rate $\text{Suc}(s_i)$ of a service s_i is the probability that a request is correctly responded (i.e., the operation is completed and a message indicating that the execution has been successfully completed is received by service requester), ranging from 0 to 1. The value of the successful execution rate is computed from historical data. If a service is successfully executed, the inputs of successors must be available. Because successful execution rate is only related to service's own environment (including hardware and/or software), it is reasonable to assume that each service's successful execution rate is independent on one another. OR structure is treated as a whole again, and its successful execution rate is computed using the following formula:

$$\text{Suc(OR)} = \sum_i p_i \times \text{Suc}(\text{OR.CS}_i).$$

Thus, for any composite service CS in terms of a plan, the following formula can be applied for availability aggregation about CS.

$$\text{Suc(CS)} = \prod_{\text{OR}} \text{Suc(OR)} \times \prod_{o_i.SC.s_i \in CS-OR} \text{Suc}(o_i.SC.s_i);$$

and the corresponding rule is:

(11) $\text{Suc(CS)} \geq \text{Suc}_{req;}$

where Suc_{req} is a user's requirement about the successful execution rate.

(4) *Reputation issue*

The reputation $\text{Rep}(s_i)$ of a service s_i is a measure of its trustworthiness. The reputation Rep(CS) of a composite service is the average of the reputations of the services that participate in CS. OR structure is treated as a whole, and its Reputation is computed using the following formula:

$$\text{Rep(OR)} = \sum_i p_i \times \text{Rep}(\text{OR.CS}_i).$$

Thus, for any composite service CS in terms of a plan, the following formula can be applied for availability aggregation about CS.

$$\text{Rep(CS)} = 1/|\text{CS.O}'| \times \left(\sum_{\text{OR}} \text{Rep(OR)} + \sum_{o_i.SC.s_i \in CS-OR} \text{Rep}\ (o_i.SC.s_i) \right);$$

and the corresponding rule is:

(12) $\text{Rep(CS)} \geq \text{Rep}_{req}$;

where Rep_{req} is a user's requirement about the reputation.

(5) *Price issue*

Given a service s_i, the price $\text{Pri}(s_i)$ of a service is the fee that a service requester has to pay for the service, usually greater than 0. Analogous analysis yields:

(13) $\text{Pri(CS)} \leq \text{Pri}_{req}$;

where Pri_{req} is a user's requirement about the price and the computation of Pri(CS) is as the following:

$$\text{Pri(CS)} = \sum_{\text{OR}} \text{Pri(OR)} + \sum_{o_i.SC.s_i \in CS-OR} \text{Pri}(o_i.SC.s_i);$$

where $\text{Pri(OR)} = \sum_i p_i \times \text{Pri}(\text{OR.CS}_i)$.

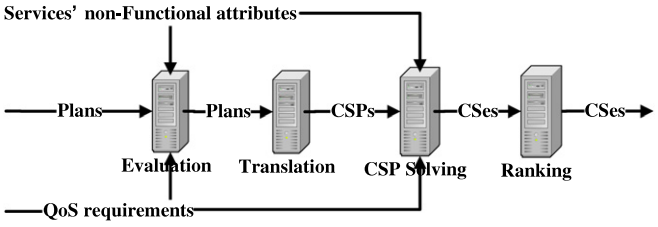

Fig. 6. CSP Module workflow.

3.3. CSP Solving Module

Inputs of CSP Solving Module are extended plans described in Section 3.1 produced by Planning Module, end user's QoS requirements and services' non-functional attributes. The extended plans only show how to take advantage of functional attributes of services to meet user's functional requirements. But whether these plans can satisfy users' QoS requirements such as time is not considered. Additionally, the plans are abstract and are not to be executed directly.

Definition 11 (*CSP*)**.** CSP is a constraint satisfaction problem which is defined as a triple ⟨V, D, C⟩, where

(1) V is a set of variables $\{v_1, v_2, v_3, \ldots, v_n\}$.
(2) domain (v_i) denotes the set of allowable values the variable v_i can be assigned with.
(3) D is a set of domains $\{D_1, D_2, \ldots, D_n\}$ such that D_i = domain (v_i)
(4) C is a set of constraints on the legal values for variables in V.

CSP Solving Module is constituted by several steps to translate extended plans into CSP's and to solve them to get possible composite services. Fig. 6 gives an overview of the module's workflow.

