6-2007

# Automatic Composition of Semantic Web Services Using Process Mediation

Zixin Wu

Karthik Gomadam
*Wright State University - Main Campus*

Ajith Harshana Ranabahu
*Wright State University - Main Campus*

Amit P. Sheth
*Wright State University - Main Campus*, amit@sc.edu

John A. Miller
*Wright State University - Main Campus*

# AUTOMATIC COMPOSITION OF SEMANTIC WEB SERVICES USING PROCESS MEDIATION

Zixin Wu[1], Karthik Gomadam[2], Ajith Ranabahu[1], Amit P. Sheth[2] and John A. Miller [1]
[1] *LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA.*
[2] *kno.e.sis center, Department of Computer Science and Engineering, Wright State University, Dayton, OH.*
{*zixin, ranabahu, jam*}*@cs.uga.edu,*{*amit.sheth, gomadam-rajagopal.2*}*@wright.edu*

Keywords: Semantic Templates, Process Mediation, Semantic Web Services, SAWSDL, SWS Challenge

Abstract: Web service composition has quickly become a key area of research in the services oriented architecture community. One of the challenges in composition is the existence of heterogeneities across independently created and autonomously managed Web service requesters and Web service providers. Previous work in this area either involved significant human effort or in cases of the efforts seeking to provide largely automated approaches, overlooked the problem of data heterogeneities, resulting in partial solutions that would not support executable workflow for real-world problems. In this paper, we present a planning-based approach to solve both the process heterogeneity and data heterogeneity problems. Our system successfully outputs a BPEL file which correctly solves a non-trivial real-world problem in the 2006 SWS Challenge.

## 1 Introduction

Web services are software systems designed to support interoperable machine-to-machine interactions over a network. They are the preferred standards-based way to realize Service Oriented Architecture (SOA) computing. A problem that has seen much interest from the research community is that of automated composition (i.e., without human involvement) of Web services. The ultimate goal is to realize Web service compositions or Web processes by leveraging the functionality of autonomously created services. While SOAs loosely coupling approach is appealing, it inevitably brings the challenge of heterogeneities across these independently developed services. Two key types of heterogeneities are those related to data and process. It is necessary and critical to overcome both types of these heterogeneities in order to organize autonomously created Web services into a process to aggregate their power.

Previous efforts related to Web service composition considered various approaches, and have included use of HTN (Sirin et al., 2004), Golog (Narayanan and Mcilraith, 2002), classic AI planning (Rao et al., 2006), rule-based planning (Ponnekanti and Fox, 2001) model checking (Traverso and Pistore, 2004), theorem proving (Rao et al., 2004) etc. Some solutions involve too much human effort; some overlook the problem of data heterogeneities. Overcoming both process and data heterogeneities is the key to automatic generation of executable process.‘

The way to measure the flexibility of a solution is to see how much human effort is needed if the scenario is changed. Our solution involves minimal human effort. Only the specification of the task, i.e., initial state and goal state of the task, has to be changed. We are assuming that all Web services are already semantically annotated.

In our solution, we extend GraphPlan(Russell and Norvig, 2003), an AI planning algorithm, to automatically generate the control flow of a Web process. Our extension is that besides the preconditions and effects of operations, we also take into consideration in the planning algorithm the structure and semantics of the input and output messages. This extension reduces the search space and eliminates plans containing operations with incompatible messages. Our approach for the problem of data heterogeneities is a data mediator which may be embedded in the middleware or an externalized Web service. Let us say that message M1 need to be converted into message M2 since they have different structure and/or semantics. The data

mediator takes as input the message M1 along with the semantically annotated schemas of M1 and M2, and then automatically gives as output the message M2. Our system continues to support loose coupling paradigm of SOA by separating the data mediation from the process mediation. This approach lets the process mediation system concentrate on generating the control flow, and it also makes it easier to analyze the control flow.

