A Fast Algorithm for Web Service Composition Based on Dynamic Description Logic

1Wei Liu, 1,2Yu Yue Du, 1Bao Qi Guo, 1Chun Yan and 1Qiang Xu
1College of Information Science and Engineering,
Shandong University of Science and Technology, Qingdao 266510, China
2The State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 10080, China

Abstract: Dynamic Description Logic (DDL) is an extension of description logic. In dynamic description logic, both services and the service composition can be expressed in formulae, but there is still much room to promote the efficiency. In this study, a fast and effective method for web service composition is proposed for the services which perform in sequence. This method takes advantage of the partial order relation among services, dividing the service composition into two phases: on the first stage, the partial order diagram is generated in the registration server; on the second stage, the fast algorithm is run based on the partial order diagram to return web service composition to meet requests of users. By analysis and verification, the fast algorithm for web service composition presented in this study can be implemented in a linear time complexity, greatly shortening response time of the system.

Key words: Dynamic description logic, service composition, fast algorithm, partial order, web ontology language, time complexity

INTRODUCTION

With the development of the semantic web (Mao, 2010) and Web Ontology Language (OWL) (Martin et al., 2007; Hasany et al., 2010), we hope to utilize the semantic web technology to integrate intelligently all kinds of web services, thus producing semantic web services. The automatic composition of semantic web services is one of the key technologies in integrating web services and it is much concerned by many researchers form the beginning. The methods for web service composition are dependent on specific framework of web services in a certain degree. The typical framework of web services is OWL-S which is based on OWL and OWL is on the basis of description logic (Horrocks et al. 2003). Description logic is effective in representing and reasoning static knowledge. However, description logic cannot represent and reason dynamic knowledge such as behaviors and services. So, Dynamic Description Logic (DDL) (Shi et al., 2004) is presented by combining description logic ALC, dynamic logic and action theory. The description ability of DDL is over logic and system based on proposition language in describing actions. The problems about reasoning are all decidable in reasoning actions. All the problems about reasoning actions can be transformed to the satisfiable problems. So, they can be reasoned based on the decision algorithm in absence of information. In sum, DDL is more applicable for describing semantic web by combining knowledge of static field based on description logic and action knowledge of dynamic field.

At present, the service composition are mainly the following several methods:

- The service composition methods based on type matching of input and output parameters

A solution based on the DAML-S (Paolucci et al., 2002) is proposed. The overlap relationship of the types of parameters is studied and the partial matching algorithm is proposed (Li and Horrocks, 2003). For large number of services, a synthetic testbed (Constantinescu et al., 2004) is proposed that can be used for simulating large deployments of services and also for generating service composition problems.

- The service composition methods based on artificial intelligence planning

The pioneer of such approaches (McIlraith and Tran, 2002) takes web services as the actions of AI planning. Therefore, web service composition process is the
process of generating planning. The adopted planning algorithm is improved based on logic program Golog. The service composition based on HTN (Hierarchical Task Network) planning algorithm (Sirin et al., 2004) is proposed.

As the description logic can effectively express and reason knowledge of static domain and cannot handle the dynamic knowledge such as actions and services, dynamic description logic DDL (Shi et al., 2004) is put forward and the service description method based on DDL (Shi and Chang, 2008) is proposed. And then the service composition algorithm based on DDL (Peng et al., 2008) is presented. But because the algorithm needs to enumerate all composition sequences, efficiency is difficult to be guaranteed.

In this study, DDL is taken as service description framework. For services which are performed in sequence, a fast and effective service composition algorithm is presented which takes full advantage of DDL describing dynamic run of services. This method utilizes the partial order relationship between services dividing the service composition into two stages: on the first stage, the partial order diagram is constructed on the registration server; on the second stage the fast service composition is implemented based on the partial order diagram to meet the requirement of users. To achieve the goal, First inclusion operation on sets of DDL is extended, the partial order relationship between services is defined based on extended inclusion operation and the satisfiability problem of DDL formula is transformed to operations between sets. Second, the fast composition algorithm is proposed based on DDL and the correctness and time complexity is analyzed. Finally, the algorithm of generating the partial order diagram between services is proposed and the time complexity of the algorithm is analyzed. By analysis and verification, the method of service composition designed in this study is implemented in linear time complexity greatly reducing response time of the system.

