Context Optimization of AI planning for Services Composition

Lirong Qiu*  Zhongzhi Shi* Fen Lin*  
*Key Laboratory of Intelligent Information Processing,  Institute of Computing Technology, Chinese Academy of Sciences, 100080, Beijing, China  
*Graduate School of the Chinese Academy of Sciences, 100039, Beijing, China  
{qiulr, shizz, linf}@ics.ict.ac.cn

Abstract

Web services composition techniques are gaining momentum as the opportunity to establish reusable and versatile interoperability applications. Many researchers propose their composition approach based on planning techniques. We propose our context aware planning method which comprises global planning and local optimization based on context information. The major technical contributions of this paper are: (1) We propose an ontology-based framework for the context-aware composition of Web services. Context model, which are structured based on OWL-S, captures the Service-related, Environment-related, and User-related context and can be used in an unambiguous, machine interpretable form. (2) We propose context-aware plan architecture and thus is more scalability and flexibility for the planning process, and thereby improving the efficiency and precision. (3) We propose a hybrid approach to build a plan corresponding to a context-aware service composition, based on global planning and local optimization, considering both the usability and adoption. We test our approach on a simple, yet realistic example, and the preliminary results demonstrate that our implementation provides a practical solution.

1 Introduction

A Web service is a set of related functionalities identified by URIs that can be advertised, located, and triggered by other applications and humans through the Web [1]. If no single available Web service can satisfy the functionality required by a user, there should be a possibility to combine existing services together to fulfill the request. The problem of Web service composition is the process of selecting, combining and executing existing services to achieve a requestor’s objective, which is a highly complex task. Currently, human beings perform manual Web service composition by reading information provided on service’s Web pages. With the ever increasing number of Web services being made available on the Web, it is already beyond the human ability to analysis them and generate the composition process manually. This has triggered an active area of research and development on Web Service Composition [12].

The purpose of Semantic Web Services [8] is to use semantic specification to automate the discovery, invocation, and composition Web services. The Ontology Web Language for services (OWL-S) is the most direct outcome for describing Web services, semantics of which are defined in terms of Description Logic, to enable automated planning. Given a representation of services in OWL-S, many researchers exploit AI planning techniques for service composition by treating service composition as a planning problem. Ideally, following a users request and a set of Web services described as actions, a planner would find a collection of Web services that achieves the request [13].

However, the size of the data involved in the planning process over Semantic Web will be much bigger than the ones encountered in classical planning problems. Following a user’s request, the number of component services selected in the composite service maybe large, and the number of Web services being considered from which these component services are selected is likely to be even larger [4]. Therefore, the performance and applicability to complex problems of planners are still being debated.

While context information, which is the important element to be considered during selecting and combining services, can increase the effectiveness and acceptance of composition. The use of context in Web service composition has similar incentives to its use in the domain of pervasive computing or mobile services. These incentives include helping reach personalization if there is a need of adapting to user’s preference, filtering the inappropriate services during the service performance and outcome delivery, and thereby improving the usability and adoption. A service composition model that can integrate, and make use of context information to derive the optimal component services of the composite service is still an ongoing research problem.
In this perspective, utilizing a means of context aware, AI planning method to services composition is the motivation and central foundation of our work. The major technical contributions of this work are the following:

- We propose an ontology-based framework for the context-aware composition of Web services. Context model, which are structured based on OWL-S, captures the Service-related, Environment-related, and User-related context and can be used in an unambiguous, machine interpretable form.

- We propose context-aware plan architecture, which is more scalability and flexibility for the planning process, and thereby improving the efficiency and precision.

- We propose a hybrid approach based on global planning and local optimization, considering both the usability and adoption, used to build a plan in both offline and on-line ways.

The remainder of this paper is structured as follows. Section 2 provides extensions to OWL-S as regards support for service composition tasks. In particular, we illustrate our OWL-SC model for describing context information. In Section 3 we present our system architecture and algorithm of global planning and local optimization. And we test our approach on a simple, but realistic example. The related work and conclusions will be given in Section 4 and Section 5 respectively.

