AI planning-based semantic web service composition

G.R. Karpagam* and A. Bhuvaneswari

CSE Department,
PSG College of Technology,
Coimbatore – 641 004, Tamilnadu, India
E-mail: grkrm@mail.psgtech.ac.in
E-mail: bhuvan888@yahoo.com
*Corresponding author

Abstract: Semantic web service composition constitutes the phases namely specification, matchmaking, planning, validation, discovery and execution. As web service composition is categorised as an AI planning problem, the objective of this paper is to propose a fluent calculus approach for the planning phase of the semantic web service composition. Among various methods to solve the AI planning problem, logic programming has been identified as the most appropriate candidate to handle the runtime behaviours of web services. FLUX has been used for representing constraints in fluent calculus formalism. Label transition system analyser (LTSA) formalisms are used to validate the plans generated using FLUX. An e-shopping domain is considered as a case study.

Keywords: semantic web service composition; AI planning; fluent calculus; automation; dynamism.


Biographical notes: G.R. Karpagam is a Professor with 15 years of experience in the Department of Computer Science and Engineering in PSG College of Technology. She is an IEEE member doing her research in the field of web services, model driven architecture and cloud service discovery. She has published papers in national/international journals and conferences and a Reviewer of international journals. She has co-authored monograph for database management systems and has assisted in writing ISO procedure manual. She has established industry funded laboratories namely Open-Source Software Laboratory, Service Oriented Architecture Laboratory and Cloud Computing Laboratory in collaboration with leading MNCs.

A. Bhuvaneswari is a full time research candidate pursuing her research in the Department of Computer Science and Engineering in PSG College of Technology affiliate to Anna University. Her area of interest includes bio inspired computation and web service composition. Her research focuses on investigating the AI techniques for planning phase of semantic web service composition. She has published papers in IEEE and ACM Digital Libraries and national/international journals.

1 Introduction

Web service composition orchestrates existing services to achieve a larger task, resulting in a new composite and value-added web service. The major challenges in composition are:

1. to discover the exact and qualified services
2. to automate the generation of a plan that satisfies the user request.

The first challenge is unravelled by representing web services as semantic web services. They enhance the service discovery by annotating the web services based on shared ontologies. Ontologies are recognised as the basis for shared conceptualisation of a domain and comprise of concepts with their relationships and properties. The explicit definition of semantics of a web service in a particular domain should be mapped with the ontological concepts of that domain. The second challenge can be tackled by various ways of automating the service composition as shown in the Figure 1. Several papers (McDermott, 2000; Srivastava and Koehler, 2003; Carman et al., 2003; Sirin and Parsia, 2004) have explored the prospects of applying AI planning techniques to automate the generation of a plan that achieve the desired goal.

An AI planning problem (Rao and Su, 2004) is generally described as a tuple <S, S0, G, A, T> where S represents the
set of all the possible states of the world; $S_0 \subset S$ is the initial word state; $G \subset S$ represents the goal state of the planner; $A$ represents the set of all possible actions; the translation relation $T \subset S \times A \times S$ defines the preconditions and effects for each action. Autonomous agents calculate the effect of different action sequences in advance using the current state of the action. This will aid in choosing the best plan. One of the most important approaches among several AI techniques is domain theories based on classical logics. Logic programming has three major classes of calculi namely situation calculus, event calculus and fluent calculus.

**Figure 1** Classification of service composition

In classical planning, all objects are available in the initial state and the actions change the state of objects, whereas, web services create new message objects at runtime. Hence, modelling web services with the currently available planning techniques is as an open research problem (Srivastava and Koehler, 2003). This paper identifies fluent calculus as a solution to this open problem based on the following two-fold reasons:

1. Supports logic programming for reasoning and planning in dynamic domains. It extends the situation calculus which had been already proved for composition by Nariai et al. (2005).
2. Uses principle of progression in which the computational effort remains the same irrespective of the number of performed actions.

