Cite as: Technical Report, Kno.e.sis Center, Wright State University, February 28, 2007 *or*
Technical Report, LSDIS lab, University of Georgia, February 28, 2007.

Automatic Composition of Semantic Web Services using Process and Data Mediation

Zixin Wu[1], Ajith Ranabahu[1], Karthik Gomadam[2], Amit P. Sheth[2], John A. Miller[1]
[1] LSDIS Lab, University of Georgia, Athens, Georgia, USA.
[2] Kno.e.sis Center, Wright State University, Dayton, Ohio, USA.
{zixin, ajith, jam}@cs.uga.edu, {karthik.rajagopal.2, amit.sheth}@wright.edu

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 an executable BPEL file which correctly solves non-trivial real-world process specifications outlind in the 2006 SWS Challenge.

1. Introduction

Web services are software systems designed to support interoperable machine-to-machine interaction 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 SOA's 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 [1, 2], Golog [3-5], classic AI planning [6], rule-based planning [7, 8], model checking [9-11], theorem proving [12-15], 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. Fortunately SAWSDL, a standard for annotating Web services largely based on our input to W3C is a candidate recommendation, and open source tools for annotation have also been developed, such as [16] and the SAWSDL editor in [17].

In our solution, we extend GraphPlan[18], 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 to handle different structure and/or semantics. Our approach continues to support loose coupling paradigm of SOA by separating the data mediation from the process mediation. Thehe process mediation system concentrate on generating the control flow, making it easier to analyze the control flow.

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 an executable BPEL process automatically according to the specification of initial state, goal state, and semantically annotated Web service descriptions in SAWSDL, now a W3C candidate recommendation.

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

Background

There are two categories of partners that are described within the Web services domain, namely the service provider and service requester. [1] 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 oftentimes 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.

We describe Web services and Semantic Templates (discussed next) in SAWSDL [19], the W3C candndate recommendation 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." [2] 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 [20] for an operation, which will be discussed in later sections. The need for precondition and effects had also been recognized by several Semantic Web Service (SWS) specifications, including OWL-S [21] and WSML [22], but left out in the first version of SAWSDL as a practical matter of reaching agreement on a baseline specification.

A Semantic Template [23] is the way a service requester defines its task specifications. We 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. The formal model for Semantic Templates appears in sec. 4.2.

2.2 Related work

Rao et al. [6] discuss the use of the GraphPlan algorithm to successfully generate a process. It relies on interaction with the users, and hence provide limited support for automation. Also this work, unlike ours does not consider the input/output message schema when generating 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.

Cardoso et al. [24] focus on the discovery of Web service objects and resolution of structural and semantic heterogeneity of a manually created workflow whose activities are Web service templates, i.e., not concrete Web service operations. In that wprk, users have to design the workflow, though on an abstract level.

Ziyang et al. [25] 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 often the case and a reason contributor to heterogeneities.

A correlation between Hierarchical Task Network (HTN) planning and Web service representation in the OWL-S framework is discussed in [21]. HTN planning uses the approach of refining plans by applying action, or task decompositions. Their strategy is to divide high-level tasks into smaller sub-tasks until primitive, atomic tasks that can be performed directly are reached. The benefit of this approach is the reduction in the complexity of planning for tasks that require many actions. However, the mechanism of dividing high-level tasks itself is problematic. If human intervention is needed, it defeats the whole notion of automation.

[1] "Web Services Glossary" (http://www.w3.org/TR/ws-gloss/), and the discussion of terminologies (http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#wordonspr).

[2] http://www.w3.org/2002/ws/sawsdl/

Pistore et al. [11] 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.

WSMO group [26] refers to the problem of process mediation as orchestration. A graphical tool in [27] is presented to guide the user to compose a process, but no additional computational support or automation is present.

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[1]. 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.

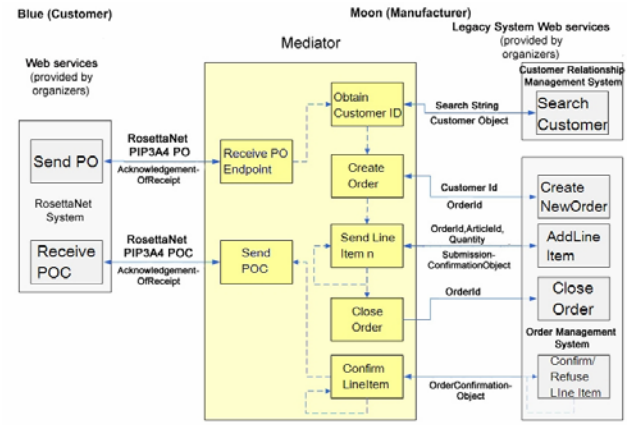

Figure 1. SWS Challenge mediating scenario

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. Recently, the W3C also come up with a candidate recommendation for Semantic Anotation of WSDL -- SAWSDL [28] that is largely based on WSDL-S, our W3C member input with IBM [20]. 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