2 Extending OWL-S with Context Elements

As opposed to traditional Web applications intended for manual interactions, Web services encapsulate application functionality and information resources, and make them available through interfaces, which can be discovered and used by other applications through the web. Web services need to be described and understood in a formalized way. In this section, We propose an ontology-based method to describe context-aware services.

2.1 Ontology-based Context Modeling

Context is defined as any information that is relevant to the interactions between a user and an environment by Dey et al. [2]. Context modeling defines how context data is structured and maintained, which play a key role in supporting efficient context management.

In realistic composition domain, services are usually grouped as a collection of sub-domains for different goals. Context in each domain shares common concepts that can be modeled using a general context model, while differs significantly in detailed features. This characteristic largely encourages the reuse of general concepts.

Ontology, as a formal and explicit specification of a shared conceptualization, is expected to empower web resources with semantics so that software agents can read these descriptions and reason about how to interact with the resources they describe. Therefore, we propose an approach based on ontology to model context information.

This choice is motivated by several other reasons: (1) Context modeling needs to adopt a flexible structure with explicit concepts and associations to define concepts and relationships between them. (2) Logic reasoning or inference mechanisms are necessary to deduce abstract contextual information. (3) Knowledge sharing and reuse can be achieved between heterogeneous sources. Hence it is possible to adapt using the ontology mechanisms to fit other alternative methods.

2.2 OWL-SC: Extending OWL-S with Context Elements

OWL-S is a set of OWL ontologies supporting the rich description of Web services, thus facilitating the automation of service composition. OWL-S comprises three interrelated subontologies, known as the profile, process model, and grounding. In short, Service Profile ontology describes capabilities of Web services by specifying the input and output types, preconditions and effects, namely IOPE. The Process Model describes how the service works. The Service Grounding specifies the information necessary for service invocation and execution. The OWL-S has an extensible service parameter mechanism which allows the additional description of non-IoPE attributes of services. Therefore, we can extend the OWL-S with context attributes.

There are various categories of context knowledge that are pertinent to both the user and the service. Due to evolving nature of services, completely formalizing all context information is likely to be an in-surmountable task. To ascertain the quality of the composition, three types of context are defined to track the user, the service, and the environment. Figure 1 shows the OWL-S based context model for U-Context, E-Context and W-Context.

U-Context: A user’s context should leverage knowledge about who the user is, where the user is and what the user is doing. Then, we divide the information into two types of contexts: User static context and User dynamic context. User static context specified in a user’s profile and describes information interests and preferences, While user dynamic context defines a user’s location, current activity and task.

W-Context encompasses a number of non-functional properties of services, such as execution price, execution duration, availability, reliability, and reputation. In Short,
W-Context is much like the QoS model. Many of the ideas of QoS modeling proposed to track the service context could be integrated into our approach.

E-Context comprises the information about where the user is, such as the weather, the date, and some kind of surrounding situation.

In the following, we elaborate on representative associated context elements, and on a corresponding ontology of context expressed in OWL. Figure 2 shows a partial definition of specific ontology for U-Context. In this definition, User context consists static elements and dynamic elements. Classes are related by RDFS properties. For example, the Preference context class is a subclass of the context class User-Static-Context, meaning that it is a static type of context.

Also in this figure, we give a restaurant preference as an example. We define some subclass of Restaurant-Preference, such as diet, cuisine, average-cost and meal-time. And define Vegetarian, Kosher as instances of the ontology class diet.

3 Context-based AI planning for Service composition

3.1 System Architecture

The diagram shown in Figure 3 gives an overview of the pivotal components of the system. Context Proxy retrieves context information (defined in the previous section) from the Web and generates context conditions that must hold for this composition request. Context Repository is a component that stores the explicit context representations of users, services and environment based on the definition of OWL-SC ontology. How to sense the current context and deliver it to the application is also a hot research domain. Typical techniques are used in many context-aware applications, which is not the point of this paper. Context Spec. stored predefined rules with associated context conditions, which is the judging rules for filtering actions considering the current states and situation.

The core of the composition framework is the planner, takes user’s request, and uses the domain definition, available context information, to identify a sequence of actions that can be executed in order to achieve the desired request. In the Plan Library, there stored many classified plan schemas (stored as UML statecharts) which can be defined off-line by users or can be learned from the execution. The diagram represents the main motivation behind our method: global planning (search plan library and find a plan template) and local optimization (generate DAG dynamically and select an execution path).

