A Declarative Approach using SAWSDL and Semantic Templates Towards Process Mediation

Karthik Gomadam[1], Ajith Ranabahu[1], Zixin Wu[2], Amit P. Sheth[1] and John Miller[3]

[1] kno.e.sis center, Wright State University, Dayton, OH
kgomadam@gmail.com,{ranabahu.2,amit.sheth}@wright.edu
[2] Nextag Corporation, San Mateo, CA
wuzixin@gmail.com
[3] LSDIS Lab, Department of Computer Science, University of GA, Athens, GA
jam@cs.uga.edu

Summary. In this paper we address the challenges that arise due to 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. We adopt a declarative approach to capture the partner specifications external to the process and demonstrate the usefulness of this approach in adding more dynamism to Web processes. Our system successfully outputs a BPEL file which correctly solves a non-trivial real-world problem in the SWS Challenge.

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

6.1 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 [1], Golog [2], classic AI planning [3], rule-based planning [4] model checking [5], theorem proving [6] 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.

One of the metrics for evaluating the solution is to measure the flexibility of a solution. The objective is to minimize the human effort required when the execution scenario is modified. We adopt a declarative approach towards achieving a greater degree of flexibility. Our solution externalizes the partner and the process requirements from the process control flow. Various variable parameters including QoS requirements, data and functional requirements are specified using semantic templates in a declarative manner. In the event of a change in the environment, one can reconfigure the process by changing the external specification.

In our solution, we extend GraphPlan[7], 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 heterogeneity is a data mediator which may be embedded in the middleware or an externalized Web service. By separating the data mediation from the process mediation, we allow the process mediation system concentrate on generating the control flow. This separation of concerns also makes it easier to analyze the control flow.

The key benefits of our solution are

1. The 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).
2. A pattern-based approach for loop generation in planning.
3. A loosely coupled data mediation approach and a context-based ranking algorithm for data mediation.
4. Declarative approach towards specifying requirements that makes it easier to manage change.

We demonstrate the above capabilities using a case/scenario in the 2006 SWS Challenge.

The remainder of this paper is organized as follows. We first give some background information of the problem of Web service composition in section 6.2, and then introduce a motivating scenario in section 6.3. The next two sections form the technical core of this paper–section 6.4 presents a formal definition of semantic Web services and Semantic Templates, and section 6.6 discusses the automatic Web service composition capability.

Finally, we give conclusions and future work in section 6.7.

6.2 Background and Related Work

6.2.1 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.

It is also important that we use a framework that has the flexibility to support both functional and non-functional requirements for service discovery. Such a framework must allow service providers to publish their non-functional capabilities. In such a framework the criteria for selection must not be too restrictive, since it may be very difficult to find services that exactly match the requirements. The requester must be able to specify the expected level of match for the different aspects of the request. For example, a requester can specify that an exact match is needed with respect to the operation while a *sufficiently similar* match would suffice for the input and output parameters.

SAWSDL

We describe Web services and Semantic Templates (discussed next) in SAWSDL. SAWSDL [8] 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 [9] for an operation, which will be discussed in later sections.

Rao et al. [3] 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 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.

Duan et al. [10] 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. [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 nondeterminism, 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.

6.3 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.

6.4 Declarative Approach towards Solution

One of the evaluation measures to determine the efficiency of the composition approach is the ability to manage change with minimal programming efforts. Systems developed using conventional approaches where the requirements and the services are not externalized from the actual system itself, may often prove to be inflexible. To overcome this limitation, we adopt an declarative approach to capture the requirements of the process and the service description of partner services. Our system generates a plan based on the requirement and discovers partner services based on their descriptions. A Web process is then generated that can be deployed and executed. When there is a change in the requirement, a new process can be generated using the changed requirements. The requirements are captured as semantic template and partner services are described using SAWSDL. The non-functional properties of both the requirement and the service can be captured using WS-Policy. We define a new class of assertions called business assertions that can be added to WS-Policy to describe business level non-functional properties such as shipment destinations and shipment weight. It is our belief that the availability of visual XML editors and WSDL editors would make it easier to change these specifications. Further, this externalization eliminates re-compilation of the system for each change.