Extended plans are firstly transferred to Evaluation Module to foresee whether these plans can satisfy a user's QoS requirements in terms of existing service instances' extreme non-functional attributes' values. If not available, the plan will be discarded and will not be processed further to avoid meaningless translation and service search. Else, they will be transferred to Translation Module to convert the plans to several CSP's according to QoS Rules introduced in the Section 3.2. After CSP's are obtained, CSP search algorithm can be used to find appropriate service instances to fill in the instantiated operators in the plans. After that, several composite services relevant to a plan may be found. Because they are all satisfactory, ranking is needed before submitted to the end user for selection.

3.3.1. Evaluation

Evaluation aims to estimate the availability of an extended plan. Because of CWA (Closed World Assumption) [15], for each non-functional attribute, its minimum and maximum of a service class can be found in the service library. These extreme values of different non-functional attributes can be used for evaluation. If these extremes can satisfy a user's requirements, the plan may be available. If not, it is just discarded. To some extent, this method can reduce unnecessary further processing and searching.

Because different non-functional attributes have different comparability, some of them are the greater, the better while others are the less, the better. All these variations are not taken into account in detail in evaluation to reduce complexity.

The algorithm is shown in Fig. 7. The input of the algorithm is an extended plan denoted as set O and the corresponding Happens Before Relation on O. In order to be differentiated from real start and finish time about a service in the service information library, temporary start time and finish time are denoted as temStart and temFinish for convenience. If the procedure returns true, then the plan can be used in the next step.

```
Inputs: set O, happens before relation on O
Output: true or false
Procedure:
Begin
  Foreach oi in O
    If no oj happens before oi
      Then oi.temStart = 0;
           oi.temFinish = min(oi.SCi.si.duration);
           put ok where only oi happens before ok in queue Q;
           delete oi and the corresponding happens before;
    EndIf
  EndForeach
  While Q ≠ ∅ do
    Foreach oi in Q
      oi.temStart = max (oj.temFinish)
                    where oj happens before oi;
      oi.temFinish = oi.temStart + min(oi.SCi.si.duration);
      put only ok where oi happens before ok in queue Q;
      delete oi and the corresponding happens before;
    EndForeach
  EndWhile
  If  max(oi.temfinish) > Dureq or
      Πoi∈O max(Avi(oi.SC.si)) < Avireq or
      Σoi∈O min(Pri(oi.SC.si)) > Prireq or
      1/|CS.O|Σoi∈O max(Rep(oi.SC.si)) < Repreq or
      Πoi∈O max(Suc(oi.SC.si)) < Sucreq
      Then return false;
  Else return true;
  Endif
End
```

Fig. 7. Evaluation algorithm.

3.3.2. Translation

Each instantiated operator o_i can be mapped to a variable x_i in the CSP, so that all service instances inherited from the same service class become the domain of variable x_i. There are three key points used recursively in the mapping from QoS requirements of users to CSP constraints about composite services.

Firstly, for all paths that go through a pair of AND Split/Join structures without OR Split/Join structures, the computations of aggregated QoS is according to happens before relation between any two instantiated operators.

Secondly, the aggregated QoS for all paths that begin with the same OR Split and end with the same OR Join is computed according to formulae of different QoS criteria for OR structures.

Thirdly, if the aggregated QoS between OR Split/Join has been computed, the services between OR Split and OR Join can be reduced to a single service that has the same values of non-functional attributes as the QoS between OR Split/Join.

3.3.3. CSP solving

Definition 12 (*Conflict*)**.** There are two functions called $f(x_1, x_2, \ldots, x_n)$ and $g(z_1, z_2, \ldots, z_m)$ where $n, m \geq 0$ and a relation predicate P on them, denoted as $P(f(x_1, x_2, \ldots, x_n), g(z_1, z_2, \ldots, z_m))$. A substitute $u = [u_1, u_2, \ldots, u_{n+m}]$ is a conflict iff $P(f(u_1, u_2, \ldots, u_n), g(u_{n+1}, u_{n+2}, \ldots, u_{n+m})) =$ false, where u_i is a variable having the same meaning as x_i (or z_i) or a constant whose value belongs to the domain of the corresponding variable.