It worth noting that Web services will be grouped according to different domains classified by context attributes. Thus, we can define different plan scenarios to represent typical domain cases. For each scenario, the pertinent context elements is specified which can deduce actions. For example, in the travel scenarios, user’s context should be comprises preference for type of transportation and environment’s context should be comprises the current day of week and time of day and in the shopping scenarios, user can indicate in their respective profiles whether they are willing to receive shopping advertisements.

3.2 Global AI Planning Algorithm

A composite service is specified as a collection of generic service tasks described in terms of service ontologies and combined according to DL-based reasoning. In AI domain, researchers exploited AI planning techniques for automatic service composition by treating service composition as a planning problem, given a representation of services based on OWL-S.

Definition 1 (Planning Domain) A planning domain corresponds to a particular domain to be planned, provided
with states, actions, and functions. Each Planning domain $D$ is described as a tuple:

$$D = (S, A, C, I, \Gamma, \Upsilon)$$

1. $S$ is the set of states.
2. $A$ is the set of actions/services.
3. $C$ is the set of context elements.
4. $I$ is the set of initial states and $I \subseteq S$;
5. $\Gamma$ is the transitional function, $S \times A \rightarrow 2^S$, which associates to each current state $s \in S$ and to each action $a \in A$ the set $\Gamma(s,a) \in S$ of next states;
6. $\Upsilon$ is the context function, $S \times C \rightarrow A$, which associates to a state $s \in S$ and a context $c \in C$, an action $a \in A$ to be executed, $\Upsilon(s,c) \in A$.

A state can be represented as a conjunction of positive literals. For example, $At(Beijing, Zhang) \land Destination(Shanghai)$.

An action $a \in A$ is defined as $(Pre, Eff)$, $Pre \in S$ defines the precondition must hold before the action can be executed, and $Eff \in S$ defines the execution outcome.

There are three kinds of context elements in $C$ defined in the above section. User may define his preference in advance. For example, if the user wants to find a restaurant to eat, and he emphasizes to save time. Then if current state is $Hungry(Zhang)$ and current context is $At(Yayuncun, Zhang)$, then the action finding a restaurants in the vicinity of Yayuncun is selected.

The execution of a plan over a domain can be described as a set of actions to be executed, which, defined by function $\Upsilon$, depend on the current state and on the context. Considering a scenario, the user wants to find a service to rent a vehicle. He defines if the weather is fine, he prefers to flight, else he prefers train. Then we can store the rules in $\Upsilon$.

**Definition 2 (A Planning Task)** A plan task in a planning domain $D$ is defined as a three-tuple $T = (S_1, S_n, D)$, where $S_1 \in I$ denotes the initial states of the planner; $S_n$ denotes the goal state the planning system attempted to reach; $D$ is a domain description.

**Definition 3 (Plan)** A plan in a planning domain $D$ for a planning problem $T = (S_1, S_n, D)$ is defined as a sequence of actions (or services) that will achieve $S_n$ from $S_1$ in $D$. In brief, by taking $T = (S_1, S_n, D)$ as input, the planner will return a plan $P = (a_1, a_2, \ldots, a_n)$.

There are some plan schemas stored in Plan Library classified by different domains. Those plan schemas are represented in UML statechart [3], which have many advantages in composite service specification, such as possessing a formal semantics, easily integration into the Unified Modeling Language and suitable for expressing typical control-flow dependencies.

A statechart is made of states and transitions. States can be simple or composite. Simple states are labeled with a single component service/action. A composite state is a state that, in contrast to a simple state, has a graphical decomposition, which can be OR-states and AND-states. An OR-states are used as a grouping mechanism for modularity purposes whereas AND-state contains several regions intended to be executed concurrently. The Initial State of a statechart
is marked using a solid circle. Final states are drawn using a solid circle with a surrounding circle. A simplified statechart Planning for attending a conference specifying a composite Web service is depicted in Figure 4(a). To simplify the discussion, we initially assume all the startcharts involved in this paper are acyclic.