**DDL AND SEMANTIC WEB SERVICES**

The basic idea of DDL is that the world is in the process of constant evolution and development and the occurrence of actions can push evolution. The occurrence of actions is restricted by the world today and also influences the world changing from one state to another.

The DDL contains 3 essential elements: concept, relation and actions. Based on simple concept, relation and action, complicated concepts can be constructed with construction operators which is called TBox.

Some definitions related to DDL are given as follows:

**Definition 1:** An atomic action with arguments is the form:

\[ \alpha(V_1, V_2, \ldots, V_n) = (P, E) \]

where, \( \alpha \) is a string representing the name of the atomic action, \( (V_1, V_2, \ldots, V_n) \) is a finite sequence of all individual variables in \( P \) and \( E \) representing the interface of the atomic action, \( P \) is a set of preconditions which must be satisfied before an action are executed and \( E \) is a set of postconditions which denotes the effects of an action.

**Definition 2:** Formulas in DDL are defined and generated as follows:

\[ \varphi, \psi : = C(u) \land R(u, v) \land \neg \varphi \lor \psi \lor \exists \pi \varphi \]

where, \( u \) and \( v \) are individual variables, \( C \) is a concept, \( R \) is a relation and \( \pi \) is an action.

**Definition 3:** An action of DDL is defined as follows:

\[ \pi, \pi' : = \alpha[\varphi \land \pi \cup \pi'] \land \pi, \pi' \land \pi' \]

where, \( \alpha \) is an atomic action and \( \varphi \) is a formula.

The service description in OWL-S and the transformation method from OWL-S description to DDL description are given in the following.

In OWL, the function of an service is described by Input, Output, Precondition and Effect (IOPE) which is denoted by \( S = \{I, O, P, E\} \). Let \( S.I, S.O, S.P \) and \( S.E \) be respectively input, output, precondition formula set and effect formula set of service \( S \). If \( \alpha(V_1, \ldots, V_n) = (P, E) \) is an atomic action of DDL, \( S \) can be described as \( \alpha = S \), where the action name is service name. \( P - S.I \cup S.P \) is the premise formula of actions which is the union of input formula set and precondition formula set of service \( S \); \( E - S.O \cup S.E \) is the result formula of actions which is the union of output formula set and effect formula set of service \( S \). Shi and Liang (2008) proposed a detailed transformation process.

A concrete service example of booking air tickets (Peng et al., 2008) is given below:

**Example 1:** The service description of booking air ticket.

TBox:

- FlightTicketReserveConfirm ⊑ ReserveConfirm,
- TrainTicketReserveConfirm ⊑ ReserveConfirm,
- LDBusTicketReserveConfirm ⊑ ReserveConfirm,
AirTicket ∈ Ticket,
TrainTicket ∈ Ticket,
LDBusTicket ∈ Ticket

In example 1, the input and output of booking air ticket are as follows:

Input:
Tourism staff: one person
Departure: Beijing
Destination: Yunnan
Departure Time: December 13, 2007
Transportation: Aircraft
Credit Card Number: creditCardNo
ID Number: idCardNo
Output: booked air ticket

The following is the DDL description of booking air ticket service.