In Definition 12, $P(f(u_1, u_2, \ldots, u_n), g(u_{n+1}, u_{n+2}, \ldots, u_{n+m})) =$ false means that according to the substitution, the predicate is definitely proved to be false. According to Definition 12, any CSP constraint from 3.3.2 can be interpreted as two functions and a relation predicate on them. Conflict can be judged according to the substitute and the truth of the predicate. When searching solutions of CSP, the algorithm will repeatedly evaluate the truth value of

```
Input: CSP
Output: all satisfactory composite services
Procedure:
Begin
  for i = 1 to n
    Select a candidate s_i for o_i with no conflict
    if i = 1 and no candidates,
      return;
    elseif no candidates
      backtrack;
    elseif i = n
      for each candidate s_n for o_n
      output {s_1, s_2, ..., s_n}
    endif
  endfor
End
```

Fig. 8. Basic CSP search algorithm.

```
Input: CSP
Output: satisfactory composite services
Procedure:
Begin
  find one satisfactory solution;
  put it into a queue Q;
  establish a set S;
  While Q is not empty
    take one solution CS from Q and delete it from Q;
    find all new satisfactory solutions around Q within SLL;
    insert them into S;
    insert them into the Q;
  EndWhile
  output S;
End
```

Fig. 9. Improved CSP search algorithm.

each CSP constraint. The main idea of the detailed algorithm is illustrated in Fig. 8. The algorithm of Fig. 8 is called Basic CSP search algorithm, because it has high complexity but finds all satisfactory solutions.

Because of Basic CSP search algorithm's high complexity, it cannot be used in practical environment. However, it provides an effective way to compare different CSP search algorithms designed for finding as many satisfactory solutions as possible and having a low complexity.

One idea is that if one satisfactory solution is found, it can be used to find as many satisfactory solutions as possible, which leads to Definition 13.

Definition 13 (*Step Length*)**.** There are two substitute $u = [u_1, u_2, \ldots, u_{n+m}]$ and $u' = [u'_1, u'_2, \ldots, u'_{n+m}]$, the step length between u and u' is the number of different values between u and u', denoted as $SL(u, u')$.

The algorithm of improved CSP search algorithm is illustrated in Fig. 9. First, find one satisfactory solution. If it exists, then all satisfactory solutions around it within the Step Length Limit (SLL) should be found. These found satisfactory solutions will be inserted into a queue Q. After that, using these found satisfactory solutions, the algorithm continues finding new satisfactory solutions around each of them within step length limit, inserting them into queue Q. This process will not stop until no new satisfactory solution is found. This algorithm can effectively limit search scope.

3.3.4. Ranking

Each pair of OR Split/Join can be replaced equivalently by an imaginary service which has duration of OR.duration. OR.duration is greater than m with the probability of p. m is a critical value that makes the afterward services expired or the composite service's certain QoS criterion unsatisfactory in terms of user's requirements. Component services also have such p's to indicate the probabilities of serious deviations.

Definition 14 (*Exceptional Probability*)**.** The maximum of p's is called the Exceptional Probability of a composite service.

Rank composite services according to ascending Exceptional Probabilities before they are submitted to the end user. It is assumed here that users prefer composite services of low exceptional probabilities. It is reasonable, but, of course, a user can choose any satisfactory composite service via interface.

4. Case study

Tudo Company plans to transport two equipments denoted by package1 and package2 from Nanjing to Shanghai. There are two classes of trunks available. Trunk1 can carry at least one package. It can carry not only package1, but also package2. Trunk2 can carry at most one which must be package2. The company needs stevedores to load and unload equipments. All these services are provided by the other companies. These companies advertised their services on the web. The information about these services are collected and stored in the local service information library. The local service information library is updated periodically in order to keep the information consistent with the reality. Tudo Company wants the two equipments to arrive in Shanghai in 1.5 days. The total price should not exceed 10 000 dollars. All other indices should be greater than 0.9.

After all above requirements are submitted to the framework, they are first processed by Planning Module. The functional requirements are described as:

(:unordered (package-transfer package1 Nanjing Shanghai)

(package-transfer package2 Nanjing Shanghai)).

This kind of expression is a specialized language for SHOP2 [15]. There are 78 plans produced. One plan is showed as:

(!load trunk2 package2) (!move trunk2 Nanjing Shanghai)

(!unload trunk2 package2) (!load trunk1 package1)

(!move trunk1 Nanjing Shanghai) (!unload trunk1 package1).

Other plans have similar expressions. Some of them have common instantiated operators which are showed in different orders. This is the reason why we want to combine them using various algorithms.