**Definition 4 (Planning Process in Statechart)** The decomposition of a Planning Task \( T = (S_1, S_n, D) \) of a statechart is a sequence of states \([S_1, S_2, \ldots, S_n]\), such that \( S_1 \) is the initial state, \( S_n \) is the final state, and for every state \( S_i(1 < i < n) \): (1) \( S_i \) is a direct successor of one of the states in \([S_1, \ldots, S_{i-1}]\) and is not a direct successor of any of the states in \([S_{i-1}, \ldots, S_n]\). (2) \( \exists S_j \in [S_1, \ldots, S_{i-1}] \) where \( S_j \) and \( S_i \) belong to OR-states of the statechart. (3) if an initial state of one of the concurrent regions of an AND-state execute, then all the other branches of this AND-state are executed.

Many planning methods already exist and we choose HTN planning method as our global planning algorithm to build statechart. The key to the global planning algorithm is the construction of a plan library. This process is a kind of off-line planning, thus can alleviate the issue of performance efficiency.

### 3.3 Local Optimization based on Context Configuration

If a statechart contains OR-states, it has multiple paths from the initial state to the final state. Each path represents a different plan to complete a compound service execution. Hence, it is possible to execute different path of a statechart by allocating different Web service to the basic states in the path. How to structure and maintain execution paths play a key role in supporting efficient planning. A directed acyclic graph (DAG) can be used to represent a set of actions where the input, output, or execution of one or more actions is dependent on others. By representing actions in DAG, we can use topological searching of a DAG, which gives an allowable (total) order for carrying out the basic actions one at a time.

Therefore we transfer the statechart into an execution plan presented as a Directed Acyclic Graph (DAG), \( G = G(V, E) \), where \( V = \{v_1, \ldots, v_n\} \) is a set of actions and \( E \) is a set of weighted directed edges (arcs) identify the dependencies. For each states \( S_i \) in a statechart, there is a set of candidate actions that can achieve the state \( S_i \). Firstly, we list the detailed definition of the weighted DAG as follows:

**Definition 5 (DAG of Task Accomplishment Path)**
Given a task decomposition statechart \([S_1, S_2, \ldots, S_n]\), the DAG \( G = G(V, E) \), representation of the plan is a graph based as follows: (1) The DAG has at least two nodes Start and Finish. (2) \( a_i = (a_{i1}, a_{i2}, \ldots, a_{it}) \) is the action set for state \( S_i \) in ST and \( E \ni f(a_i) \supseteq S_i \). (3) The DAG has one node for each action \((a_1, a_2, \ldots, a_n)\). (4) The DAG contains an edge from action \( a_i \) to action \( a_j \) iff \( S_j \) is a direct successor of \( S_i \). (5) The edge from \( a_i \) to \( a_j \) is weighed by the context elements \( c \in C \) and \( T(S_i, c) \rightarrow a_i \).

To note that there are some edges have no weighed attributes in the DAG, which demonstrate that the action \( a_i \) can be executed in state \( S_i \) no matter what current context is. Then we give an algorithm to build the DAG dynamically according to the given statechart. Algorithm 1 represents how to find the candidate actions and generate the DAG. The key to the above algorithm 1 is the construction of execution path of plans. The empty plan contains just the Start and Finish actions.

The local optimization process can be guided by following methods: (1) domain control knowledge; (2) Utility function; (3) User input. In this work, we are focusing on the automated optimization for finding the execution path.

In addition, there could be two operations of the DAG execution path: union and intersection. Using these two operations, we can integrate the small-granularity plan into a large granularity plan or divide a big plan problem into several small-granularity components.

**Definition 6 (The union of two DAG path)** Given two execution path represented in DAG, \( G_1 = G_1(V_1, E_1) \) and \( G_2 = G_2(V_2, E_2) \), and the finish node of \( G_1 \) is just the start node of \( G_2 \), then the union of \( G_1 \) and \( G_2 \) is also a DAG execution path, denoted as \( G_1 \cup G_2 \). Let \( G_3 = (V_3, E_3) \) be \( G_1 \cup G_2 \) and \( G_3 \) is created according to the following steps: (1) all the nodes in \( G_1 \) and \( G_2 \) are considered as atomic nodes, \( V_3 = V_1 \cup V_2 \). (2) if a node \( V_c \) is not \( \in V_1 \cap V_2 \), then the DAG \( G_3 \) expanded by \( V_c \) and all the edges that corresponds with \( V_c \). (3) if a node \( V_c \in V_1 \cap V_2 \), then compare the edges \( E_{1c} \) corresponds with \( V_c \) in \( G_1 \) with the edges \( E_{2c} \) corresponds with \( V_c \) in \( G_2 \), then select the optimal edges to expand \( G_3 \).