The initial state

\[
\begin{align*}
A_0 & = \{ \text{Person (person), DeparturePlace(Beijing), ArrivalPlace(Yunnan), departureDate(2007-12-13),} \\
& \text{departureTime(17:00), arrivalDate(2007-12-24), arrivalTime(8:00), IntendVehicle(flight), CreditCardNo(creditCardNo),} \\
& \text{IDCardNo(idCardNo), knows(agent, person), knows (agent, Beijing),} \\
& \text{knows(agent, Yunnan), knows(agent, 2007-12-13), knows(agent, 17:00), knows(agent, 2007-12-24), knows (agent, 8:00),} \\
& \text{knows(agent, flight), knowsVisacard (agent, creditCardNo), knowsIDcard (agent, idCardNo), VisaValidate (person, creditCardNo),} \\
& \text{AirTicket(y), IDCertify(person, idCardNo), SendTicketConfirm (sendTicketConfirm), holds (person, y)} \}. 
\end{align*}
\]

Goal:
\[
\varphi_1 = \{ \text{SendTicketConfirm(sendTicketConfirm), holds (person, y)} \}.
\]

The following are the atomic services used in the matching algorithm.

\(S_1\): The service for searching for the flight information. \(S_1\) searches for the flight information based on the departure place, the arrival place and departure date. The DDL description of \(S_1\) is as follow:

CheckFlight(departurePlace, arrivalPlace, departureDate):
P = \{ DeparturePlace(departurePlace), ArrivalPlace (arrivalPlace), DepartureDate(departureDate), knows (agent, departurePlace), knows(agent, arrivalPlace), knows(agent, departureDate) \}
E = \{ FlightNo(flightNo), knows(agent, flightNo) \}.

\(S_2\): The service for reserving air ticket. \(S_2\) reserves the air ticket based on the flight number, departure date and ID Number. The DDL description of \(S_2\) is as follow:

ReserveFlightTicket(flightNo, departureDate, person, idCardNo):
P = \{ FlightNo(flightNo), DepartureDate(departureDate), Person(person), IDCardNo(idCardNo), IDCertify (person, idCardNo), knows(agent, flightNo), knows (agent, departureDate) \}
E = \{ FlightTicketReserveConfirm(flightTicketReserveconfirm), knows(agent, flightTicketReserveconfirm) \}.

\(S_3\): The service for bank transfer. \(S_3\) is used for bank transfer based on the credit card and the reserve confirmation information. The DDL description of \(S_3\) is as follow:

AccountTransForReserve(person, creditCardNo, reserveconfirm):
P = \{ Person(person), CreditCardNo(creditCardNo), ReserveConfirm(reserveconfirm), VisaValidate (person, creditCardNo), knows (agent, reserveconfirm) \}
E = \{ AccountTransConfirm (accountTransConfirm), knows(agent, accountTransConfirm) \}.

\(S_4\): The service for delivering tickets. \(S_4\) delivers tickets based on the reserve confirmation information and bank transfer information. The DDL description of \(S_4\) is as follow:

SendTicket(person, reserveconfirm, accountTransConfirm):
P = \{ Person(person), FlightTicketReserveConfirm(reserveconfirm), AccountTransConfirm(accountTransConfirm), knows (agent, person), knows(agent, reserveconfirm), knows (agent, accountTransConfirm) \}
E = \{ SendTicketConfirm(sendTicketConfirm), AirTicket(y), holds (person, y) \, knows (agent, sendTicketConfirm) \}.

In example 1, \(S_1\), \(S_2\), \(S_3\) and \(S_4\) are four atomic services. Set \(A_0\) is the initial state composed of the input of users. Set \(\varphi_1\) is a set of goals that users need to achieve. The task is to find the sequence of service composition making the expression \(\varphi_1\) is true in final state.

**THE FAST WEB SERVICE COMPOSITION ALGORITHM BASED ON DDL**

A service composition to meet goals correspond to an action sequence plan = \(\{S_1, S_2, \ldots, S_k\}\), where \(S_k = (P_k, E_k)\) \((0 = i = k)\) is a plan to achieve the goal \(\varphi_i\). That is, \(\varphi_i\) is satisfiable in the state followed by implementation of actions of the plan in sequence. Therefore, the following formula is not satisfiable (Peng et al., 2008):
\[(S_1, S_2, \ldots, S_n)^* \ii \text{Conj}(P) \wedge \text{<plan> } \varphi_n(*)\]

where, \( \wedge \) denotes the logical AND operator, \( \ii \) is the formula \((\text{Conj}(P) \rightarrow \langle P_1, E_1 \rangle \text{ true}) \wedge \ldots \wedge (\text{Conj}(P) \rightarrow \langle P_n, E_n \rangle \text{ true})\) and \( \text{Conj}(S) \) is the conjunction of all formulae in the partial order set \( S \).