All plans are covered by 9 instantiated operators listed in Table 1. O3(1) and O3(2) in Table 1 are distinguished by their preconditions and effects. Inputs and Outputs are not considered for simplicity. Effects also have preconditions which are split by "/". The effects that have Minus "−" mean that the condition is not indicated by the environments because of the execution of the service. The preconditions and effects involved are all numerical and listed in Table 2.

The plans constructed by instantiated operators can be combined. For dependency analysis, the algorithm of Figs. 2 and 3 can be used. For further combination, the algorithm of Fig. 5 can be used. After that, only one plan is obtained. The plan contains AND/OR constructs and is described by Petri Net. The extended plan is depicted in Fig. 10.

According to Fig. 4, it is easy to see And Split, And Join, OR Split and OR Join in Fig. 10. 0.7 and 0.3 are the probabilities with which the corresponding branches will be chosen because of uncertainty of the availability of trunks. For simplicity, preconditions and effects are not depicted in Fig. 10. Transitions with labels denote service classes.

Table 1
Instantiated operators.

Num	Functionality	Preconditions	Effects
O1	Load(Trunk1, Package1, Nanjing)	1, 5, 9, 12, 17, 18	14, −17
O2	Load(Trunk1, Package2, Nanjing)	1, 7, 9, 12, 19, 20	15, −19
O3(1)	Move(Truck1, Nanjing, Shanghai)	1, 11	(14, 15)/(6, 8, −5, −7),
O3(2)	Move(Truck1, Nanjing, Shanghai)	1, 11	(14)/(6, −5)
O4	Unload(Trunk1, Package1, Shanghai)	2, 6, 10, 14	17, −14
O5	Load(Trunk2, Package2, Nanjing)	3, 7, 9, 13, 19, 20	16, −20
O6	Move(Trunk2, Nanjing, Shanghai, Package2)	3, 11	4, −3
O7	Unload(Trunk2, Package2, Shanghai)	4, 8, 10, 16	20, −16
O8	Unload(Trunk1, Package2, Shanghai)	2, 8, 10, 15	19, −15

Table 2
Preconditions and effects.

1	At(Trunk1, Nanjing)	11	NoStorm(Road)
2	At(Trunk1, Shanghai)	12	RoomGE(Trunk1, 1)
3	At(Trunk2, Nanjing)	13	RoomE(Trunk2, 1)
4	At(Trunk2, Shanghai)	14	On(Package1, trunk1)
5	At(Package1, Nanjing)	15	On(Package2, trunk1)
6	At(Package1, Shanghai)	16	On(Package2, trunk2)
7	At(Package2, Nanjing)	17	NotOn(Package1, trunk1)
8	At(Package2, Shanghai)	18	NotOn(Package1, trunk2)
9	NoStorm(Nanjing)	19	NotOn(Package2, trunk1)
10	NoStorm(Shanghai)	20	NotOn(Package2, trunk2)

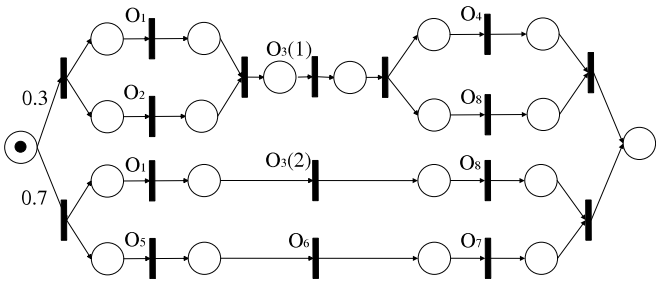

Fig. 10. Extended transportation plan.

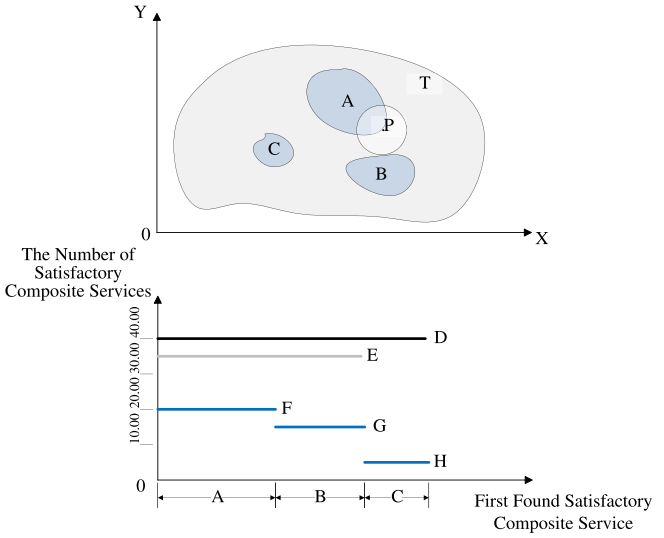

Fig. 11. CSP search analysis.