**Definition 7 (The intersection of two DAG path)**
Given two execution path represented in DAG, \( G_1 = G_1(V_1, E_1) \) and \( G_2 = G_2(V_2, E_2) \), and the finish node of \( G_1 \) is the same as the finish node of \( G_2 \), then \( \{\text{first, finish}\} \neq \emptyset \) then the intersection of \( G_1 \) and \( G_2 \) is also a DAG execution path, denoted as \( G_1 \cap G_2 \). Let \( G_3 = (V_3, E_3) \) be \( G_1 \cap G_2 \) and \( G_3 \) is created according to the following steps:
(1) all the nodes in G1 and G2 are considered as atomic nodes, let E3 = E1 − E2 and V3 be the set that consists of the nodes that are adjoined by every edge in E3.
(2) if a node Vc ∈ V1 ∩ V2, and Vc is not ∈ V3 then adds Vc and the related edges to expand G3.

Algorithm 1 Local Optimization Algorithm

1: Given a statechart ST [S1, S2, . . . , Sn],
2: RestofST ← ST,
3: Generate two nodes Start and Finish
4: Pre(Start) ∈ I ∩ Eff(Start) = S1 and Eff(finish) = ∅
5: Link Start to Finish, stored in DAGCurrentPath, DADBPath
6: while RestofST ̸= ∅ do
7:    CS ← FirstState(RestofST)
8:    if CS is not OR-States then
9:      CA ← GetBestCandidate(CS)
10:     RestofST = RestofST − CS
11:   else
12:      CS′ ← NextOSTate(RestofST)
13:     CA ← GetBestCandidate(CS′)
14:     end if
15:     CA ← SelectBestCandidate(CA, CA′)
16:     link CA to DAGCurrentPath
17: end while
18: return DAGCurrentPath

Procedure GetBestCandidate(CS)
20: CT ← findRelatedContextTypes(c)
21: OT ← orderContextTypes(CT)
22: for all c ∈ OT do
23:    v ← value(c)
24:    a ← getRelevantSelection(T(CS, v))
25:    CA ← CA + a
26: end for
27: end procedure

3.4 Case Study

To illustrate how web service composition can be accomplished by using AI planners coordinated by Context optimization, we give an example of composing services for a researcher to participate in a conference. The detailed scenario is as follows: A researcher in Beijing China wants to participate a conference in Shanghai. He requires a service of “Travel Plan”. In his preference, he defined his transportation preference is train if it’s rain. The composite request can be achieved using our method presented above.

1. Define the domain D is “Plan trip”.
2. According to users preference, update the context repository and context spec. A rule R1 : (S, C) → ai is stored in the context function set Υ, where S is BookTraffic, C is the weather information.

\(\text{(BookTraffic, Sunny)} \rightarrow \text{AirlineBooking}\),
\(\text{(BookTraffic, Cloudy)} \rightarrow \text{TrainBooking}\)

3. Translate the request to a plan problem (S1, Sn, D), which is the initial state, goal state and the domain D.

\(S_1 : \text{At(Beijing, Zhang)}, \text{DepartureDate}(15)\),
\(S_n : \text{Destination(Shanghai)}, \text{TrafficTicket, RentleHotel, RentleCar}\)

4. Search Plan Library to find the task statechart using global planning algorithm. This gives us a consistent plan, represented in UML statechart as shown in Figure 4(a).

\(S_1 : \text{BookTrafficTicket}; S_1 : \text{CalculateArriveDate}; S_2 : \text{RentleHotel}; S_5 : \text{RentleCar}\)

5. Generate DAG Graph using optimization algorithm. Firstly, find the candidate actions and create different paths. There are some candidate services for the selection 4(b). Then search the context function rules in Υ and selecting the best candidate action according the context attributes stored in Context Repository. Next, build the execution path 4(c).