The key benefit of our solution is its ability to automatically generate executable workflow that addresses both control flow and data flow considerations (in our current implementation it is a BPEL process specification). In the process, we propose and implement (a) an extended GraphPlan algorithm, (b) a loosely coupled data mediation approach, (c) a context-based ranking algorithm for data mediation, and, (d) a pattern-based approach for loop generation in planning.

We demonstrate the above capabilities using a case/scenario in the 2006 SWS Challenge that has many real-world complexities. Our system generates a BPEL process automatically according to the specification of initial state, goal state, and semantically annotated Web service descriptions. The generated BPEL process can be executed successfully and accomplishes the desired task.

The remainder of this paper is organized as follows. We first give some background information of the problem of Web service composition in section 2, and then introduce a motivating scenario in section 3. The next two sections form the technical core of this paper– section 4 presents a formal definition of semantic Web services and Semantic Templates, and section 5 discusses the automatic Web service composition capability. The system architecture and implementation is briefly introduced in section 6, and the evaluation results are given in section 7. Finally, we give conclusions and future work in section 8.

# 2 Background and Related Work

## 2.1 Background

There are two categories of partners that are described within the Web services domain, namely the service provider and service requester. A service provider presents its Web service functionality by providing a set of operation specifications (or operations for short). These operations allow service requesters to use the services by simply invoking them. These operations might be inter-dependent. The dependences can be captured using precondition, effect, input, and output specifications of the operation. Using these available operations, a service requester performs one or more inter-related steps to achieve the desired goal. These steps can be best viewed as activities in a process and can be divided into smaller and more concrete sub-steps, and eventually invocations of concrete operations. Specifications by service requesters and providers are often times autonomously created. This causes heterogeneities to exist between the requester and provider when Web services need to interoperate as part of a composition of Web services. Two key types of heterogeneities may exist – the data related and the communication/process related. We say that process heterogeneity exists when the goal of the service requester cannot be achieved by atomically invoking exactly one operation once. On the other hand, data heterogeneity exists when the output message of an operation has different structure or semantics from the input message of the consecutive operation.

**SAWSDL:** We describe Web services and Semantic Templates (discussed next) in SAWSDL. SAWSDL (39, 2007) is a W3C standard to add semantics to Web services descriptions. SAWSDL does not specify a language for representing the semantic models, e.g., ontologies. Instead, it provides mechanisms by which concepts from the semantic models that are defined either within or outside the WSDL document can be referenced from within WSDL components as annotations. Semantic annotations facilitate process composition by eliminating ambiguities. We annotate a Web service by specifying Model References for its operations as well as Model References and Schema Mappings for the input and output message of its operations. We also extend SAWSDL by adding preconditions and effects as in our W3C submission on WSDL-S (Akkiraju et al., 2004) for an operation, which will be discussed in later sections.

**Semantic Templates**(Verma, 2006) is the way a service requester defines its task specifications. We again represent a Semantic Template in SAWSDL, in a manner very similar to Web service description, except that it is the specifications of a task, not of a specific Web service. We will discuss the formal model for Semantic Templates in a session 4.2.

## 2.2 Related Work

Rao et al. (Rao et al., 2006) discuss the use of the GraphPlan algorithm to successfully generate a process. While it is good to consider the interaction with the users, their approach suffers from the extent of automation. Also this work, unlike ours does not consider the input/output message schema when gener-