In the current state, \( \text{Conj}(P) \) can achieve the goal \( \varphi_i \) through the implementation of the planning \( \langle \text{plan} \rangle \). The formula \( \ii \) ensures that an arbitrary atomic action is certain to execute when its premise is satisfied. And the formula \((S_n, S_{n+1}, \ldots, S_m)^* \ii\) ensures that the sequence composed of atomic actions is certain to implement when the premise is satisfied.

Therefore, as long as an action sequence that makes the formula (*) unsatisfiable can be found, the action sequence is the target service composition sequence that meets the goal. It is inefficient by the enumeration of all possible service sequences to verify the satisfiability. In fact, DDL description for services is simple and effective and there exists partial order relation between services. Therefore, the service composition problem can be transformed to operations on sets to gain more efficient algorithms. To do that, first the extension of inclusion operation on sets is given and then the partial service composition algorithm based on partial order diagram between services is put forward.

The extension to inclusion operation on sets and the definition of partial order between services: Concepts in DDL has the nature of heritage and the extension to inclusion operation on sets is necessary.

**Definition 4:** In DDL, let \( M \) and \( N \) be two sets, \( \subseteq \) is the generalized inclusion operation on sets, then \( M \subseteq N \) iff the following two conditions are satisfied simultaneously:

1. \( \forall x \in M, \exists y \in N, C = D \vee C \iff D \) is satisfiable
2. For a given map \( f \) : variable name \( \rightarrow \) the concept name that the variable belongs to, \( \forall (x, y) \in M, \exists (z, w) \in N, (f(x) = f(z) \vee f(x) \subseteq f(z)) \wedge (f(y) = f(w) \vee f(y) \subseteq f(w)) \) is satisfiable

In the definition 4, the conditions 1 is related to concepts in sets, while the condition 2 is related to the relation in sets. \( C \subseteq D \) means that \( C \) is the sub-concept of \( D \), \( f(x) \subseteq f(z) \) means that \( f(x) \) is the sub-concept of \( f(z) \). In example 1, \( SI. P \subseteq A_9 \) is because the element knows (agent, departurePlace) in SI. P can find a relation with the same name knows(agent, Beijing) in \( A_9 \) and departurePlace and Beijing belong to the same concept name DeparturePlace.

In addition, it is not difficult to find the partial order relation between services. For example, in the service for reserving air tickets, due to the need for information such as flight number, the service for searching for the flight information must first be performed. And the bank transfer needs service reserve confirmation information, so the service for reserving air tickets must be first performed.

The following are the strict formal definition of the partial order between services.

**Definition 5:** Given two services \( S_0 \) and \( S_n \), \( S_0 \) executes before \( S_n \), denoted by \( S_0 \subseteq S_n \), if and only if \( S_0 \subseteq S_n \).

In example 1, \( S_0 \subseteq S_4 \), \( S_3 \subseteq S_0 \), \( S_3 \subseteq S_5 \), and \( S_3 \subseteq S_7 \) because FlightTicketReserveConfirm is the sub-concept of ReserveConfirm.

After determining the partial order between services, the partial order diagram between the services will be formed. A web service composition corresponds to a top-down chain of the partial order. Therefore, the service composition problem is transformed to the process of finding top-down chain in the partial order diagram. In example 1, \( S_0 \subseteq S_1 \subseteq S_2 \subseteq S_3 \) is one target chain to find.