It is assumed that each instantiated operator o_i corresponds to a variable x_i. Then the constraints in the CSP are listed in Table 3. The formulae in the left side of inequalities are inferred from the formulae in Section 3.

According to CSP constraints listed in Table 3 and the search algorithms of Figs. 8 or 9, it is apparent to find several satisfactory solutions. One of them is showed in Table 4.

The availabilities of services are related to their preconditions, and these preconditions may also be related. In all the conditions listed in Table 2, conditions 9–11 which indicate the conditions of that weather have uncertainty. However, the others must be true when the precedent services are successfully executed. Conditions 9–11 are also related. It is assumed that the probability of noStorm(Nanjing) is 0.99, then P(noStorm(Road)|noStorm(Nanjing)) is 0.99. So does the conditional probability of noStorm(Shanghai) when the weather in Nanjing and on road are not stormy. According to these probabilities, it is easy to obtain Avi's in Table 4.

Finally, the maximum probability of unavailability of services is 0.03. If the durations of services are definite, then the exceptional probability of the composite service is 0.03.

Fig. 11 analyses the difference between basic CSP search algorithm and improved CSP search algorithm used in this example of Tudo Company. There are two kinds of coordinates included in Fig. 11, *X*–*Y* coordinates and "First Found Satisfactory Composite Service"—"The number of Satisfactory Composite Services" coordinates abbreviated as F–T coordinates.

X–*Y* coordinates roughly display the distribution of satisfactory composite services. All possible composite services are in the area of T. All satisfactory composite services scatter among areas of A, B and C. Step length between neighboring composite services in these areas is no more than 1. Step length between A and B is at least 2, and that between A and C or between B and C is 3. Point P represents a satisfactory composite service at the edge of A. If step length is set to 2, then from point P, at least one satisfactory composite service situated in area of B can be reached. If step length is set less than 2, point P can only reach satisfactory composite services in area of A.

F–T coordinates describe how many satisfactory composite services can be found from a first found satisfactory composite service located in area A, B or C indicated by F coordinate. The number is showed by T coordinate. If basic CSP search algorithm is applied, from any first found composite service, all 40 satisfactory composite services depicted by line D can be found. However, it must search the area of T to find all. Contrarily, if improved CSP search algorithm is applied and the step length is set to 1, then from a satisfactory composite service, 20 satisfactory composite services can be found if it is located in area A depicted by line F, 15 if area B by line G and 5 if area C by line H. If step length is set to 2, from point P, area B can be reached from A and vice versa. Consequently, if the first found composite service is located in area A or B, then 35 satisfactory composite services can be found described by line E, but unfortunately, only 5 satisfactory composite service can be found if the first one is located in area C. However, if step length is set to 3, all satisfactory composite services can be found, but the search area is much less than that of basic CSP search algorithm. So, the effectiveness and efficiency of improved search algorithm depend on the step length chosen and the distribution of satisfactory composite services.

Table 3
CSP constraints.