6.4.1 Semantic Templates

A semantic template captures the functional and non-functional requirements of a service requestor. It allows service requesters to semantically describe their requirements. Similar to SAWSDL the semantics are captured using the model reference attribute.The elements in a semantic template are the template term, operation, input, output, and term policy. The model reference attribute in a template term captures the domain requirement which is a concept in a classification hierarchy such as the NAICS industry classification hierarchy. In addition to the domain attribute, a template term also consists of one or more operations. The model reference attribute in the operation element carries a reference to a concept in a semantic meta-model that provides a richer description of the operation including its behavioral aspects. Each operation element has input and output elements. The model reference attribute of the input and the output elements is a concept in the semantic meta-model that describes their schema. The non-functional requirements are captured using the term policy element. Each term policy element is a collection of assertions. The term policy element can be attached to a operation, template term or to the entire semantic template. [12] and [13] discuss the Semantic Template in great detail. However for the sake of clarity and completion we describe the semantic template briefly.

Formally semantic templates are defined by:

Definition 1. *A semantic template ψ is a collection of template terms $= \{\theta | \theta$ is a template term$\}$. A template term $\theta = \{\omega, M_r^\omega, I_\omega, O_\omega, \pi_\omega, p_\omega, e_\omega\}$ is a 7-tuple with:*

- *ω: the operation*
- *M_r^ω: set of operation model references*
- *I_ω: operation inputs and their model references*
- *O_ω: operation outputs and their model references*
- *π_ω: operation level term policy and the non-functional semantics*
- *p_ω: operation precondition*
- *e_ω: operation effect*

The template term $\theta_s = \{\epsilon, \epsilon, \epsilon, \epsilon, \pi_s, \epsilon, \epsilon\}$ defining just the term policy defines semantic template wide term policies.

Figure 6.1 illustrates the conceptual model of a semantic template.

6.4.2 Business assertions in WS-Policy

The motivating scenario illustrates the importance to model the non-functional properties towards enhancing the discovery of partner services. In this section we present our approach to declaratively specify the non-functional properties of both a request as well as a service. The WS-Policy specification provides a flexible grammar for describing the non-functional properties. The WS-Policy specification defines a policy as a collection of alternatives; each policy alternative is a collection of assertions [14]. Leveraging this flexibility, we define a new class of assertions called business assertions to capture business level non-functional metrics. Examples of these metrics in the Muller service include maximum weight of shipment and shipping destinations. When used by service providers, they are attached to the SAWSDL service descriptions in the same manner as WS-Policy, using WS-PolicyAttachment.The elements of a business assertion are described in Table 6.1. These assertions are illustrated in the business policy example in Figure 6.2

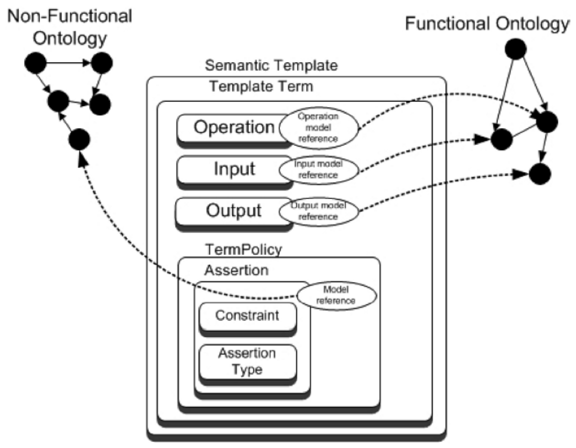


Fig. 6.1. Conceptual Model of Semantic templates

6.4.3 Formal model of abstract Web services:

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 pre-conditions 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.