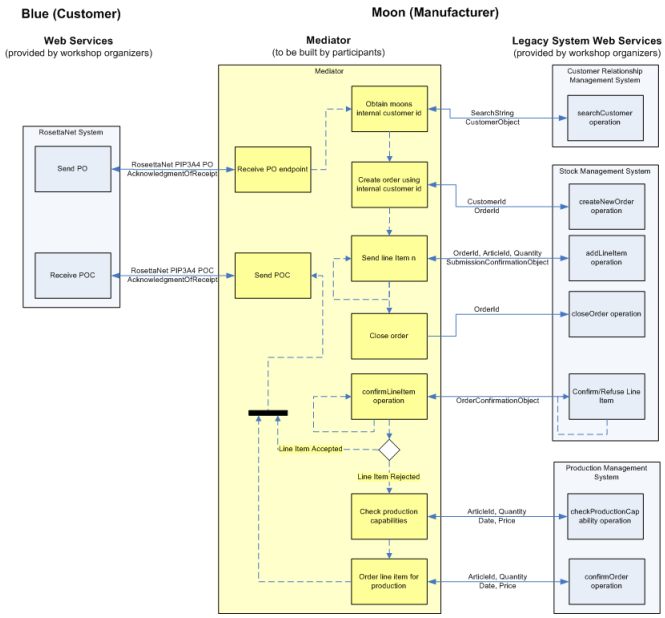


Figure 1: SWS Challenge Scenario

ating the plan, though their system does give alert of missing message to the users. This is important because an operation's precondition may be satisfied even when there is no suitable data for its input message. Another limitation of their work is that the only workflow pattern their system can generate is sequence, although the composite process may contain other patterns. As the reader may observe from the motivation scenario, other patterns such as loops are also frequently used.

Duan et al. (Duan et al., 2004) discuss using the pre and post-conditions of actions to do automatic synthesis of Web services. This is initiated by finding a backbone path. One weakness of their work is the assumption that task predicates are associated with ranks (positive integers). Their algorithm gives priority to the tasks with higher rank. However, this is clearly invalid if the Web services are developed by independent organizations, which is the common case and the main reason leading to heterogeneities.

Pistore et al. (Pistore et al., 2005) propose an approach to planning using model checking. They encode OWL-S process models as state transition systems and claim their approach can handle non-determinism, partial observability, and complex goals. However, their approach relies on the specification of OWL-S process models, i.e., the users need to specify the interaction between the operations. This may not be a realistic requirement in a real world scenario where multiple processes are implemented by different vendors.

# 3 Motivating Scenario

The 2006 SWS Challenge mediation scenario version 1 is a typical real-world problem where distributed organizations are trying to communicate with each others . A customer (depicted on the left side of the figure) desires to purchase goods from a provider (depicted on the right side of the figure). The anticipated process, i.e., the answer of this problem, is depicted on the middle of the figure which should be generated by a mediation system automatically. Both process and data heterogeneities exist in this scenario. For instance, from the point of view of the service requester called Blue, placing an order is a one-step job (send PO), while the service provider called Moon, involves four operations (searchCustomer, createNewOrder, addLineItem, and closeOrder). The message schemas they use are not exactly the same. For example, Blue uses fromRole to specify the partner who wants to place an order, while Moon uses billTo to mean the same thing. The structures of the message schemas are also different. To make matters worse, an input message may involves information from two or more output message, for example, the operation addLineItem requires information from the order request message by Blue and the newly created order ID from the output message of operation createNewOrder. In order to solve this problem successfully and automatically, the composition system at least should be able to do the following: generate the control flow of the mediator that involves at least

two workflow patterns (Sequence and Loop) based on the specification of the task and the candidate Web service(s), and convert (and combine if needed) an input message to an acceptable format annotated with appropriate semantics.

# 4 METEOR-S approach to semantic Web services

## 4.1 Abstract semantic Web service description

WSDL is a widely accepted industry standard (a W3C recommendation) for describing Web services. SAWSDL is expressive for functional and data semantics, and sufficient to solve the problem of semantic discovery and data mediation. We extend SAWSDL by adding preconditions and effects in the operations for process mediation. Preconditions and effects are necessary because not all the states of a Web service are represented by the input/output message. For example, both a book buying service and book renting service may take as the input the user ID and the ISBN, and give as the output the status succeed or fail. Importance of pre-condition and effects have been recognized by major semantic Web services initiatives including OWL-S, WSMO and WSDL-S, here we do that by extending the emerging standard of SAWSDL.