[1] The reader may find the detail on "Challenge on Automating Web Services Mediation Choreography and Discovery" http://sws-challenge.org/wiki/index.php/Scenario:_Purchase_Order_Mediation.

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*.[1]
- *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** [29]. Our definition of an abstract semantic Web service is built upon SAWSDL [19] and WSDL-S [20]. 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 defined as a 6-tuple:

sop = <*op, in, out, pre, eff, fault*>

- *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.

[1] For convenience, in the following examples we do not explicitly write the status flags whose values are unknown.

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

Table 1. Representation of Order Management System Web service

sop	sop$_1$	sop$_2$	sop$_3$
op	CreateNewOrder	AddLineItem	CloseOrder
in	CustomerID(sop$_1$InMsg)	LineItemEntry(sop$_2$InMsg) OrderID(sop$_2$InMsg)	OrderID (sop$_3$InMsg)
out	OrderID(sop$_1$OutMsg)	AddItemResult (sop$_2$OutMsg)	ConfirmedOrder (sop$_3$OutMsg)
pre		¬ orderComplete ∧ ¬ orderClosed	orderComplete ∧ ¬ orderClosed
eff	negative:{orderComplete, orderClosed}	positive:{orderComplete}	positive:{orderClosed}
fault	sop$_1$Fault	sop$_2$Fault	sop$_3$Fault

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 [2] specifications of a task.

Formal model of Semantic Template

A Semantic Template (ST) can also be represented as a vector:

ST = (*sopt$_1$, sopt$_2$,…, sopt$_n$*)

Each *sopt* is a semantic operation template, which is defined as a 6-tuple:

sopt = <*op, in, out, ssf$_0$, gl, fault*>

- *op* is the semantic description of the operation template. It is a membership statement of a class or property in an ontology.
- *in* is the semantic description of the initial message. It is a set of semantic data types stating what data are available at the beginning of the process.
- *out* is the semantic description of the output message. It is a set of semantic types stating what data are required at the end of the process.
- *ssf$_0$* is the semantic description of the initial status flags. It is a set of status flags as in an extended state.
- gl is the semantic description of the goal. It is a formula of status flags.

The following table shows the representation of a Semantic Template *SendPO* from the scenario in sec. 2.

[2] For more information about non-functional semantics, please refer to some previous work such as 30.Siddharth Bajaj, V., et al. *Web Services Policy 1.2 - Framework (WS-Policy)*. 2006 [cited; Available from: http://www.w3.org/Submission/WS-Policy/..

Table 2. An example of Semantic Template

sopt	$sopt_1$
op	SendPO
in	OrderInfo ($sopt_1$InMsg)
out	Acknowledgement ($sopt_1$OutMsg)
ssf_0	
gl	orderComplete ∧ orderClose
fault	$sopt_1$Fault

4.3 Semantic discovery

Semantic discovery is the process of discovering services based on the semantic metadata attached with the services. The proposed composition framework uses the METEOR-S Web Service Discovery Infrastructure (MWSDI) [31] for discovering candidate services. MWSDI extends the basic UDDI [32] data structures to capture the data and functional semantics. MWSDI is built on top of UDDI4J and jUDDI registry framework. In addition to supporting model reference based discovery of services, MWSDI also provides reasoning capabilities based on subsumption and equivalence. This allows for selecting the candidate services based on data and functional semantics. Given a Semantic Template as an input, MWSDI returns a set of services which meet the data and functional requirements modeled in the Semantic Template.

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.

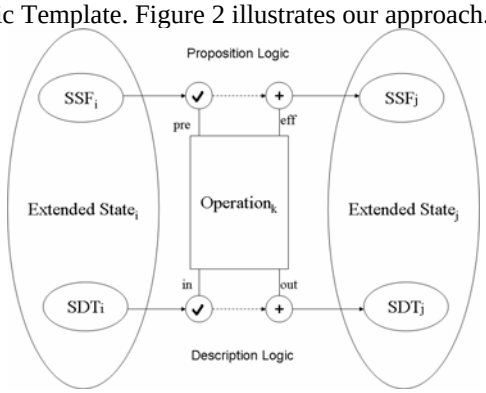

Figure 2. Semantic Web service composition

A semantic operation (O*peration*$_k$ in the diagram) needs to be checked by the *satisfy* operator (the check mark in the figure) 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* operator (the plus mark in the diagram). We will give the formal definition of *satisfy* and *apply* operators below. For convenience, we use the following notations.