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

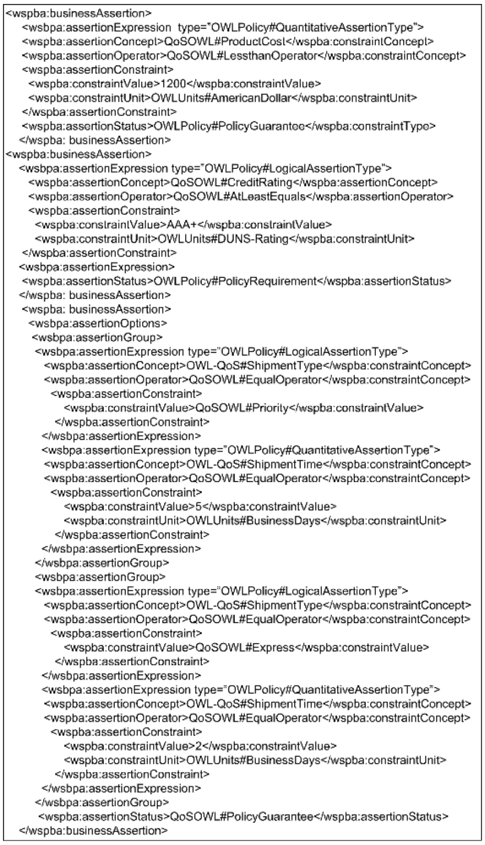
Business Assertion Element	Definition	Example
Assertion Expresssion	captures a unit assertion. Assertion expressions can either be quantitative or logical.The *ignorable* tag from the WS-Policy specification can be added to an assertion expression to indicate that the assertion encapsulated in this expression can be ignored during policy matching.	
Assertion Concept	The *ontology concept* that describes the entity of the assertion	
Assertion Operator	The fulfillment condition that this assertion will satisfy in case of a guarantee or the condition that needs to be satisfied in case of a requirement	In the cost constraint illustrated in the above example, the lessthan operator is the assertion operator.
Assertion Constraint	Captures the value of the assertion expression. Each assertion constraint has a constraint value and a unit that denotes the unit in which the constraint is expressed in.	In the cost constraint illustrated in the above example, 50 is the constraint value and AmericanDollar is the unit. The unit is usually a ontology concept. This mapping allows us to represent unit conversion rules in the ontology.
Assertion Options	When the assertion expression can have multiple values, one or more either is guaranteed or required, they are represented as options. Options contains assertion constraints	In the business assertion example, the provider agrees to a 2 day shipping if express shipping option is chosen and a 5 day delivery if priority shipping is chosen.

Table 6.1. Elements in a Business Assertion

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 since it makes it easier to express relationships such as sub-class relationships.