Formal model of abstract Web services: For the purpose of service composition, our model only focuses on the abstract representation of Web services, i.e., operations and messages, but does not consider the binding detail. Before giving our formal model, we need to introduce some definitions of the basic building blocks. Most classic AI planning problems are defined by the STRIPS representational language (or its variants like ADL), which divides its representational scheme into three components, namely, states, goals, and actions. For the domain of Web service composition, we extend the STRIPS language as the representational language of our method.

- **Extended state:** We extend a state by adding a set of semantic data types in order to ensure that the data for the input message of an operation is available before the operation is invoked. An extended state s has two components: s = <SSF, SDT >, where:
  - SSF is a set of status flags, each of which is an atomic statement with a URI in a controlled vocabulary. SSF defines the properties of the world in the specific state. We use ternary logic for status flags, thus the possible truth values are True, False, and Unknown. We use the open-world assumption, i.e., any status flag not mentioned in the state has the value unknown.
  - SDT is a set of semantic data types representing the availability of data. A semantic data type is a membership statement in Description Logic of a class (or a union of classes) in an ontology. An example state could be: $<\{$ orderComplete=True, orderClosed=False $\},\{$ ontology1#OrderID$(Msg_1)\} >$

  The reason why we use predicate logic for status flags is because it is simple for the user to specify the values of status flags in predicate logic, and computationally efficient. On the other hand, we use description logic for semantic data types because we need more expressive power to compare related messages, such as those with sub-class relationships.
- **Abstract semantic Web service** (Verma, 2006): Our definition of an abstract semantic Web service is built upon SAWSDL (39, 2007) An abstract semantic Web service SWS can be represented as a vector: $SWS = (sop_1, sop_2, , sop_n)$ Each sop is a semantic operation, which is defined as a 6-tuple: sop = <*op, in, out, pre, eff, fault*>where,
  - *op* is the semantic description of the operation. It is a membership statement of a class or property in an ontology.
  - *in* is the semantic description of the input message. It is a set of semantic data types, stating what data are required in order to execute the operation.
  - *out* is the semantic description of the output message. It is a set of semantic data types, stating what data are produced after the operation is executed.
  - *pre* is the semantic description of the precondition. It is a formula in predicate logic of status flags representing the required values of the status flags in the current state before an operation can be executed.
  - *eff* is the semantic description of the effect. It can be divided into two groups: positive effects and negative effects, each of which is a set of status flags describing how the status flags in a state change when the action is executed.
  - *fault* is the semantic description of the exceptions of the operation represented using classes in an ontology.

Table 1 illustrates an example of the representation of part of the Order Management System Web service described in our running scenario.

| $sop$ | $sop_1$ | $sop_2$ | $sop_3$ |
|---|---|---|---|
| $op$ | CreateNewOrder | AddLineItem | CloseOrder |
| $in$ | CustomerID | LineItemEntry,Order | OrderID) |
| $out$ | OrderID | AddItemResult | ConfirmedOrder |
| $pre$ | | orderComplete $\wedge$ orderClosed | orderComplete $\wedge$ orderClosed |
| $eff$ | negative:{orderComplete, orderClosed} | positive:{orderComplete} | positive: { orderClosed } |
| $fault$ | $sop_1$fault | $sop_2$fault | $sop_3$fault |

Table 1: Representation of Order Management System Web service

### 4.2 Semantic Template

While an abstract semantic Web service definition represents the operations and messages of a service provider, a Semantic Template models the requirement of the service requester. It is the way a service requester models the data, functional and non-functional specifications of a task. Formally semantic templates ($ST$ ) are defined by a collection of template terms.
$ST = \{sopt$ is a template term$\}$.
A template term $sopt = \{op, in, out, pre, eff, fault\}$ is a 6-tuple with:

- $op$: The operation
- $in$: The inputs to the operation
- $out$: The output of the operation
- $pre$:The pre conditions of the operations
- $eff$: The effects of the operation
- $fault$: The fault generated by this operation.