Pri	$0.3 \times (x_1.\text{Pri} + x_2.\text{Pri} + x_{31}.\text{Pri} + x_4.\text{Pri} + x_8.\text{Pri}) + 0.7 \times (x_1.\text{Pri} + x_5.\text{Pri} + x_{32}.\text{Pri} + x_6.\text{Pri} + x_8.\text{Pri} + x_7.\text{Pri}) \leq 10\,000$;
Suc	$0.3 \times (x_1.\text{Suc} \times x_2.\text{Suc} \times x_{31}.\text{Suc} \times x_4.\text{Suc} \times x_8.\text{Suc}) + 0.7 \times (x_1.\text{Suc} \times x_5.\text{Suc} \times x_{32}.\text{Suc} \times x_6.\text{Suc} \times x_8.\text{Suc} \times x_7.\text{Suc}) \geq 0.9$
Avi	$0.3 \times \min(x_1.\text{Avi}, x_2.\text{Avi}, x_{31}.\text{Avi}, x_4.\text{Avi}, x_8.\text{Avi}) + 0.7 \times \min(x_1.\text{Avi}, x_5.\text{Avi}, x_{32}.\text{Avi}, x_6.\text{Avi}, x_8.\text{Avi}, x_7.\text{Avi}) \geq 0.9$
Rep	$0.3 \times (x_1.\text{Rep} + x_2.\text{Rep} + x_{31}.\text{Rep} + x_4.\text{Rep} + x_8.\text{Rep})/5 + 0.7 \times (x_1.\text{Rep} + x_5.\text{Rep} + x_{32}.\text{Rep} + x_6.\text{Rep} + x_8.\text{Rep} + x_7.\text{Rep})/6 \geq 0.9$
Dua	$0.3 \times (\max(\text{OR.SC}_i.s_i.\text{finish}) - \min(\text{OR.SC}_i.s_i.\text{finish})) + 0.7 \times (\max(\text{OR.SC}_j.s_j.\text{finish}) - \min(\text{OR.SC}_j.s_j.\text{finish})) \leq 1.5$ days;$i = 1, 2, 31, 4, 8$; $j = 1, 5, 32, 6, 8, 7$;
All happens before relation constraints	

Table 4
A satisfactory composite service.

Operator	Provider	Pri	Suc	Rep	Avi	Dua (h)
O1	1	500	1	0.90	0.99	8
O2	2	500	1	0.90	0.99	8
O3(1)	3	3000	1	0.98	0.98	5
O3(2)	4	3000	1	0.98	0.98	5
O4	5	600	1	0.95	0.97	8
O5	6	700	1	0.95	0.99	10
O6	7	4000	1	0.98	0.98	4
O7	8	900	1	0.90	0.97	10
O8	9	600	1	0.95	0.97	8

5. Related work and comparison

Some efforts are in progress to establish QoS ontology. DAML-QoS [2] is an ontology for QoS using DAML+OIL, a predecessor to OWL-S. It is composed of three aspects including QoS profile, QoS property definition and metrics. It is now proceeding to OWL-S. It is easy to include links to OWL-S to describe non-functional attributes about Web services. But it is so limited in description ability in terms of absent vocabulary. QoSOnt [10] takes a layered method to establish a QoS Ontology. It also contains links to OWL-S and concentrates upon the definition of metrics and requirements matching, so it has more vocabulary and can identify the correct semantic for matching. In order to establish such ontology, QoSOnt depends on appropriate OWL tools. After that, a QoS ontology language for web services has been proposed in [9]. The QoS ontology language provides a standard and generic model for arbitrary QoS criteria. It is a flexible and inter-operable scheme to formally describe arbitrary QoS parameters. Varieties of QoS Ontologies are analyzed and an idea of a unified ontology is proposed in [11].

AI Planning is investigated in service composition. It is divided into two types: total-order and partial-order planning. Two prominent partial-order planners are SHOP2 [15] and XPlan [13] which have been studied in the field of service composition. In [12], a sound and complete algorithm is provided to translate OWL-S service descriptions into a SHOP2 domain in order to automate service composition. This approach proves to be effective. Moreover, in [13] another OWL-S service composition planner, OWLS-Xplan, is presented combining OWL-S and XPlan. Because both OWLS-XPlan and SHOP2 are based on the Closed World Assumption and use PDDL [4] for problem and domain description, HTN-DL [21] is developed, combining description logics (DLs) and SHOP2 in order to make the planner work in the Open World Assumption.

In order to make planning adapted to Open World Assumption, another way is followed in [22,23]. In [22], an action formalism based on DLs is proposed and the main advantage of such combination is that it is more expressive than the decidable fragments of situation calculus but still have efficient reasoning. In [23], this formalism is used to solve the planning problem. These methods have high complexity in inference and lack of domain knowledge. In [24], Service Dynamic Description Logic (SDDL) is proposed to represent and model the dynamic aspect of web services. But they are seldom put into practice.

Despite of these efforts, they are all partial-order method that produces total-order plans. QoS analysis meets difficulties in the face of total-order plans. However, combining non-functional requirements into planning repository faces some difficulties and complexities. In the face of those difficulties, we propose a two-step workflow framework for service composition. In our framework, SHOP2 produces plans that are abstract. Each instantiated operator corresponds to a service class and only includes functional information. Plans are combined to produce extended plans described by Petri Net. So it is sufficient to describe and analyze extended plans.