Based on the investigation, web service composition is facilitated by semantic web services and AI planning domain which leads to AI-based semantic web service composition. Semantic web service composition constitutes six phases namely specification, matchmaking, planning, validation, discovery and execution. Specification phase provides an easy way for a user to specify task goals, requirements and constraints without extensive domain knowledge. Matchmaking phase provides a way to identify the semantic service description based on the service request. Planning phase provides an automatic way to compose a plan based on the input, output, preconditions and effect of services. Validation phase provides techniques to ensure that the composite process realised through a plan. Discovery phase provides a way to discover services that satisfy task specifications in the workflow. Execution with monitoring phase provides a framework for monitoring and executing the services including automatic fault-handling mechanisms.

Generally, specification phase will represent user goal according to the planning domain. Since semantic web services are utilised in this work OWL-S is used for request specification. The planning domain of the proposed work expects the semantic description of services that should be composed. Hence, the service identification in the planning phase is separated into a new phase called matchmaking phase. This phase obtains the service request from the specification phase and finds the match between service request and service advertisement. The OWL-S representation of relevant services is given as input to the planning phase. The advantage of this segregation is that planning phase will be provided with the set of services that are relevant to service request. Hence, the replanning and dynamism can be introduced in the planning phase.

Among six phases, the planning phase is crucial as it needs intelligence to automate the construction of a plan satisfying the user requirement. Russel and Norvig (1995) describe the problem of planning as: “Planning can be interpreted as a kind of problem solving, where an agent uses its beliefs about available actions and their consequences, in order to identify a solution over an abstract set of possible plans”. A planning problem should comprise of descriptions for:

1. initial state of the world
2. desired goal
3. set of actions to be executed in a formal language.

This paper suggests fluent calculus as the formal language for planning phase.

The planning phase is achieved by applying fluent calculus formalism categorised under logic programming. The essentials of fluent calculus are:

1. sorts
2. macros
3. axioms.

Fluent calculus (Thielscher, 1999, 2001; Chifu et al., 2008) has four sorts: fluents, states, actions and situations. A fluent is an atomic component that describes a variable state. A state is a collection of fluents at a particular point of time like a snapshot. The fluents in a particular state are connected via the binary function symbol ‘o’, written in infix notation, which is assumed to be both associative and commutative, and to admit a unit element, denoted by $\phi$. Actions are the ones that can change the state. A situation is a sequence of states. The basic signature of fluent calculus is based on these sorts having finite set of functions into
action and state sort and two binary predicates tabulated in Table 1.

Macros defined by the fluent calculus are as follows:

- $\text{Holds}(F,z) = (\exists z')z = f o z'$: Denotes that fluent $f$ holds in state $z$ and $z$ can be decomposed into $f$ and some state $z'$
- $\text{Knows}(f,s)$: Denotes that formula $f$ is known in situation $s$
- $\text{Kwhether}(f,s) = \text{Knows}(f,s) \lor \text{Knows}(\neg f,s)$: indicates whether the true value of $f$ is known in situation $s$
- $\text{KnowVal}(x,f,s)$: Denotes that the functional value $x$, of formula $f$ is known in situation $s$.

### Table 1. Basic signature of fluent calculus

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Function into action sort</th>
<th>State(s)</th>
<th>Explanation</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>Initial situation</td>
<td>STATE</td>
<td>SIT</td>
<td>STATE \rightarrow SIT</td>
</tr>
<tr>
<td>Do(a, s)</td>
<td>The situation obtained by executing the action $a$ in situation $s$.</td>
<td>STATE</td>
<td>ACTION $\times$ SIT</td>
<td></td>
</tr>
<tr>
<td>$\phi$</td>
<td>Empty state; a state in which no fluent is true.</td>
<td>STATE</td>
<td>STATE</td>
<td></td>
</tr>
<tr>
<td>$o$</td>
<td>A function that maps two states into a new state in which the fluents of both arguments hold</td>
<td>STATE $\times$ STATE</td>
<td>STATE $\rightarrow$ STATE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary predicate</th>
<th>Poss</th>
<th>KState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represents the precondition of an action in a state</td>
<td>ACTION $\times$ STATE</td>
<td>SIT $\times$ STATE</td>
</tr>
</tbody>
</table>

Fluent calculus is based on the following set of axioms. The foundational axioms ensuring the identification of a state if the fluents held in it are known; the unique name axioms ensuring that the fluent and action terms denote the unique objects; the axiom for defining the initial situation defines $S_0$ which denotes the real world situation in which no actions have been performed. The action precondition axioms denote the precondition that should be satisfied to execute an action. The state update axioms define the relation between a state and its successor. The auxiliary axioms define the restrictions on the knowledge state using the domain constraints that need to be satisfied for a state to appear in reality.