## 5 Automatic Web service composition

### 5.1 Formal definition of Web service composition

A semantic Web service composition problem involves composing a set of semantic Web services (SWSs) to fulfill the given requirements, or in our case a Semantic Template. Figure 2 illustrates our approach.

A semantic operation ($Operation_k$ in figure 2) has to be checked by the *satisfy* operator ($X$ in figure 2)against the current extended state before it can be added in the process specification. After it is added, a successor extended state is created by applying the *apply* (+ in figure 2) operator. We will give the formal definition of *satisfy* and *apply* operators below. For convenience, we use the following notations.

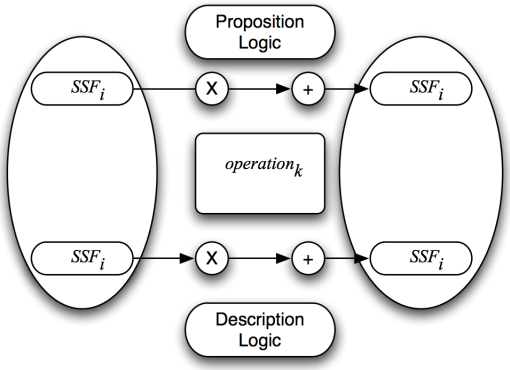


Figure 2: Business Process Levels

**Satisfy operator** is a function mapping an extended state $s_i$ and a semantic operation $sop_k$ to $T$ or $F$. Formally $textitsatisfy$ is defined as:

**Definition 1** *satisfy:* $(s_i, sop_k) \rightarrow \{T, F\}$
*This function maps to T (in such case,* $s_i$ *satisfies* $sop_k$ *and is written as:* $s_i \times sop_k$*) if and only if:*

- $\varepsilon(Pre(sop_k), SSF(s_i)) = True$, *where* $\varepsilon(f, v)$ *is an evaluation of formula* $f$ *based on the truth values in* $v$.
- $(Onto \cup SDT(s_i)) \models in(sop_k)$ , *where Onto is the ontology schema for semantic data types.*

That is, the precondition of $sop_k$ holds based on the truth values of the status flags in state $s_i$, and the semantic data types of $s_i$ together with the ontology schema entails the input of $sop_k$. For example, the following state satisfy the operation $sop_3$ in table 1:

$$<\{orderComplete = True, orderClosed = False\}, \{ontology1\#OrderID(Msg_x\}>$$

Here the semantic data type *OrderID* comes from an output message of any previous operation, or the initial message of the Semantic Template, so we put $Msg_x$ in the above example.

**Apply operator** is a function mapping an extended state si and a semantic operation $sop_k$ to a new extended state $s_j$. Formally this is defined as

| *Notation* | *Explanation* |
|---|---|
| *SSF(s)* | The set of status flags of extended state *s* |
| *Value(s)* | The truth value of a status flag *sf* in extended state *s* |
| *SDT(s)* | The set of semantic data types of extended state *s* |
| *in(sop)* | The input messages of semantic operation *sop* |
| *pre(sop* | The output messages of semantic operation *sop* |
| *eff(sop)* | The effect of semantic operation *sop* |
| *positive(eff)* | The positive effects of *eff* |
| *negative(eff)* | The negative effects of *eff* |

Table 2: Representation of Order Management System Web service

**Definition 2** *apply:* $(s_i, sop_k) \rightarrow s_j$
*Alternatively, we write* $s_i + sop_k \rightarrow s_j$ *This operator does the transition both on status flags and semantic data types.*