Table 3. Notations used in definitions

Notation	Explanation
SSF(s)	The set of status flags of extended state *s*
value(sf, 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*
out(sop)	The output messages of semantic operation *sop*
pre(sop)	The precondition 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*
in(sopt), out(sopt), ssf_0(sopt), gl(sopt)	The same fashion as the above notations, but applied to Semantic Templates

***Satisfy* operator.** It is a function mapping an extended state s_i and a semantic operation sop_k to *T* or *F*:

$satisfy: (s_i, sop_k) \mapsto \{T, F\}$

This function maps to *T* (in such case we call it "s_i satisfies sop_k" and write it as: s_i ✓sop_k) if and only if:

- $\mathcal{E}(Pre(sop_k), SSF(s_i)) = True$, where $\mathcal{E}(f, v)$ 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. [1]

***Apply* operator.** It is a function mapping an extended state s_i and a semantic operation sop_k to a new extended state s_j:

$apply: (s_i, sop_k) \mapsto s_j$

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:

$\forall sf \in positive(eff(sop_k)), value(sf, s_j) = True$

$\forall sf \in negative(eff(sop_k)), value(sf, s_j) = False$

$\forall sf \notin (eff(sop_k)), sf(s_j) = sf(s_i)$

[1] The scope of messages in workflow is part of our future work.

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 Table 1 to the state:

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

we will get a new state:

$<\{orderComplete=True, orderClosed=True\}, \{ontology1\#OrderID(Msg_x), ontology1\#ConfirmedOrder(sop_3OutMsg)\}>$

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. It is the extended state at the beginning of the process. It is defined by the precondition and initial message of the semantic operation template *sopt*.

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

Goal state. A 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:

$swsc: (sopt, SWSs) \mapsto plan$

Where,

- *sopt* is a semantic operation template.
- *SWS*s 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:

  $s_0 \checkmark <pre(sop_1), in(sop_1)>$

  $s_0 + sop_1 \rightarrow s_1$

  $s_{i-1} \checkmark <pre(sop_i), in(s_i)>$

  $s_{i-1} + sop_1 \rightarrow s_i$

  $s_n \checkmark goalstate$

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, which will be explained at the end of the next sub-section[1].

5.2 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 Loop 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 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] In order to generate OR-Split workflow pattern, the problems of non-determinism and partial observability need to be addressed. The reader may refer to other approaches for alternative solutions, such as 33. Doshi, P., et al., *Dynamic Workflow Composition Using Markov Decision Processes.* JWSR, 2005. **2**(1): p. 1-17.

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.

Extended GraphPlan algorithm. Although most AI planning algorithms are suitable for the task here, we use GraphPlan algorithm [18]. 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 algorithm, GraphPlan only considers the precondition and effect of actions, thus does not takes 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.

Pattern-based approach for loop generation. GraphPlan algorithm may generate plans only with sequence and AND-split workflow patterns [34]. 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 [35]. It is believed by Stephan and Biundo [36] and other researchers that iterative planning cannot be carried out in a fully automatic way. [37] 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 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.

5.3 Data mediation.

Most of the previous work in this area focused on the generation of the control flow hence overlooked the problem of data heterogeneities and assumed there are no such problems or it is handled automatically in an unspecified way. We consider data mediation as critical for generating executable workflows for real-world problems. To be more intuitive, let us say that we need to convert a message M_1 with schema MS_1 into a message M_2 with schema MS_2, and let us call M_1 the source message, M_2 the target message, MS_1 the source schema, and MS_2 the target schema.

We discuss different types of message-level heterogeneities, including syntactic, structural, model/representational, and semantic heterogeneities in [38]. We need to focus on structural and semantic heterogeneities, as the XML based environment adresses the syntactic heterogeneities.

Semantic heterogeneities in message schema means that terms with different names may refer to the same concept, or terms with the same name may refer to different concepts. The solution is to annotate the message schema by using ontological concepts, thus making sure different Web services reach an agreement on the semantics of the terms.