CSP-based service composition methods are used in [17–19]. In [17], a framework is proposed, combining SHOP2 with CSP for automatic service composition. After planning, the planner will produce a set of CSP's for further selection of appropriate service instances. In [18] a CSP solver is developed for CSP Solving of [17]. The solver depends on a CSP Ontology, so it has several advantages such as flexibility, generalization. But this method only put constraints into SHOP2 domain using special constructs in order to get CSP Set after planning, so some evaluation about QoS such as time cannot be solved well in some circumstances because of the internal characteristics of SHOP2. The difficulties are detailed in [15]. In [19], only CSP-based method is used directly for service composition. But it mixed functional and non-functional attributes about services and lacked effective domain knowledge, so generally speaking, the complexity increases. In our framework, non-functional attributes are separated from functional attributes. Plans can be produced more quickly using SHOP2 with domain knowledge than using CSP. And CSP search is restricted to several service classes. After all satisfactory composite services are found, they will be ranked according to exceptional probability. According to [1] the execution of services could be promoted in the form of workflow paradigms, so how to control execution of composite services to reduce exception is not discussed here. One method can be related to scheduling and runtime adaptation of workflow in [25].

To select appropriate service instances, linear programming is used in [26]. It is based on critical path to reduce complexity. In practice, constraints are often non-linear, especially when the non-functional attributes of service instances are related. In our QoS rules, various criteria are calculated recursively (non-linear). Preconditions are related. Without critical path, AND/OR structure is considered under a unified framework.

6. Conclusion

This paper presented a workflow framework for intelligent service composition. The framework consists of two processing modules: Planning Module and CSP Solving Module. Taking advantage of services' functional attributes, Planning Module aims at producing referred composite plans that play as general patterns. CSP Solving Module focuses on selecting qualified services to implement the composite plans, based on their non-functional attributes' evaluation. Although this framework is put into effect in a real-life application, some technical issues such as how to set up the indispensable repositories should be discussed in our future work.

Acknowledgement

This paper is partly supported by the National Science Foundation of China under Grant Nos. 60721002, 60673017 and 60736015, Jiangsu Provincial NSF Project under Grants BK2008017.