- **Abstract semantic Web service** [13]: Our definition of an abstract semantic Web service is built upon SAWSDL [8] 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.



```
<wspba:businessAssertion>
   <wsbpa:assertionExpression  type="OWLPolicy#QuantitativeAssertionType">
   <wspba:assertionConcept>QoSOWL#ProductCost</wspba:constraintConcept>
   <wspba:assertionOperator>QoSOWL#LessthanOperator</wspba:constraintConcept>
   <wspba:assertionConstraint>
    <wspba:constraintValue>1200</wspba:constraintValue>
    <wspba:constraintUnit>OWLUnits#AmericanDollar</wspba:constraintUnit>
   </wspba:assertionConstraint>
   <wspba:assertionStatus>OWLPolicy#PolicyGuarantee</wspba:constraintType>
  </wspba: businessAssertion>
<wspba:businessAssertion>
  <wsbpa:assertionExpression type="OWLPolicy#LogicalAssertionType">
    <wspba:assertionConcept>QoSOWL#CreditRating</wspba:assertionConcept>
    <wspba:assertionOperator>QoSOWL#AtLeastEquals</wspba:assertionOperator>
    <wspba:assertionConstraint>
     <wspba:constraintValue>AAA+</wspba:constraintValue>
     <wspba:constraintUnit>OWLUnits#DUNS-Rating</wspba:constraintUnit>
   </wspba:assertionConstraint>
  <wsbpa:assertionExpression>
   <wspba:assertionStatus>OWLPolicy#PolicyRequirement</wspba:assertionStatus>
  </wspba: businessAssertion>
   <wspba: businessAssertion>
    <wsbpa:assertionOptions>
     <wsbpa:assertionGroup>
      <wsbpa:assertionExpression type="OWLPolicy#LogicalAssertionType">
        <wspba:assertionConcept>OWL-QoS#ShipmentType</wspba:constraintConcept>
        <wspba:assertionOperator>QoSOWL#EqualOperator</wspba:constraintConcept>
          <wspba:assertionConstraint>
             <wspba:constraintValue>QoSOWL#Priority</wspba:constraintValue>
           </wspba:assertionConstraint>
       </wsbpa:assertionExpression>
       <wsbpa:assertionExpression type="OWLPolicy#QuantitativeAssertionType">
        <wspba:assertionConcept>OWL-QoS#ShipmentTime</wspba:constraintConcept>
        <wspba:assertionOperator>QoSOWL#EqualOperator</wspba:constraintConcept>
          <wspba:assertionConstraint>
             <wspba:constraintValue>5</wspba:constraintValue>
             <wspba:constraintUnit>OWLUnits#BusinessDays</wspba:constraintUnit>
           </wspba:assertionConstraint>
       </wsbpa:assertionExpression>
      </wsbpa:assertionGroup>
      <wsbpa:assertionGroup>
      <wsbpa:assertionExpression type="OWLPolicy#LogicalAssertionType">
        <wspba:assertionConcept>OWL-QoS#ShipmentType</wspba:constraintConcept>
        <wspba:assertionOperator>QoSOWL#EqualOperator</wspba:constraintConcept>
          <wspba:assertionConstraint>
             <wspba:constraintValue>QoSOWL#Express</wspba:constraintValue>
           </wspba:assertionConstraint>
       </wsbpa:assertionExpression>
       <wsbpa:assertionExpression type="OWLPolicy#QuantitativeAssertionType">
        <wspba:assertionConcept>OWL-QoS#ShipmentTime</wspba:constraintConcept>
        <wspba:assertionOperator>QoSOWL#EqualOperator</wspba:constraintConcept>
          <wspba:assertionConstraint>
             <wspba:constraintValue>2</wspba:constraintValue>
             <wspba:constraintUnit>OWLUnits#BusinessDays</wspba:constraintUnit>
           </wspba:assertionConstraint>
       </wsbpa:assertionExpression>
      </wsbpa:assertionGroup>
       <wspba:assertionStatus>QoSOWL#PolicyGuarantee</wspba:assertionStatus>
  </wspba:businessAssertion>
```

Fig. 6.2. Example Business Policy

- *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 6.2 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_1fault	sop_2fault	sop_3fault

Table 6.2. Representation of Order Management System Web service

6.5 Discovering Services

In this section we discuss the hierarchy-based matching algorithm. for discovering services The algorithm exploits the hierarchical structure of service definitions and the semantic template to compute the level of similarity between them. We define a mapping between the elements in the service structure hierarchy and the elements in the structure hierarchy of the semantic template. This mapping is illustrated in Figure 6.3. The elements in the service structure hierarchy are then compared with their mapped counterparts in the structural hierarchy of the semantic template.

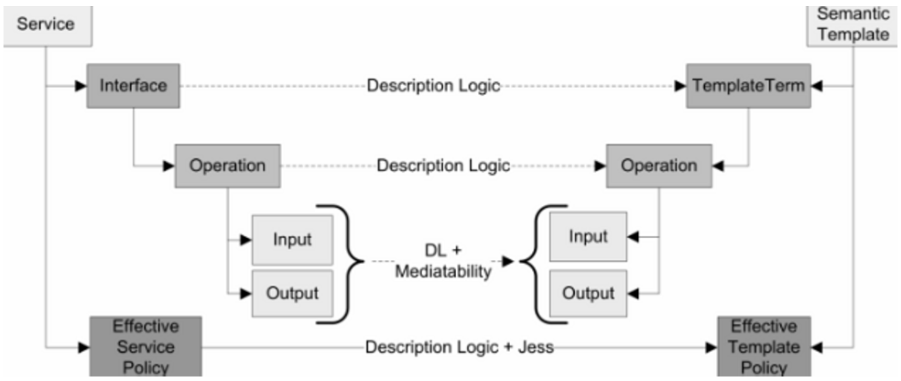


Fig. 6.3. Mapping Between Elements in Service Structure Hierarchy and Semantic Template Hierarchy