In [38], we address the problem of structural heterogeneities in message schema by having the developer associate mappings using the Schema Mapping on Web service message (input and output) elements. The source message is transformed to a format identified by an OWL concept to which it is mapped by the "up cast"

attribute ("liftingSchemaMapping" in SAWSDL), and then transformed to the target message format by the "down cast" attribute ("loweringSchemaMapping" in SAWSDL). The ontologies become a vehicle through which Web services resolve their message level heterogeneities.

We adopt this approach, and also consider the situation where Schema Mapping is unavailable in a given SAWSDL file. We created an algorithm to convert the source message into a target message. The basic idea is that we traverse the target schema tree (or DAG) in a top-down direction, and try to fill up each node by using the data in the source message. Let us say that we are currently handling the node N_t in the target schema. N_t is filled up if one of the following happens:

- N_t has the annotation of Schema Mapping and there is another node in the source schema who also has a Schema Mapping and whose Model Reference refers to a class which entails the class of N_t's Model Reference, thus we assume that the node in the source can be converted into the target format according to the Schema Mapping and we do not look into the sub-tree of N_t anymore.
- N_t is a leaf and there is another leaf in the source schema whose Model Reference refers to a class which entails the class of N_t's Model Reference.
- All the nodes in the sub-tree of N_t is filled up.
- N_t is allowed to be empty in the target message.

Context-based ranking algorithm. In case more than one node in the source message(s) is suitable for a node in the target message, we have the following context-based ranking algorithm to select the best one automatically according to the usage context of the nodes. This is necessary because an XSD element may refer to another element by using "ref" attribute. For example,

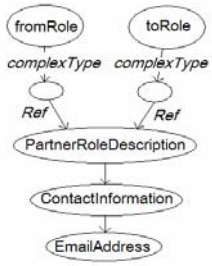

Figure 5. Semantic difference because of different context

Suppose the node in the target schema has annotation referring to an ontology class which is the same as the annotation of the "EmailAddress" in this example, by only comparing the annotation of "EmailAddress", we cannot decide which node in the source message is better. Thus we need to also look at the Model Reference of their ancestors in order to get the most accurate meaning of the "EmailAddress". Then the problem becomes comparing the similarity of the XPaths formed by the ancestors and the nodes themselves. The algorithm is a variant of edit distance algorithm using the dynamic programming technique.

```
function getXpathSim (srcXpath, tarXpath)
  srcArray ← extractElement(srcXpath)
  tarArray ← extractElement(tarXpath)
  for i ← 1 to length(srcArray)
    for j ← 1 to length(tarArray)
      sim[i][j] = 0
      elementSim ← compareElement(srcArray[i], tarArray[j])
      x1 ← (1-fadingFactor) * sim[i-1][j-1] + elementSim
      x2 ← (1-fadingFactor) * sim[i-1][j]
      x3 ← (1- fadingFactor) * sim[i][j-1]
      x ← max(x1, x2, x3)
      if x > sim[i][j]
        sim[i][j] ← x
```

Figure 6. Context-based data type ranking algorithm

Where "fadingFactor" is to give more weight to the elements near the current one, while make the ancenstors far away less important. This program calls the function "compareElement" to calculate the similarity of two XSD elements. In this function, if both elements have Model References, it only compares their Model References, i.e., return 1 if the class of the source's Model Reference can entail the class of target's; return 0 otherwise. If either element does not have a Model Reference, it comparess their names by using certain string comparison algorithm, and assigns it a predefined weight. The value is accepted only if it is above a predefined threshold. The user also has the choice to disable the name comparison by setting a parameter in the system.

For the example in Figure 5, if the target XPath is "billTo/email", where "billTo" is an equivalent class of "fromRole" and "email" is an equivalent class of "EmailAddress", our system gives score 0.252 and 0.16666667 to the left and right XPath in Figure 5 respectively, thus successfully selects the best matched XPath.

Data mediator. Although data mediation can be handled by a set of assignment activities in a BPEL process, we use a loosely coupled component called data mediator in our system to handle this problem. A data mediator may be embedded in a middleware, or it can be an externalized Web service. In the experiment, we deploy the above data mediation program as a dedicated Web service which converts and combines messages at run-time, thus alleviating the burden of data mediation from the generated process and make it easier to analyze the control flow. This loose coupling promotes reusability and facilitates dynamic partner binding, especially at run-time.

6. Implementation and system architecture

Figure 7 is the overview of our implemented system. We implement the system in Java, and use Jena to handle the ontology[1]. We develop our SAWSDL API [39] to parse Semantic Templates and annotated Web service descriptions. We use IBM BPWS4J API to generate BPEL, and run it on Oracle BPM engine.