**The basic idea of fast web service composition algorithm:**

The world is a collection of concepts and relations and changes with occurrence of services. As a result, service composition in DDL is turned into the following: the input of users is taken as the initial state of the world \( A_0 \), select a service sequence in which the occurrence of each service will add some new elements for the world; if this sequence occurs and the arrival state set of the world contains all goal elements that users need in the set \( \varphi_n \), the sequence is the service composition sequence meeting goals of users.

Let \( A \) be current state of the world and \( \varphi \) be the goal elements which is not achieved in the current state, according to the above analysis, there will be the following conclusions:

- The service \( S \) in the current state \( A \) can be executed if and only if \( SP \subseteq A \)
- After execution of service \( S \), \( A = A \cup \{S.E\} \)
- After execution of service \( S \), \( \varphi = \varphi - S.E \)

where, \( \cup \) and \(-\) are common operations on sets, \( A \) and \( \varphi \) have initial value \( A_0 \) and \( \varphi_n \).

In order to strike a balance between the execution time of the algorithm and expression ability. This study does not use all syntax elements in DDL because it is not acceptable for the user if the response time of the system is too long. The following is the specific realization process.

First of all, given the initial state set of the world \( A_0 \) and the goal set \( \varphi_n \), find the first service that can be
executed where \( P \subseteq A_r \) must be satisfied. After the service is selected and executed, the state of the world is changed by \( E \) of the service. And then in the service partial order diagram, find all the successor service nodes and select the service that can be executed in the new state of the world. However, there are many services that meet the needs. Which service node should be selected? In order to avoid blind search, set up instruction information set and the instruction information set points out all elements which can be gotten after executing all successor services. Only elements of the goal set \( \varphi \) are in instruction information set can the service be selected.

Below is a general example to illustrate the above realization process

In Fig. 1, there are eight services \( S_1 \sim S_8 \). Each edge connects with the services with the partial order relation, the alphabets next to the services refer to the specific elements in the set. For example, the alphabets letters +c and +d next to \( S_j \) show \( S_j E = \{e, d\} \). That is, the implementation of the service \( S_j \) will add two elements \( c \) and \( d \) to the current state. The set \( \{c, d, e, f, g, h, i, j\} \) is the instruction information set of \( S_j \), indicating all elements which can be gotten after executing all successor services of \( S_j \). As the partial order diagram has hierarchy, instruction information sets of the upper level can be obtained by merging instruction information sets of the lower level.

**The fast service composition algorithm**: After obtaining the partial order diagram between services, the fast service composition based on DDL can be executed.

**Algorithm 1**: the fast service composition algorithm based on DDL.
In this case, the changes in the stack are as follows:

1. \((S_0, \{m, n, a\}, \{i\})\)
2. \((S_0, \{m, n, a, c, d\}, \{j\})\)
3. \((S_0, \{m, n, a, c, d, f, g\}, \{j\})\)
4. \((S_0, \{m, n, a, c, d, f, g, j\}, \{\})\)
5. \(*\Phi\) (note: \* indicates finding a path)

The changes in the stack show the access path in the partial order diagram. In step 1, only \(S_0\) is selected and pushed into the stack. In step 2, when accessing \(S_0\), according to instruction information goals cannot be achieved along \(S_0(\varnothing \subseteq S_0, C)\). And goals can be achieved along \(S_0(\varnothing \subseteq S_0, C)\). Therefore, \(S_1\) is abandoned and \(S_0\) is pushed into the stack. In step 3, when accessing \(S_0\), according to instruction information goals cannot be achieved along \(S_0(\varnothing \subseteq S_0, C)\). And goals can be achieved along \(S_0(\varnothing \subseteq S_0, C)\). Therefore, \(S_0\) is abandoned and \(S_0\) is pushed into the stack. In step 4, when accessing \(S_0\), \(S_0\) is pushed into the stack. In step 5, when accessing \(S_0\), \(\varnothing\) is empty and the goals are achieved. So, the target chain is \(S_0 < S_1 < S_2 < S_3\).