6.5.1 Overview of the Hierarchy-based Matching Algorithm

Adopting an approach that exploits the hierarchy found in service descriptions allows us to customize the comparison technique for each of the service elements. While techniques based on description logic can help in determining the semantic similarity and can be used in matching the interface and operation elements. They would not be sufficient for matching the data objects , however, because when comparing data objects one must also consider the structural similarity between them in addition to the semantic similarity.

We define a weighted scheme to compute the match score. The weights are determined by the *matchlevel* attribute of each element defined in the semantic template. This weighted scheme is used in ranking the discovered services. The rest of this section discusses the matching approach for different elements and the ranking of discovered services.

6.5.2 Description Logic Based Matching

We employ a description logic-based matching for identifying the semantic similarity between the ontological concepts captured in the modelreference attributes of the elements in the service structure and semantic template hierarchies. The similarity measure can be one of *subsumption-similar*, *equivalence*, or *generalized-similar* defined as:

- **Equivalence**: The interface element is equivalent to the templateterm element, if the ontological concepts represented by their respective modelreferences are either the same or equivalent. The equivalence measure is similarly defined for operations, input and output elements.
- **Generalized-Similar**: The interface element is *generalized-similar* to the templateterm element , if the ontology concept represented by modelreference attribute of the interface element *subsumes* the ontology concept represented by the modelreference attribute of the templateterm element. For example, a service whose domain is electronics would be *generalized-similar* to a semantic template for personal computers since electronics subsumes personal computers in the NAICS ontology illustrated in The *generalized-similar* measure is similarly defined for operations, input and output elements..
- **Subsumption-Similar**: The interface element is *subsumption-similar* to the templateterm element , if the ontology concept represented by modelreference attribute of the interface element *is subsumed by* the ontology concept represented by the modelreference attribute of the templateterm element. The *subsumption-similar* measure is similarly defined for operations, input, and output elements.

6.5.3 Non-functional Matching

In this section we describe the approach to matching non-functional requirements during service discovery. Non functional requirements consist of certain quality of service (QoS) gurantees the service provider advertices and possibly the service requestor would expect other than the functional capabilities such as security and reliability. These non-functional aspects are usually expressed using policies. To match non-functional requirements, we match the policies of the provider and the requestor. We first create the normalized effective policy for the operations in the service and in the semantic template. The procedure for creating the normalized policy is described in [14]. The effective policy of a service operation is the disjunction of the service policy, the interface policy and the operation policy. The normalized effective policy of a service operation is the normalized form of the effective policy of the operation. We define this policy as the effective provider policy. The effective policy of a semantic template operation is the disjunction of the template policy, templateterm policy and the operation policy. The normalized effective policy of a semantic template operation is the normalized form of the effective policy of the operation. We define this as the effective requestor policy. [14] describes two modes for policy matching: (1) The *Lax* mode in which assertions marked

Requestor Assertion Operator	Provider Assertion Operator	Jess Rule	Example
$<, >, \leq$	$<, >, \leq, \geq$ $=$	(Requestor Assertion Operator, Provider Assertion Value, Requestor Assertion Value)	Requestor assertion expression: Memory >512 MB; Provider assertion expression: Memory <2GB;Jess rule: (>, 2048, 512).
=	$<, \leq, >, \geq,$ $=$	(Inverse of provider operator, Provider Assertion Value, Requestor Assertion Value)	Requestor assertion expression: Memory = 512 MB; Provider assertion expression: Memory <2GB;Jess rule: (>, 2048, 512).

Table 6.3. Generating Jess Rules From Assertion Expressions

as *ignorable* can be ignored. It is not necessary for such assertions to match for the policies to match and (2) The *Strict* mode in which all the assertions must match for the two policies to match. In our policy matching approach we adopt the lax mode.