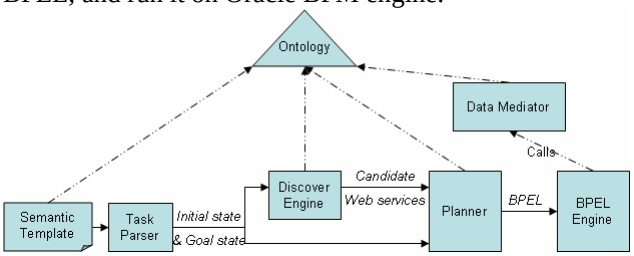

Figure 7. System architecture

7. Evaluation

Our system generates a BPEL file according to the Semantic Template we created (Table 2). We ran it on Oracle BPM engine, and part of the graphical result is in Figure 8. It placed an order successfully as we see the record in our account in the 2006 SWS challenge server. The only thing we cannot do is the "confirmLineItem" operation, as it uses a Solicit Response message pattern which is not supported by BPEL.

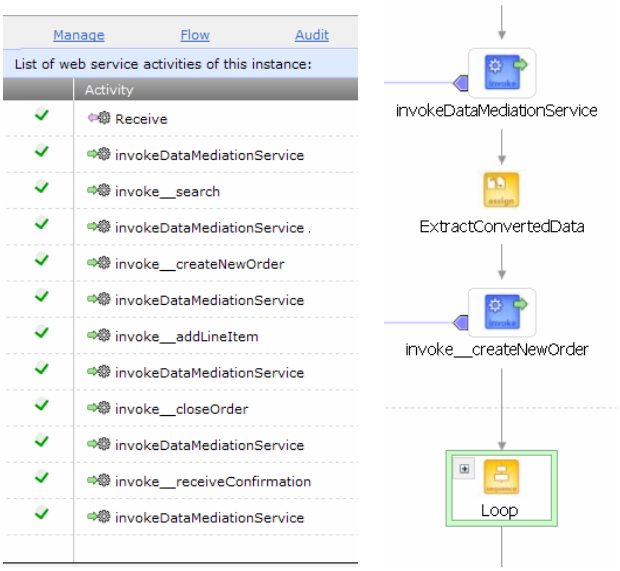

Figure 8. Part of the generated BPEL process and invocation results

8. 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 such as WS-Policy [30].

[1] Note that if more than one ontology is involved, ontology matching/mapping is needed.

9. References

1. Sirin, E., et al., *HTN Planning for Web Service Composition Using SHOP2.* Web Semantics Journal, 2004. **1**(4): p. 377-396.
2. Sirin, E., B. Parsia, and J. Hendler, *Template-based composition of semantic web services*, in *AAAI fall symp on agents and the semantic web*. 2005: Virginia, USA.
3. Narayanan, S. and S.A. McIlraith. *Simulation, verification and automated composition of Web service.* in *The 11th International World Wide Web Conference*. 2002. Honolulu, Hawaii, USA.
4. McIlraith, S.A., T.C. Son, and H. Zeng, *Semantic Web Services.* IEEE Intelligent Systems, 2001. **16**(2): p. 46-53.
5. McIlraith, S. and T.C. Son. *Adapting Golog for composition of Semantic Web services*. in *Knowledge Representation and Reasoning (KR2002)*. 2002. Toulouse, France.
6. Rao, J., et al., *A Mixed Initiative Approach to Semantic Web Service Discovery and Composition: SAP's Guided Procedures Framework*, in *The IEEE Intl Conf on Web Services (ICWS'06)*. 2006..
7. Ponnekanti, S.R. and A. Fox, *SWORD: A Developer Toolkit for Web Service Composition*,