Fluent calculus formalisms are provided by FLUX (Thielscher, 2004, 2005), a constraint logic programming language which also provides logical constructs for assembling primitive actions into complex actions like concurrence, conditional, sequential and non-deterministic actions. By combining progression with much of GOLOG’s powerful concept for plan search control, FLUX combines the best of both worlds.

Semantic web services are described by OWL-S, a web ontology language that helps in exactly finding a specific service. It has a well-defined semantic, making service interpretable and unambiguous. It also allows the definition of objects, complex relationships between them, including class, subclass and cardinality restrictions. OWL-S (Bhuvaneswari and Karpagam, 2010a) comprises of three components: service profile which describes the properties of a service, such as, what it offers, its input and outputs; service model that describes how it works and whether it allows composition of services and service grounding which describes how an agent can access a service, thus, specifying a communication protocol, message formats, etc. An ontology editor (Protégé) is used to obtain OWL-S specification of web services from existing WSDL. AI planning is applied for achieving only automation in composition by Peer (2005). This paper includes both automation and dynamism in planning operation.

### 2 Methodology

A plan should be automatically generated to execute the user request using AI planning strategy. Among numerous planning strategies fluent calculus formalism is considered and applied for the planning phase of semantic web service composition. When a user query is given to the system, specification phase will analyse and generate the service request as OWL-S specification. Based on this request matchmaking phase will semantically match and find the OWL-S specification of service operations which are relevant to the user query. Planning phase will translate the OWL-S service request and service operations into fluent calculus formalism and automatically constructs the plan. This paper concentrates on the contribution of AI in planning phase. The service request generated by the specification phase will be as in Figure 2.

Matchmaking phase will accept this request and find the OWL-S specification of services relevant to this query. A snippet of the fetched OWL-S representation of a service is shown in Figure 3. The snippet shows only the input for convenience. Planning phase will populate the fluent knowledge base by converting the service ontology into fluent calculus formalisms. User goal are specified as required inputs and outputs which are represented as a set of values in fluent calculus. The algorithms for these conversions are explained in next section.

The planner algorithm will use this knowledge base. The planning phase is elaborated in Figure 4.
AI planning-based semantic web service composition

3 Case study

Let us consider the scenario in which the user needs to purchase a book through online shopping. The user provides a title of the book as input. The system should find the author details, price details and availability of the book. If the book is available credit card is validated and processed; otherwise, alternate books are displayed to the user. The flow of this case study can be illustrated as in Figure 5.

OWL-S description of web service PurchaseBook is shown in Figure 6(a). The conversion of OWL-S into fluent calculus axioms is specified in the Figure 6(b). The inputs and preconditions of OWL-S will be considered as the knowledge preconditions and physical preconditions respectively in fluent calculus. The outputs and effects of OWL-S will be considered as the knowledge effects and physical effects, respectively in fluent calculus. This section provides the algorithms for the case study to convert the atomic process of OWL-S into fluent calculus translation. The algorithm converts inputs and preconditions of the atomic process into action-precondition axioms and outputs and effects into state update axioms calculus (Thielscher, 1999). The translation algorithm for the inputs and precondition of atomic process PurchaseBook is given below in Figure 7. The procedure for the translation of outputs and effects into state update axiom is shown below in Figure 8. The effect of the execution of this web service will be Payment is made and book is purchased.

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Selected service</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>bookFinder</td>
<td>S12</td>
</tr>
<tr>
<td>S2</td>
<td>getPrice</td>
<td>S24</td>
</tr>
<tr>
<td>S3</td>
<td>isBookAvailable</td>
<td>S35</td>
</tr>
<tr>
<td>S4</td>
<td>purchaseBook</td>
<td>S43</td>
</tr>
<tr>
<td>S5</td>
<td>displayOp</td>
<td>S55</td>
</tr>
</tbody>
</table>

Let us consider the services to accomplish the online book purchasing scenario that are specified in Table 2. Each service will have n number of alternate services in a registry. After obtaining the required translations, Planner ()
algorithm in Figure 9 generates a plan automatically by employing forward chaining strategy (Chifu et al., 2008). The initial state is a set of user inputs.