- *For status flags:*

$$\begin{aligned} \forall sf \in positive(eff(sop_k)), value(sf, s_j) &= True \\ \forall sf \in negative(eff(sop_k)), value(sf, s_j) &= False \\ \forall sf \in (eff(sop_k)), sf(s_j) &= sf(s_i) \end{aligned}$$

  *That is, a status flag in the positive effects is true in* $s_j$*, a status flag in the negative effects is false in* $s_j$*, while any status flag in* $s_i$ *but not in the effect is assumed to be unchanged in* $s_j$*.*
- *For semantic data types:* $SDT(s_j) = SDT(s_i) \cup out(sop_k)$ *That is, the semantic data types (membership statements) in* $s_j$ *are the union of the semantic data types in* $s_i$ *and the output of* $sop_k$*.*

As an example, if we apply the operation $sop_3$ in 1 to the state

$$<\{orderComplete = True, orderClosed = False\}, \{ontology1\#OrderID(Msgx)\}>$$

we will get a new state:

$$<\{orderComplete = True, orderClosed = True\}, \{ontology1\#OrderID(Msgx), ontology1\#ConfirmedOrder(sop_3 OutMsg)\}>$$

## 5.2 Composition of semantic Web services

We consider a SWS composition problem as an AI planning problem such that the semantic operation template defines the initial state and the goal state of the problem specification: **Initial state** is the extended state at the beginning of the process. It is defined by the precondition and initial message of the semantic operation template $\psi$.

$$s_0 = <ssf_0(sopt), in(sopt)>$$

**Goal state** is a requirement of the extended state at the end of the process. It is defined by the goal and output of sopt.

$$goalstate = <gl(sopt), out(sopt)>$$

**Composition of semantic Web services** is a function

$$swsc : (sopt, SWS_s) \rightarrow plan$$

Where,

- sopt is a semantic operation template.
- SWSs is the set of the semantic operations in the semantic Web services.
- *plan* is a DAG (Directed Acyclic Graph) of operations. Every topological sort of the DAG (say one of them is $sop_1, sop_2, , sop_n$) must conform to the following restrictions:
  - $s_0 \times <pre(sop_1), in(sop_1)>$
  - $s_0 + sop_1 \rightarrow s_1$
  - $s_{i-1} \times <pre(sop_i), in(s_i)>$
  - $s_{i-1} + sop_1 \rightarrow s_i$
  - $s_n \times goalstate$

That is, every topological sort of the plan must transform the initial state into the goal state by conforming to the *satisfy* and *apply* operators. Loops are generated in a post-process step that is explained at the of subsection 5.3.

## 5.3 Planning For Process Mediation

AI planning is a way to generate a process automatically based on the specification of a problem. Planners typically use techniques such as progression (or forward state-space search), regression (or backward state-space search), and partial-ordering. These techniques attempt to use exploration methods such as searching, backtracking, and/or branching techniques in order to extract such a solution. There are two basic operations in every state-space-based planning approach. First, the precondition of an action needs to

be checked to make sure it is satisfied by the current state before the operation can be a part of the plan. Second, once the operation is put into the plan, its effect should be applied to the current state and thus produce a consecutive state. We address the significant differences between classic AI planning and semantic Web service composition as follows:

1. Actions in AI planning can be described completely by its name, precondition, and effect, while Web services also include input and/or output message schema.
2. For AI planning, it is assumed that there is an agreement within an application on the terms in the precondition and effect. Terms with same name (string) mean the same thing, while terms with different name (string) mean different things. For example, in the famous block world scenario, if both block and box exist in the precondition/effect, they are treated as different things. This obviously does not carry over to the resources on the Web, thus it is necessary to introduce semantics in Web service composition.
3. More workflow patterns such as loops are desired in Web service composition. We address this problem by a pattern-based approach.

As discussed in the previous sections, both Web services and the specification of the task, i.e., Semantic Template are described in extended SAWSDL standard, so the terms in the precondition, effect, and input/output messages reach an agreement which is captured by the ontologies. For the first two types of differences we mentioned above, to apply AI planning techniques to semantic Web service composition, any state-space-based planning algorithm needs to be revised according to the following criteria.