Given two policies, the first step is to identify the equivalent business assertions. This is done by comparing the assertion concept elements of the business assertions in the two policies. Two business assertions are equivalent if the ontology concepts described in their assertion concept elements are equivalent. The two policies are said to match, if all pairs of equivalent assertions that are not ignorable match. To match a pair of equivalent business assertions, we first identify the type of the assertion expression. If the assertion expression is quantitative, then they are compared using the Jess framework. Assertion expressions that are logical are compared using description logics.

In case of quantitative assertion expressions, we first ensure that both expressions are expressed in the same assertion unit. If not, we convert the provider assertion expression into the same unit as the requestor assertion expression. We assume that the rules for unit conversion are modeled in the ontology.Once the units are normalized, a Jess is rule is created from the assertion expressions. This rule is evaluated and if it evaluates to True, then we say the assertions match. The approach to creating the rule is determined by the assertion operator of the assertion expression obtained from the effective requestor policy. This is described in Table 6.3. The assertions expressions are said to match if the rule can be asserted.

Logical expressions are evaluated using description logics. If the provider assertion value subsumes or is equivalent to the requestor assertion, then we deem the expressions are a complete match. If the requestor assertion value subsumes that of the provider, then we deem it a partial match. If the subsumption or equivalence relationship cannot be determined between the provider and requestor assertion units, we check if there is a property in the schema that relates the assertion concept elements. If such a property P exists, we check if P holds between the provider and requestor assertion values. We deem a match, if P holds. The following example illustrates this better. The provider assertion expression is : Shipment destination is Europe. The client assertion expression is: Shipment destination is Germany. The provider assertion unit is a continent and the requestor assertion unit is a country. From the ontology

(ISOCountries.rdf available at the web resource[4]), we identify that *belongs_to* property exists between country and continent. We check in the ontology, if Germany *belongs_to* Europe. Since it does, we deem it a match.

6.6 Automatic Web service composition

6.6.1 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 6.4 illustrates our approach.

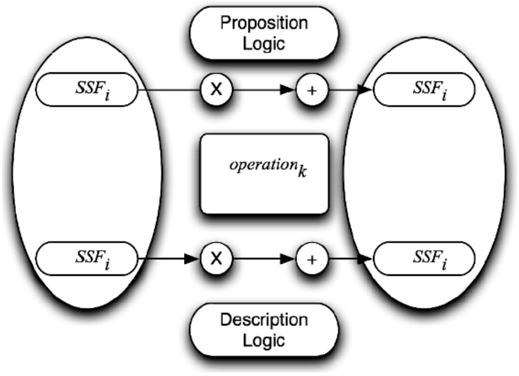


Fig. 6.4. Business Process Levels

A semantic operation ($Operation_k$ in figure 6.4) has to be checked by the *satisfy* operator (X in figure 6.4)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 6.4) operator. We will give the formal definition of *satisfy* and *apply* operators below. For convenience, we use the following notations.

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 2. *satisfy:* $(s_i, sop_k) \to \{T, F\}$
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:*

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

[4] http://knoesis1.wright.edu/swsc/

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 6.4. Representation of Order Management System Web service

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 6.2:

$$¡\ \{orderComplete = True, orderClosed = False\}, \{ontology1\#OrderID(Msg_x\}¿$$

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

Definition 3. *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 6.2 to the state

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

we will get a new state:

$$\begin{aligned} < \{orderComplete = True, orderClosed = True\}, \{& \\ ontology1\#OrderID(Msgx),& \\ ontology1\#ConfirmedOrder(sop_3OutMsg)\} >& \end{aligned}$$

6.6.2 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 ψ.

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

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 in section 6.6.6.

6.6.3 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.

6.6.4 Discovering Services

6.6.5 Extended GraphPlan Algorithm

Although most AI planning algorithms are suitable for the task here, we use GraphPlan algorithm [7]. 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.