- Single-goal and multi-path: \(A = \{m, n\}\) and \(\varnothing = \{i\}\)

In this case, the changes in the stack are as follows:

1. \((S_0, \{m, n, a\}, \{i\})\)
2. \((S_0, \{m, n, a, b\}, \{i\})\)
3. \((S_0, \{m, n, a, b, c\}, \{i\})\)
4. \((S_0, \{m, n, a, b, c, d\}, \{i\})\)
5. \((S_0, \{m, n, a, b, c, d, e\}, \{i\})\)
6. \((S_0, \{m, n, a, b, c, d, e, f\}, \{i\})\)
7. \((S_0, \{m, n, a, b, c, d, e, f, g\}, \{i\})\)
8. \((S_0, \{m, n, a, b, c, d, e, f, g, h\}, \{i\})\)
9. \(*\Phi\)

In step 3, when accessing \(S_0\), \(\{d\}\) is achieved, \(\{i\}\) can be achieved along either \(S_0\) or \(S_1\) and so \(S_0\) and \(S_1\) are pushed into the stack. In step 6, when accessing \(S_0\), the first target chain \(S_0 < S_1 < S_2 < S_3\) is found. In step 9, when accessing \(S_0\), the second target chain \(S_0 < S_1 < S_2 < S_3\) is found. In step 9, the third target chain \(S_0 < S_1 < S_2 < S_3\) is found.

It is shown that all target sequences that can satisfy goals of users can be found using the algorithm and there is no redundant information in the stack.

- Multi-goal and single-path: \(A = \{m, n\}\) and \(\varnothing = \{b, i\}\)

In this case, changes in stacks are as follows:

1. \((S_0, \{m, n, a\}, \{b, i\})\)
2. \((S_0, \{m, n, a, b\}, \{i\})\)
3. \((S_0, \{m, n, a, b, c\}, \{i\})\)
4. \((S_0, \{m, n, a, b, c, d, e\}, \{i\})\)
5. \((S_0, \{m, n, a, b, c, d, e, f\}, \{i\})\)
6. \(*\Phi\)

In step 2, when accessing \(S_0\) according to instruction information \(\{b, i\}\) can be achieved along \(S_0\) while only \(\{i\}\) can be achieved along \(S_0\). Therefore, \(S_1\) is abandoned and \(S_0\) is pushed into the stack. In step 5, when accessing \(S_0\), \(\varnothing\) is empty and the goals are achieved. So, the target chain is \(S_0 < S_1 < S_2 < S_3\).

- Multi-goal and multi-path: \(A = \{m, n\}\) and \(\varnothing = \{d, i\}\)

In this case, changes in stack are as follows:

1. \((S_0, \{m, n, a\}, \{d, i\})\)
2. \((S_0, \{m, n, a, b\}, \{i\})\)
3. \((S_0, \{m, n, a, b, c, d, e\}, \{i\})\)
4. \((S_0, \{m, n, a, b, c, d, e, f\}, \{i\})\)
5. \((S_0, \{m, n, a, b, c, d, e, f, g\}, \{i\})\)
6. \(*\Phi\)

In step 3, when accessing \(S_0\), \(\{d\}\) is achieved, \(\{i\}\) can be achieved along either \(S_0\) or \(S_2\) and so \(S_0\) and \(S_2\) are pushed into the stack. In step 6, when accessing \(S_0\), the first target chain \(S_0 < S_1 < S_2 < S_3\) is found. In step 9, when accessing \(S_0\), the second target chain \(S_0 < S_1 < S_2 < S_3\) is found. In step 9, the third target chain \(S_0 < S_1 < S_2 < S_3\) is found.

No path: \(A = \{m, n\}\) and \(\varnothing = \{i, j\}\)

In this case, changes in stack are as follows:

1. \((S_0, \{m, n, a\}, \{i, j\})\)
2. \((S_0, \{m, n, a, c, d\}, \{i, j\})\)
3. \((S_0, \{m, n, a, c, d, f, g\}, \{i, j\})\)
4. \(*\Phi\)
When accessing $S_i$ and $S_j$, the successor service set $S_i \cap S_j$ can be achieved according to the instruction information set. In step 4, when accessing $S_r$, the predecessor service set $S_r \cap S_i$ cannot be achieved simultaneously according to the instruction information set of $S_i$ and $S_r$. The stack is empty and no target chain can be found. In fact, there exists no service composition meeting requirements of users.