References

[1] G. Alonso, F. Casati, H. Kuno, Web Services: Concepts, Architecture and Applications, Springer-Verlag, 2004.
[2] C. Zhou, L.T. Chia, B.S. Lee, Qos measurement issues with DAML-QoS ontology, in: Proceedings of the IEEE International Conference on e-Business Engineering, October 2005, pp. 395–402.
[3] B. Srivastava, J. Koehler, Web service composition-current solutions and open problems, in: Proceedings of ICAPS'03 Workshop on Planning for Web Services, Trento, Italy, 2003, pp. 28–35.
[4] M.D. Dermott, PDDL-the planning domain definition language, in: Technical Report, CVC, TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, October 1998.
[5] J. Chen, Y. Yang, Activity completion duration based checkpoint selection for dynamic verification of temporal constraints in grid workflow systems, International Journal of High Performance Computing Applications, Sage 22 (3) (2008) 319–329.
[6] J. Chen, Y. Yang, Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems. ACM Transactions on Software Engineering and Methodology (in press), see http://www.swinflow.org/papers/TOSEM.pdf (accepted 17.06.2009).
[7] S.A. McIlraith, T.C. Son, H.L. Zeng, Semantic web services, IEEE Intelligent Systems 16 (2) (2001) 46–53.
[8] OWL-S: semantic markup for web services http://www.w3.org/Submission/OWL-S/.
[9] I.V. Papaioannou, D.T. Tsesmetzis, I.G. Roussaki, A QoS ontology language for web services. in: Proceedings of the 20th International Conference on Advanced Information Networking and Applications, vol. 1, April 2006, pp. 101–106.
[10] G. Dobson, R. Lock, I. Sommerville, QoSOnt: a QoS ontology for service-centric systems, in: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, Porto, Portugal, August 2005, pp. 80–87.
[11] G. Dobson, A.S. Marcian, Towards unified QoS/SLA ontologies, in: Proceedings of the IEEE Services Computing Workshop, September 2006, pp. 169–174.
[12] E. Sirin, B. Parsia, D. Wu, HTN planning for web service composition using SHOP2, Journal of Web Semantics: Science, Services and Agents on the World Wide Web 1 (4) (2004) 377–396.
[13] M. Klusch, A. Gerber, M. Schmidt, Semantic web service composition planning with OWLS-XPlan, in: Proceedings of the AAAI Fall Symposium on Semantic Web and Agents, AAAI Press, Arlington VA, USA, 2005.
[14] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, in: Hierarchical Task Network Planning, Morgan Kaufmann Publisher, 2004, (Chapter 11).
[15] D. Nau, T.C. Au, O. Ilghami, SHOP2: an HTN planning system, Journal of Artificial Intelligence Research 20 (2003) 379–404.
[16] Stuart J. Russel, Peter Norvig, Artificial Intelligence—A Modern Approach, 2nd ed., Pearson Education Asia Limited, 2006.
[17] I. Paik, D. Maruyama, Automatic web services composition using combining HTN and CSP, in: Proceeding of the 7th IEEE International Conference on Computer and Information Technology, 2007, pp. 206–211.
[18] PDaisuke Maruyama, PIncheon Paik, Mitsuteru Shinozawa, A flexible and dynamic CSP solver for web service composition in the semantic web environment, in: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, 2006, pp. 43–48.
[19] R. Thiagarajan, M. Stumptner, Service composition with consistency-based matchmaking: a CSP-based approach, in: Fifth European Conference on Web Services, November 2007, pp. 23–32.
[20] R. Dvid, H. Alla, Discrete, Continuous, and Hybrid Petri Nets, Springer-Verlag, 2005.
[21] E. Sirin, Combining description logic reasoning with AI planning for composition of web services, Ph.D. Thesis, Department of Computer Science, University of Maryland, 2006.
[22] F. Baader, C. Lutz, M. Milicic, U. Sattler, F. Wolter, Integrating description logics and action formalisms for reasoning about web services, LTCS-Report 05-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.
[23] M. Milicic, Planning in action formalisms based on DLs: first results, in: Proceedings of the 2007 International Workshop on Description Logics, 2007, see http://www.informatik.unitrier.de/~ley/db/conf/dlog/dlog2007.html.
[24] Z.X. Jiang, L. Qian, X. Pen, S.S. Zhu, Dynamic description logic for describing semantic web services. in: Proceedings of the Second International Multi-Symposium on Computer and Computational Sciences, August 2007, pp. 212–219.
[25] M. Wang, R. Kotagiri, J. Chen, Trust-based robust scheduling and runtime adaptation of scientific workflow, concurrency and computation: practice and experience, Wiley 21 (16) (2008) 1982–1998.
[26] L.Z. Zeng, B. Benatallah, H.H. Ngu, QoS-aware middleware for web services composition, IEEE Transaction on Software Engineering 30 (5) (2004) 311–327.

Xudong Song received the M.S. degree in Computer Science and Technology from Nanjing University, Nanjing, China, 2010. He received his B.S. degree in Computer Science and Technology from Xi'an Jiaotong University, Xi'an, China, 2007. Now he is a research member of Intelligent Service Composition Group in the Department of Computer Science and Technology, Nanjing University, China. His research interests include workflow management and intelligent service composition.

Dr. Wanchun Dou received his Ph.D. degree in mechanical and electronic engineering computer from Nanjing University of Science and Technology, China, in 2001. From Apr. 2001 to Dec. 2002, he did his postdoctoral research in the Department of Computer Science and Technology, Nanjing University, China. Now, he is a full professor of the Department of Computer Science and Technology, Nanjing University, China. From Apr. 2005 to Jun. 2005 and from Nov. 2008 to Feb. 2009, he respectively visited the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, as a visiting scholar. Up to now, he has chaired two NSFC projects and published more than 60 research papers in international journals and conferences. His research interests include workflow technologies, cloud computing and service computing technique.

Dr. Jinjun Chen received his Ph.D. degree in Computer Science and Software Engineering from Swinburne University of Technology, Australia, 2007. He received his M.S degree in Communication and Information Systems from Xidian University, Xi'an, China, 1999. He received his B.S. degree in Applied Mathematics from Xidian University, Xi'an, China, 1996. Now he is a Senior Lecturer in the Faculty of Information & Communication Technologies, Swinburne University of Technology. His main research interests focus on cloud computing and service computing technique.