in *The 11th World Wide Web Conference* 2002: Honolulu, Hawaii, USA.
8. Medjahed, B., A. Bouguettaya, and A.K. Elmagarmid, *Composing Web services on the Semantic Web*. VLDB Journal, 2003. **12**(4)..
9. Kuter, U., et al. *A Hierarchical Task-Network Planner based on Symbolic Model Checking*. in *The International Conference on Automated Planning & Scheduling (ICAPS)*. 2005.
10. Traverso, P. and M. Pistore. *Automated Composition of Semantic Web Services into Executable Processes*. in *The 3rd International Semantic Web Conference (ISWC2004)*. 2004.
11. Pistore, M., et al. *Automated Synthesis of Composite BPEL4WS Web Services*. in *IEEE Intl Conference on Web Services (ICWS'05)*. 2005.
12. Waldinger, R.J., *Web Agents Cooperating Deductively*, in *Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers*. 2001, Springer-Verlag.
13. Lämmermann, S., *Runtime Service Composition via Logic-Based Program Synthesis*, in *Department of Microelectronics and Information Technology*.2002, Royal Institute of Technology.
14. Rao, J., P. K¨ungas, and M. Matskin. *Application of Linear Logic to Web Service Composition*. in *the 1st Intl Conf on Web Services*. 2003.
15. Rao, J., P. Kungas, and M. Matskin. *Logic-based Web services composition: from service description to process model*. in *The 2004 Intl Conf on Web Services*. 2004. San Diego, USA.
16. Gomadam, K., et al., *Radiant: A tool for semantic annotation of Web Services*, in *4th Intl Semantic Web Conf (ISWC 2005)*. *http://lsdis.cs.uga.edu/projects/meteor-s/SAWSDL/*
17. *WSMO Studio*. [cited; Available from: http://www.wsmostudio.org/.
18. Russell, S. and P. Norvig, *Artificial Intelligence: A Modern Approach, Prentice Hall, 2006*.
19. Farrell, J. and H. Lausen, *Semantic Annotations for WSDL*. http://www.w3.org/TR/sawsdl/, 2006.
20. Akkiraju, R., et al., *Web Service Semantics-WSDL-S*, W3C Member Submission, Nov. 2005. http://www.w3.org/Submission/WSDL-S/
21. Coalition, O.S. *OWL-S: Semantic markup for web services*. 2003 [cited; Available from: http://www.w3.org/Submission/OWL-S/.
22. *ESSI WSML working group*. [cited; Available from: http://www.wsmo.org/wsml/.
23. Sivashanmugam, K., et al., *Adding Semantics to Web Services Standards*, in *International Conference on Web Services (ICWS'03)*.
24. Cardoso, J. and A. Sheth, *Semantic e-Workflow Composition*. Journal of Intelligent Information Systems, November 2003. **21**(3): p. 191-225.
25. Duan, Z., et al. *A Model for Abstract Process Specification, Verification and Composition*. in *The 2nd Intl conf on Service oriented computing*. 2004..
26. Roman, D., et al., *Web Service Modeling Ontology*. Applied Ontology, **1**(1): 2005.
27. Farshad, H., et al., *Semantic Web Service Composition in IRS-III: The Structured Approach*, in *Proc. of the 7 IEEE Intl Conf on E-Commerce Technology (CEC'05*, 2005.
28. Farrell, J. and H. Lausen. *Semantic Annotations for WSDL*. http://www.w3.org/TR/sawsdl/.
29. Verma, K., *Configuration and Adaptation of Semantic Web Processes*, PhD Dissertation, *Computer Science*. 2006, University of Georgia: Athens.
30. Siddharth Bajaj, V., et al. *Web Services Policy 1.2 - Framework (WS-Policy)*. 2006 http://www.w3.org/Submission/WS-Policy/.
31. Gomadam, K., K. Verma, and A. Sheth, *Of keywords, port types and semantics: A Journey into the land of Web services discovery*, in *Semantic Web Processes and Their Applications*, J. Cardiso and A. Sheth, Editors. 2006, Springer.
32. *Universal Description Discovery and Integration (UDDI)*. http://www.uddi.org/.
33. Doshi, P., et al., *Dynamic Workflow Composition Using Markov Decision Processes*. JWSR, 2005. **2**(1): p. 1-17.
34. Aalst, W.M.P.v.d. and A.H.M.t. Hofstede, *YAWL: yet another workflow language*. Information Systems, 2005. **30**(4): p. 245-275.
35. Biundo, S. *Present-day deductive planning*. in *The 2nd European Workshop on Planning (EWSP-93)*. 1994.
36. Stephan, W. and S. Biundo. *Deduction-Based Refinement Planning*. in *AIPS*. 1996.
37. Levesque, H.J. *Planning with Loops*. in *The 19th International Joint Conference on Artificial Intelligence*. 2005. Edinburgh, Scotland.
38. Nagarajan, M., et al., *Semantic Interoperability of Web Services - Challenges and Experiences*, in *IEEE Inl Conf on Web Services (ICWS 2006)*.
39. *SAWSDL4J*. http://cs.uga.edu/~ranabahu/sawsdl4j/.