1. State space should include status flags, as in the existing AI planning approaches, and semantic data types to represent the availability of data.
2. For each candidate action, besides checking its precondition against the status flags in the current state, it is also necessary to check its input message schema against the semantic data types in the current state. This reduces the search space and eliminates plans containing operations whose input message is unavailable in the state.
3. Since the states and the actions/operations are semantically annotated by referring to ontologies, the checking in the previous step involves reasoning based on the ontologies, not just comparing the name of the terms.
4. Once an action/operation is added into the plan, not only the status flags are updated by applying the effect, the semantic data types should also be updated by put a new semantic data type based on the output message schema.

## 5.4 Extended GraphPlan Algorithm

Although most AI planning algorithms are suitable for the task here, we use GraphPlan algorithm (Russell and Norvig, 2003). It is sound and complete thus we can always construct correct plans if there exist any, and its compact representation of the states makes it space efficient while doing a breadth-first style search. It also uses mutex links to avoid exploring some irrelevant search space. Like other classical AI planning algorithms, GraphPlan only considers the precondition and effect of actions, thus does not take into account the input/output message of actions. Our approach requires an extension of the algorithm to accommodate the semantic data types defined above. An operation may only be added in the next action level when its preconditions hold based on the current state level of the planning graph and the data types of the input message of the operation can be entailed by the union of ontology and the current state level. When an operation is placed in the next action level, its effects as well as output data types are applied to the current state level, and thus produce the next state level. Afterwards, mutex links between actions must be evaluated and placed so that they may be used when backtracking through the graph for the solution. Note that the creation of the mutex links should also consider the semantic data types accordingly.

## 5.5 Pattern-Based Approach For Loop Generation

GraphPlan algorithm may generate plans only with sequence and AND-split workflow patterns (van der Aalst and Hofstede, 2002). However, loops are also a frequently used pattern. Loop generation (or iterative planning) itself is a difficult and open problem in AI. Much work on iterative planning is based on theorem-proving (Biundo, 1994). It is believed by Stephan and Biundo (Stephan and Biundo, 1995) and other researchers that iterative planning cannot be carried out in a fully automatic way. (Levesque, 2005) proposes a new way that is not tied to proving a theorem, but it is only correct for a given bound or a certain class of simple planning problems. Here we proposed a pattern-based approach for loop generation. It is based on the observation of frequently used patterns of iterations. For example, in the motivation scenario, the order request includes multiple line

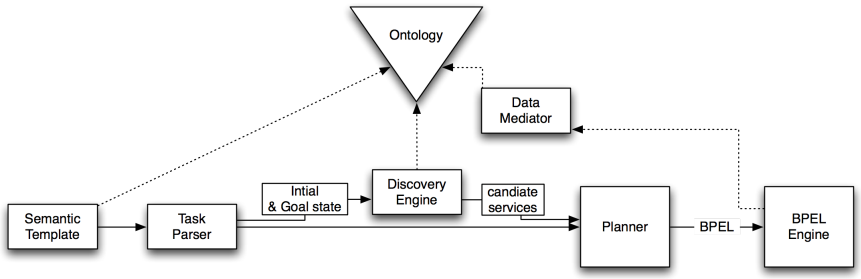


Figure 3: System Architecture

items (an array of line items) while the addLineItem operation takes as input only one line item. It is obvious that the process needs to iterate all the line items in the order request. We may extract the pattern as follows. If an operation has an input message including an element with semantic annotation $SDT_i$ and attribute maxOccurs in XML Schema whose value is 1, while the matched (see satisfy operator) semantic data type in the current state is from an output message where the corresponding element in that message has maxOccurs with value unbounded or greater than 1, then a loop is needed for this operation to iterate the array. Our approach avoids the computationally hard problem by restricting possible patterns of loops. The limitation is that the patterns need to be identified and put in the code beforehand.