The atomic operation whose precondition matches with this initial state is selected, added to the set and its state update axiom is executed. The set of inputs(S) will be updated with the inputs of the new operation added to the set. This will be repeated until the set of outputs becomes a subset of set of inputs(S). Table 3 illustrates the states and actions taken in each state. The letters M, S, A are abbreviations for the actions match, select and add and the letters NM, N are that of no match and next.

The tasks are {bookFinder, getPrice, isBookAvailable, purchaseBook, displayOp}. S₀ is the initial state which contains the user specified inputs. The successor states are updated with the inputs of the operation that is added to the list. aᵢ represent the actions that are matched, selected and added to the list. The algorithm backtracks and finds the a₁ and a₂ for tasks t₄ and t₅, respectively.

**Figure 6** (a) PurchaseBook OWL-S representation, (b) OWL-S to fluent calculus conversion

```xml
<process:AtomicProcess rdf:ID = "PURCHASE_BOOK">
  <process:hasInput rdf:resource = "#_BOOK"/>
  <process:hasInput rdf:resource = "#_AUTHOR"/>
  <process:hasInput rdf:resource = "#_PRICE"/>
  <process:hasInput rdf:resource = "#_CREDITCARDNUM"/>
  <process:hasOutput rdf:resource = "#_DEBITAMOUNT"/>
  <process:hasOutput rdf:resource = "#_PURCHASECONFIRMED"/>
</process:AtomicProcess>
```

**Table 3** List of states and actions

<table>
<thead>
<tr>
<th>S. no.</th>
<th>At</th>
<th>Tasks</th>
<th>Operation</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S₀</td>
<td>[1, 2, 3, 4, 5]</td>
<td>[ ]</td>
<td>NM, N</td>
</tr>
<tr>
<td>2</td>
<td>S₁</td>
<td>[1, 2, 3, 4, 5]</td>
<td>[ ]</td>
<td>NM, N</td>
</tr>
<tr>
<td>3</td>
<td>S₂</td>
<td>[2, 3, 4, 5]</td>
<td>[a₁]</td>
<td>M, S, A</td>
</tr>
<tr>
<td>4</td>
<td>S₁</td>
<td>[2, 3, 4, 5]</td>
<td>[a₃]</td>
<td>NM, N</td>
</tr>
<tr>
<td>5</td>
<td>S₄</td>
<td>[3, 4, 5]</td>
<td>[a₃, a₅]</td>
<td>M, S, A</td>
</tr>
<tr>
<td>6</td>
<td>S₅</td>
<td>[3, 4, 5]</td>
<td>[a₁, a₅]</td>
<td>NM, N</td>
</tr>
<tr>
<td>7</td>
<td>S₆</td>
<td>[4, 5]</td>
<td>[a₃, a₅, a₇]</td>
<td>M, S, A</td>
</tr>
<tr>
<td>8</td>
<td>S₇</td>
<td>[4, 5]</td>
<td>[a₃, a₅, a₇]</td>
<td>NM, N</td>
</tr>
<tr>
<td>9</td>
<td>S₈</td>
<td>[5]</td>
<td>[a₅, a₇, a₉]</td>
<td>M, S, A</td>
</tr>
<tr>
<td>10</td>
<td>S₉</td>
<td>[ ]</td>
<td>[a₁, a₅, a₇, a₁, a₂]</td>
<td>M, S, A</td>
</tr>
</tbody>
</table>
Figure 8  Outputs and effects into state update axiom

**Procedure**  AtomicProcess_EffAndOutput

Set $O = \{\text{DebitAmount, PurchaseConfirmed}\}$

Set $E_1 = \text{DebitAmount} \land \text{PurchaseConfirmed}$

FOR EACH output IN $O$

CALL macro_Knows(output) returning $E_1$; END FOR

RETURN Poss (BookPurchase, z) ⊃ state(Do(BookPurchase, z)) \land

knowsVal(CreditCardNum) \land knowsVal(purchaseDate) \land

knowsVal(PubName) = state(z) + BookPrice(unknown) \land

BookConfirm(unknown) \land

isPaymentMade(CreditCardNum, Price) \land

isPurchased(PubName, purchaseDate)