6.6.6 Pattern-Based Approach For Loop Generation

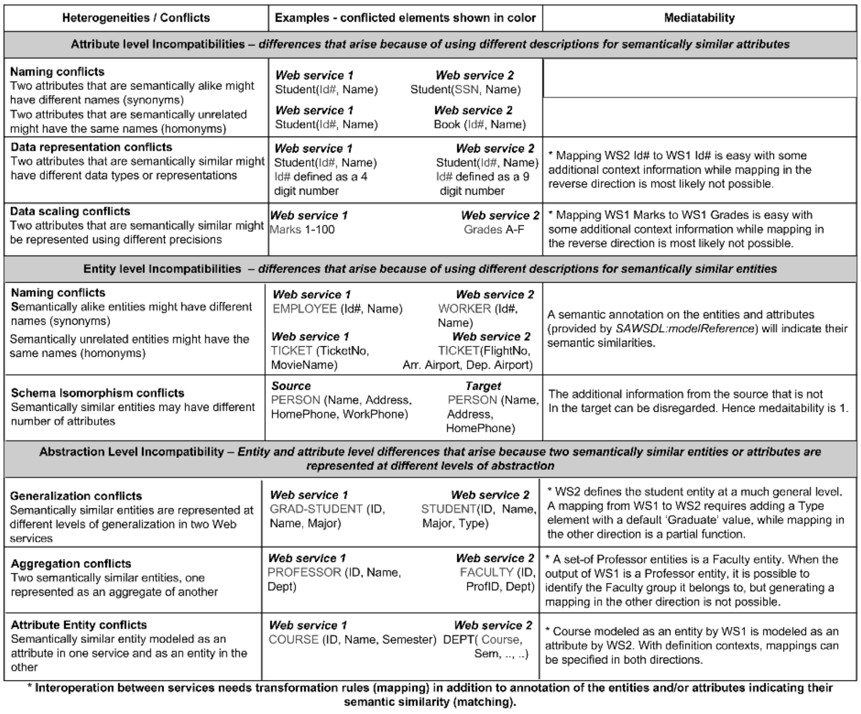
GraphPlan algorithm may generate plans only with sequence and AND-split workflow patterns [15]. However, loops are also a frequently used pattern. Loop generation (or iterative planning) itself is a difficult and open problem in AI. Much work on iterative planning is based on theorem-proving [16]. It is believed by Stephan and Biundo [17] and other researchers that iterative planning cannot be carried out in a fully automatic way. [18] 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.

Lifting and Lowering Mechanism of Data Mediation

The data mediation approach is primarily based on the lifting and lowering mechanism presented in [8]. This section looks in detail of how this lifting and lowering mapping schema functions.The base technique is to convert the message into an intermediate semantic data model and re-convert the semantic data model back into the required specific format. Converting from the message to the intermediate model is known as *lifting* and the reverse conversion is known as lowering. It is important to note that the data heterogeneities cannot be overcome merely by attaching an ontology reference. These conversions require specific references to templates or other conversion resources in order to carry out the lifting and lowering. Due to the use of XML as the primary message format, the most commonly used intermediate model is also XML and hence the conversion references are often references to XSLT documents.

To understand the importance of this approach rather than the direct use of XSLT to transform between each and every message format consider the following example. Given that there are five heterogeneous (but convertible) messages that requires conversion from message A. If direct conversion is used this requires ten conversion specifications. If the intermediate semantic model is used this conversion would require a total of tweleve conversion specifications. The advantage of the intermediate model can be seen when there is another message added along with A. This will double the number of conversion specifications if direct conversion is used. However if the intemediate model is used it results only in the addition of two new conversion specifications. It can be clearly seen that the intermediate model approach is the scalable mediation strategy.

in Figure 6.5 we describe the different heterogeneities that can exist between two XML schemas and how such heterogeneities can effect the mediations as discussed in [19]