## 6 Implementation and System Architecture

Figure 3 is the overview of our implemented system. We implement the system in Java, and use Jena to handle the ontology . We develop our SAWSDL API (39, 2007) to parse Semantic Templates and annotated Web service descriptions. We use IBM BPWS4J API to generate BPEL, and run it on Oracle BPM engine.

## 7 Conclusions and Future Work

This paper presents an automatic approach for Web service composition, while addressing the problem of process heterogeneities and data heterogeneities by using a planner and a data mediator. Specifically, an extended GraphPlan algorithm is employed to generate a BPEL process (the currently supported workflow patterns are sequence, AND-split and loop) based on the task specification (Semantic Template) and candidate Web services described in SAWSDL. Data mediation can be handled by assignment activities in the BPEL, or by a data mediator which may be embedded in a middleware or an externalized Web service. While the BPEL process is running, it calls the data mediator to convert (and combine if necessary) the available messages into the format of the input message of an operation which is going to be invoked. A context-based ranking algorithm is employed in the data mediator to select the best element from the source messages if more than one element has acceptable semantics for the target element. Our experiment shows that our systems solved the problem in SWS challenge 2006 mediation scenario successfully, which is a non-trivial challenging problem that involves process and data heterogeneities. We consider our approach to be highly flexible, since the only thing a user need to change for a new scenario is the task specification (Semantic Template). Our future work includes supporting more workflow patterns especially OR-Split, the propogation/scopes of semantic data types in messages, and non-functional semantics.

## REFERENCES

(2007). Semantic annotations for wsdl and xml schema. Technical report, W3 Consortium.

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A. P., and Verma, K. (2004). Web service semantics – wsdl-s. Technical report, LSDIS Lab and IBM Corporation.

Biundo, S. (1994). Present-day deductive planning. In Bäckström, C. and Sandewell, E., editors, *Current Trends in AI Planning: Proceedings of the 2nd European Workshop on Planning (EWSP-93)*, pages 1–5, Vadstena, Sweeden. IOS Press (Amsterdam).

Duan, Z., Bernstein, A. J., Lewis, P. M., and Lu, S.

(2004). A model for abstract process specification, verification and composition. In *ICSOC*, pages 232–241.

Levesque, H. J. (2005). Planning with loops. In *IJCAI*, pages 509–515.

Narayanan, S. and Mcilraith, S. A. (2002). Simulation, verification and automated composition of web services. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA. ACM Press.

Pistore, M., Traverso, P., Bertoli, P., and Marconi, A. (2005). Automated synthesis of composite bpel4ws web services. In *ICWS*, pages 293–301.

Ponnekanti, S. R. and Fox, A. (2001). Sword: A developer toolkit for web service composition.

Rao, J., Dimitrov, D., Hofmann, P., and Sadeh, N. (2006). A mixed initiative approach to semantic web service discovery and composition: Sap's guided procedures framework. In *ICWS*, pages 401–410.

Rao, J., Küngas, P., and Matskin, M. (2004). Logic-based web services composition: From service description to process model. In *ICWS*, pages 446–453.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education.

Sirin, E., Parsia, B., Wu, D., Hendler, J., and Nau, D. (2004). Htn planning for web service composition using shop2. *Journal of Web Semantics*, 1(4):377–396.

Stephan, W. and Biundo, S. (1995). Deduction-based refinement planning. Technical Report RR-95-13, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Erwin-Schrödinger Strasse
Postfach 2080
67608 Kaiserslautern
Germany.

Traverso, P. and Pistore, M. (2004). Automated composition of semantic web services into executable processes.

van der Aalst, W. and Hofstede, A. (2002). Yawl: Yet another workflow language.

Verma, K. (2006). *Configuration And Adaptation of Semantic Web Processes*. PhD thesis, University of Georgia.