Figure 9  Planning algorithm

**Procedure**  Planner

$z$ – Set of outputs, $p$ – Composition plan

$S$ – Current state, $S_0$ – Initial State (Set of Inputs) $A$ – Set of possible actions (service operations)

$S_i$ – Current Stage, MAXITER – maximum number of iterations

BEGIN

SET $p = \{ \}$; SET $A = \{a_1, a_2, ..., a_n\}$;

SET $S = S_0$; SET $\emptyset = \{ \}$;

WHILE ($z \not\subset S$)

$S_i = \emptyset$;

FOR EACH $a_i$ in $A$

IF Poss($a_i, S$) THEN $S_i = S_i \cup \{a_i\}$; execute_stateupdate($a_i$);

$A = A - \{a_i\}$

END IF

$i = i + 1$;

END FOR

$p = p \cup S_i$; IF($i$==MAXITER) THEN

RETURN “no matching services”

END IF END

WHILE

// deleting redundant actions

FOR EACH $a_i$ in $p$

IF Poss($a_i, S$) = Poss($a_j, S$) THEN

$p = p - \{a_i\}$

END IF

END FOR

LTSA is used for verifying the plan generated by the fluent calculus planner algorithm. LTSA is a verification tool for concurrent systems which mechanically checks the specification of a concurrent system whether it satisfies the properties required for its behaviour. In addition, LTSA supports specification animation to facilitate interactive exploration of system behaviour. It will generate the finite state machine for the corresponding plan as shown in Figure 10.

The FSM generated by the LTSA tool can be verified and validated by the animator plug-in which helps in visualising the execution order of the generated plan. Figure 11 shows the execution pattern of the plan generated. When the book is not available the bookFinder service is invoked once again by considering the alternate book provided by the system. The purchaseBook is invoked when the book is available.

Figure 10  Plan generated by the planner

Figure 11  Flow of the plan

The services of the initial plan are selected based on QoS parameters (Bhuvaneswari and Karpagam, 2010b) service availability, reputation and execution time. These QoS parameters are specifically taken into account due to the following reasons:

1. The user requirement should be satisfied as quick as possible so execution time is considered.

2. Service can be utilised only when it is having high availability so service availability is considered.

3. Among $n$ number of services, a reputed service will be the best choice as it is the value provided by clients.

Web service composition will be completed if and only if execution of the plan is monitored. While monitoring the execution a requirement may arise to change the plan under execution. In this case, dynamism should be introduced in planning phase as modification must be done at runtime. This leads to two cases of dynamic planning:
Case 1 If a service is not available and it has an alternate service then that particular service can be altered.

Case 2 If a set of services are not available then the entire plan should be restructured for successful execution.

The entire plan may be changed in the case when more number of services should be replaced. The proposed dynamic planning algorithm is shown in Figure 12. A threshold variable is assigned to check whether minimum number of the services is available or not. Let us consider that each service has five similar services. The available list of alternate services for every service is shown in the Table 4.

Case 1 When any service in the plan generated by the planner needs to be changed, it will select from the set of services in Table 4. The permissible values (Bhuvaneswari and Karpagam, 2010b) of the QoS parameters are: execution time < 3, availability must be in the range of 0.75...1 and reputation should be in the range of 0...5. QoS values of the web services are updated periodically. Let us assume that initially QoS values of S43 = {2, 0.8, 3} and during execution the value decreases as S43 = {1, 0.5, 2}. This service must be altered by one of the set of available services for P4. If S41 = {2, 0.75, 3} then S43 must be replaced by S41 by invoking the procedure Alter_Service( ) in Figure 13.

Here, the web service bookFinder has been replaced with book_author_service. Hence, the modified plan after considering Case 1 of dynamic planning will be as in Figure 14.

Figure 15 shows the animated output of the modified plan in which the book_author_service is invoked until the alternate book specified by the system is available for purchase.