For example, four actions listed as follows:

\(a_{21} : \text{China} − \text{Airline} − \text{Booking}\),
\(a_{22} : \text{Japan} − \text{Asia} − \text{Airline} − \text{Booking}\),
\(a_{23} : \text{China} − \text{Train} − \text{Booking}\),
\(a_{24} : \text{Japan} − \text{Asia} − \text{Ship} − \text{Booking}\)

According to the location context, service \(a_{21}, a_{23}\) is pick out, and according to the rules in Υ(defined Step 2), if acquired current weather is sunny, then \(a_{21}\) is selected, else \(a_{23}\) is selected.

6. Executing candidate services according to the selected path.
3.5 Implementation

In short, the overall process consists of three steps: Firstly, Gets the user’s request and context information. Secondly, the planner searches Plan Library to find the statecharts template to determine subgoals. Then, selects the applicable path according to context elements and generates DAG execution path.

We developed a prototype OntoService used to validate the feasibility and benefits of the proposed approaches. Currently, the system OntoService has been implemented as a platform that provides tools for: (1) defining plan scenarios and represent in UML statechart; (2) building service ontologies and checking its consistency; (3) demonstrating DAG execution path graphically; (4) modeling and reasoning context information based on RDQL.

In addition, we are developing a website applied for Olympic Games 2008 based on OntoService. We have developed application scenarios for different user groups, e.g., a visitor or a journalist, to identify typical use cases and to derive requirements on platform functionality and services to be integrated. And according to different user groups, we put forward different user profile template.

For user context modeling we envisage the approach that is still under development. At the time being we restrict ourselves to the use of time and location information that can be automatically gathered and of a manually maintained information about a user’s mode of activity.

4 Related Work

The issues of context-aware service composition have been widely discussed in pervasive and mobile computing domain [11, 5, 15]. To make good use of the context elements in AI planning system for service composition is practical.

We utilize the idea of global planning algorithms for dynamic composition, proposed by Nahrstedt [10]. Like our approach, the authors proposed a model to compute an initial plan at the beginning and then revise the plan as necessary during the execution. Zeng [4] provided a QoS-driven dynamic service composition model applied to composite services specified in BPEL4WS. In this work, the authors propose statecharts and DAG to represent the execution plans and execution paths. Unlike our approach however, they did not propose a formal algorithm to build the DAG dynamically.

In parallel, various models have been proposed to represent context information. Barkhuus and Dey [2] have defined three types of context-aware applications according to different interactivity: personalization, active context-awareness, and passive context-awareness. Sonia Ben Mokhtar [9] presents an approach for context-aware service composition based on workflow integration in pervasive computing environments. In [6, 7], Zakaria Maamar discusses the way context is used for Web services personalization. To be able to track personalization, three types of context are suggested: User, Service and Resource.

Despite the relatively large related work in service composition domain and context computing domain, few efforts have specifically addressed the topic of Context-based service composition using AI planning technology. An exception is the paper by Vokovic [14]. She discussed a specific context aware composition problem for context aware planning-based Web service composition using SHOP2 planner and the BPELAWS technology. However, The framework does not incorporate the use of a plan library. Hence, a plan has to be recomputed each time a request is received.

5 Conclusion

In this paper, we propose a framework for composing context-aware Web services. We present an OWL-S based context modeling method, which can be extensible easily. We propose a hybrid method for global planning and local optimization. We define a formalized model for representation task decomposition and plan execution path. Then we demonstrate the use of our algorithm for context-based optimization to generate an execution path, transferring plan task statechart into DAG simultaneously. Compare to existing work in this field, our work has some advantages in scalability and flexibility.

The work of this paper is a part of our ongoing research work, which aims to provide an open reusable infrastructure for essential service composition mechanisms. Various experiments and applications have been undergoing in our current research. Future work includes extending learning plan scenarios during planning process based on user’s feedback and considering context modeling and context reasoning based on Description Logics.

Acknowledgements

Our work is supported by the National Science Foundation of China (No.60435010), the national 973 Project (No.2003CB317004) and the Nature Science Foundation of Beijing (No.4052025).

References