From the above examples it can be concluded that the algorithm can deal with all cases correctly and the validity and correctness can be guaranteed.

Here are the time complexity of the fast service composition. For the case of the single path, let $h$ be the number of the level of the partial order diagram, $N$ be the number of services and $r$ be the average number of branches of nodes of the partial order diagram, $h = \log N$ is the time complexity of the fast service composition. For the case of the multiple paths, let $N$ be the number of services, $m$ be the number of target chains and $r$ be the number of branches and nodes of the partial order diagram, $O(N^m + m\log N)$ is the time complexity of the fast service composition. Therefore, the algorithm in this study is more efficient than the algorithm of enumerating all possible service sequences.

In the above analysis of time complexity, it is assumed that the constraint condition $t \cdot C \subseteq A$ is always true. In practical run, it is a condition for pruning. So, the practical time complexity is lower.

**Algorithm for generating partial order diagram of services:** Algorithm is built on the basis of the partial order diagram between services. Therefore, how to quickly create the partial order diagram between services is also a very important issue. The diagram describes the static relation between services and it can be created in the registration and management server of web services. Because the service registration is dispersed, the partial order diagram can be generated in a incremental way.

Let $G$ be a partial order diagram and $r$ be a new arrival service. Service $r$ needs to be added to diagram $G$ and the relevant data need to be modified. The algorithm for generating partial order diagram between services is described below.

**Algorithm 2:** The algorithm for generating partial order diagram between services

For each $s \in \text{Services}$

If ($r$ is the successor service of $s$) then $r$ is added to the successor service set of $s$ and $s$ is added to the predecessor set of $r$

If ($r$ is the predecessor service of $s$) then $r$ is added to the predecessor service set of $s$ and $s$ is added to the successor service set of $r$

Endfor

$r$ is pushed into the stack

the updated service node set $D = \{\}$

While (the stack is not empty) do

Pop the top element off the stack and assign it to $s$

If (the successor service set of $s$ is not empty) then

For each $t$ to the successor service set of $s$

If ($t$ is not empty) then

$t.C = t.C \cup r.E_r$

$D = D \cup \{t\}$

$t$ is pushed into the stack;

End If

End For

End If

End while

In Fig. 1, let $S_i$ be the new arrival service. Two branches $S_i \rightarrow S_a \rightarrow S_b \rightarrow S_c \rightarrow S_a$ and $S_a \rightarrow S_b$ are passed: the element $i$ is added to the instruction information set of nodes of every branch.

**CONCLUSION**

In this study, a web service description method based on DDL is given and an improved service composition algorithm based on DDL is proposed. The algorithm firstly extends the inclusion operation on sets, on this basis the partial order relation between services is defined and then the partial order diagram is formed. According to the partial order diagram the fast service composition algorithm is implemented. At last an example is given in order to verify the correctness and effectiveness of the proposed algorithm. The results show that this algorithm is correct and has good adaptability and its time complexity is greatly improved.

Future work is to fully excavate the potential of dynamic description logic, continue to expand operations on sets to incorporate more syntax elements of DDL and grammatical components and seek the algorithms to support rapid web service discovery and automatic service composition.

**ACKNOWLEDGMENTS**

This study is supported in part by the National Natural Science Foundation of China under Grants 60773034, 90618023, 90718012 and 60803032; the Scientific and Technological Developing Program of Shandong Province of China under Grant 2008GG30001024; the National Basic Research Program of China (973 Program) under Grant 2010CB328101; the Taishan Scholar Construction Project of Shandong Province, China; the Open Project of the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of
REFERENCES