### Table 4 List of available services

<table>
<thead>
<tr>
<th>Service</th>
<th>Alternate services</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S11, S12, S13, S14, S15</td>
</tr>
<tr>
<td>S2</td>
<td>S21, S22, S23, S24, S25</td>
</tr>
<tr>
<td>S3</td>
<td>S31, S32, S33, S34, S35</td>
</tr>
<tr>
<td>S4</td>
<td>S41, S42, S43, S44, S45</td>
</tr>
<tr>
<td>S5</td>
<td>S51, S52, S53, S54, S55</td>
</tr>
</tbody>
</table>

### Algorithm for dynamic planning

**Procedure** Dyn_plan_Comp

**Inputs:**
- n – number of services
- p – Expected percentage of available services
- avail – availability of a service
- max_avail – expected availability of a service
- ex_time – execution time of a service
- ex_range – expected range of execution time
- rep – reputation of the service
- rep_range – expected range of reputation
- fs – number of failed services
- plan = [list of services]

**Output:** new plan

**BEGIN**

fs = 0;

Set threshold = n*p

FOR EACH service IN plan

IF (¬holds(hasQoS(service, avail, max_avail), z) ∨ ¬holds(hasQoS(service, ex_time, ex_range), z) ∨ ¬holds(hasQoS(service, rep, rep_range), z)) THEN

fs = fs + 1; n = n – fs;

ELSE

IF holds(hasThresh(n, threshold)) THEN CALL Planner ( ); SET plan = p; RETURN plan;

ELSE

CALL Alter_Service ( ); SET Sij = Sik; RETURN Sij;

END IF;

END IF; END FOR; END

### Algorithm for Alter_Service

**Procedure** Alter_Service( )

**Input:** Sij – Failed Service

**Output:** Sik – Alternate Service

**BEGIN**

FOR k in 1 ... n

IF Sij ≠ Sik THEN

IF holds(hasQoS(Sik, avail, max_avail), z) ∨ holds(hasQoS(Sik, service, ex_time, ex_range), z) ∨ holds(hasQoS(Sik, service, rep, rep_range), z) THEN

RETURN Sik

END IF

ELSE

CALL Alter_Service ( ); SET Sij = Sik; RETURN Sik;

END IF;

END IF; END FOR;

END
3.2 Case 2

If the number of failed services (fs) shown in Figure 12 is greater than the threshold value then algorithm Planner( ) will re-do the procedure by creating a new plan. For example, if p is 75% and fs is 4 services then the threshold is not met and hence, new plan will be generated which will omit the non-essential services as per user feedback. In this case, the planning becomes semi-automatic since user is involved in deciding the non-essential services.

Let us consider the services for checking the book availability is not available then user is allowed to select the essential services. In this case, the isBookAvailable service is omitted by user. Therefore, the modified plan in this Case 2 will be as shown in Figure 16. Thus, whenever the QoS values execution time, availability and reputation are decreasing than the specified value then planning can be done dynamically.

4 Related works

4.1 Composition

The service composition is defined as the process of combining and linking existing web services to assemble new web processes, when no atomic web service can fulfil the user’s requirements. A comparison of various web service composition models which identifies common characteristics and features had been done by Dustdar and Schreiner, 2005). In accordance with Srivastava and Koehler (2003), when a number of processes or agents concurrently and cooperatively try to achieve some goal, processes can spawn off other processes at runtime. For example, each hotel request could invoke a hotel service like checking the room availability, type of room, etc. The number of services is not known at design time, but known only at runtime. The processes will partially interleave with each other, synchronise with each other or run fully independent of each other. Each process exhibits a complex behaviour, and certain planned operations may fail and require recovering from a failed execution. Kuter et al. (2005) identifies three key features of service-oriented planning:

1. the planner’s initial information is incomplete
2. the planning system should gather information during planning
3. web services may not return needed information quickly, or at all.

Since there are crucial issues to be considered service composition has attracted researchers’ attention. This paper concentrates on how automation and dynamism can be introduced in service composition with the help of semantic web services and planning domain of artificial intelligence.

4.2 Automation

The automation in planning phase of web service composition can be achieved in two-ways either by:

1. generating the process model automatically
2. locating the correct services if an abstract process model is given.

As in the second method, the abstract process model needs to be defined manually, this paper considers the first method for automation. The need for automation is identified by Medjahed (2004) as:

1. the dramatic increase in number of web services available on the web
2. dynamic nature of the web service space
3. heterogeneous environment of the web service
4. the interaction among the autonomous web services.

Rao and Su (2004) discusses about the automated web service composition methods and concludes that workflow-based automation is useful when the process model is provided and the AI planning-based automation is useful when the requester has no process model but has a set of constraints and preferences.

4.3 Dynamism

Dynamic web service composition demands the change in either composition plan or invoked web services due to reasons like service unavailable, network problem, etc. In general, the services can have two representation viz. abstract services and concrete services. The abstract service
is the semantic unambiguous description of a service whereas the concrete service is the physically existing service that matches the above said description. There may be more than one matching concrete services for an abstract service. While executing the plan, i.e., when a concrete service is assigned to the abstract service, the composition plan may fail. Hence, dynamically planning the service composition becomes a crucial phase of automated service composition. Dynamically planning the service composition is to dynamically modify the bindings between abstract and concrete services or change the entire plan. Berbner et al. (2006) propose a heuristic-based approach for dynamic planning. Claro (2004) suggests a planning methodology called reactive planning which will re-do the plan as and when needed to achieve the initial goal. This paper depends on the quality of service parameters to apply dynamism in the planning phase of semantic web service composition.

4.4 Automation and dynamism

Most of the works related to web service composition concentrate on automating the plan generation. Only a very few work is carried out in execution monitoring of the composed plan. Web service composition life cycle completes only if execution monitoring is also considered. Fluent calculus provides the ability to work with both planning and dynamic modification of plan. The automatic web service composition is defined as planning problem in the fluent calculus because both the planning problem and the composition problem are meant for searching an ordered set of operations that starts from an initial state and completes the goal and a task is the main component for both the problems. A task in planning problem is an action and in composition it is a web service operation. The action and web service operation both have input, output, preconditions, and effects (IOPE). In the previous work (Bhuvaneswari and Karpagam, 2010b), service composition was done manually using OWL-S plug-in and the evaluation of QoS parameters was experimented as an optimisation problem using particle swarm optimisation, an evolutionary computing algorithm. As an extension of the previous work planning operation of compose phase is considered in this paper. The semantic description of the web service will help only in reasoning (discovery and select phases). As per the discussion in Peer (2005), dynamic composition of services is a hard problem and it is not entirely clear which techniques serve the problem best. Onur et al. (2008) says that none of the currently available OWL-S service composition planners does allow for dynamic planning. In the previous paper (Bhuvaneswari and Karpagam, 2010a), LTSA formalism was used to generate the finite state machine representation of the plan generated using fluent calculus. This paper verifies the FSM by applying animator in the LTSA tool to validate the plan generated by fluent calculus.

5 Contributions

In order to investigate the use of AI planning methodologies for semantic web service composition, the following investigation have been made in the paper:

1 Investigating the impact of introducing automation and dynamism in semantic web service composition.
   - Section 1 provides the background of the principles and Section 4 discusses elaborately about the various research relevant to this paper.
2 Investigating the impact of fluent calculus in achieving automation in semantic web service composition.
   - Section 1 provides the background of the principles. Section 2 discusses the methodology elaborately how this investigation was carried out.
3 Investigating the need for dynamism in semantic web service composition. Section 3 discusses elaborately how this investigation was carried out.

6 Discussion and conclusions

The experimental study reveals that AI-based approaches have to be augmented with workflow and ontology-based approaches for ensuring automation and dynamism in planning phase of web service composition.

In the case study, automation support has been evidenced through fluent calculus-based plan generation. Dynamism is supported by ensuring the availability of services during execution.

Among various logic programming formalisms fluent calculus has been identified as the best candidate as it follows principal of progression. The planning phase of the composition greatly relies on fluent calculus for achieving automation through its set of axioms namely foundational, unique name, action-precondition, state update and auxiliary axioms. Fluent calculus-based dynamism is achieved through replanning based on QoS parameters. LTSA has been used for verifying the composed services. As this paper considers only sequence and split constructs of composition, the work can be extended for other complex constructs.

References