Heterogeneities / Conflicts	Examples - conflicted elements shown in color	Mediatability
Attribute level Incompatibilities – ***differences that arise because of using different descriptions for semantically similar attributes***		
Naming conflicts Two attributes that are semantically alike might have different names (synonyms)	***Web service 1*** Student(Id#, Name) ***Web service 2*** Student(SSN, Name)	
Two attributes that are semantically unrelated might have the same names (homonyms)	***Web service 1*** Student(Id#, Name) ***Web service 2*** Book (Id#, Name)	
Data representation conflicts Two attributes that are semantically similar might have different data types or representations	***Web service 1*** Student(Id#, Name) Id# defined as a 4 digit number ***Web service 2*** Student(Id#, Name) Id# defined as a 9 digit number	* Mapping WS2 Id# to WS1 Id# is easy with some additional context information while mapping in the reverse direction is most likely not possible.
Data scaling conflicts Two attributes that are semantically similar might be represented using different precisions	***Web service 1*** Marks 1-100 ***Web service 2*** Grades A-F	* Mapping WS1 Marks to WS1 Grades is easy with some additional context information while mapping in the reverse direction is most likely not possible.
Entity level Incompatibilities – ***differences that arise because of using different descriptions for semantically similar entities***		
Naming conflicts Semantically alike entities might have different names (synonyms) Semantically unrelated entities might have the same names (homonyms)	***Web service 1*** EMPLOYEE (Id#, Name) ***Web service 2*** WORKER (Id#, Name) ***Web service 1*** TICKET (TicketNo, MovieName) ***Web service 2*** TICKET(FlightNo, Arr. Airport, Dep. Airport)	A semantic annotation on the entities and attributes (provided by *SAWSDL:modelReference*) will indicate their semantic similarities.
Schema Isomorphism conflicts Semantically similar entities may have different number of attributes	***Source*** PERSON (Name, Address, HomePhone, WorkPhone) ***Target*** PERSON (Name, Address, HomePhone)	The additional information from the source that is not In the target can be disregarded. Hence medaitability is 1.
Abstraction Level Incompatibility – ***Entity and attribute level differences that arise because two semantically similar entities or attributes are represented at different levels of abstraction***		
Generalization conflicts Semantically similar entities are represented at different levels of generalization in two Web services	***Web service 1*** GRAD-STUDENT (ID, Name, Major) ***Web service 2*** STUDENT(ID, Name, Major, Type)	* WS2 defines the student entity at a much general level. A mapping from WS1 to WS2 requires adding a Type element with a default 'Graduate' value, while mapping in the other direction is a partial function.
Aggregation conflicts Two semantically similar entities, one represented as an aggregate of another	***Web service 1*** PROFESSOR (ID, Name, Dept) ***Web service 2*** FACULTY (ID, ProfID, Dept)	* A set-of Professor entities is a Faculty entity. When the output of WS1 is a Professor entity, it is possible to identify the Faculty group it belongs to, but generating a mapping in the other direction is not possible.
Attribute Entity conflicts Semantically similar entity modeled as an attribute in one service and as an entity in the other	***Web service 1*** COURSE (ID, Name, Semester) ***Web service 2*** DEPT(Course, Sem, .., ..)	* Course modeled as an entity by WS1 is modeled as an attribute by WS2. With definition contexts, mappings can be specified in both directions.

*** Interoperation between services needs transformation rules (mapping) in addition to annotation of the entities and/or attributes indicating their semantic similarity (matching).**

Fig. 6.5. Different Heterogeneities

6.7 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

1. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: Htn planning for web service composition using shop2. Journal of Web Semantics **1**(4) (2004) 377–396
2. Narayanan, S., Mcilraith, S.A.: Simulation, verification and automated composition of web services. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM Press (2002) 77–88
3. Rao, J., Dimitrov, D., Hofmann, P., Sadeh, N.: A mixed initiative approach to semantic web service discovery and composition: Sap's guided procedures framework. In: ICWS. (2006) 401–410
4. Ponnekanti, S.R., Fox, A.: Sword: A developer toolkit for web service composition. (2001)
5. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes (2004)
6. Rao, J., Küngas, P., Matskin, M.: Logic-based web services composition: From service description to process model. In: ICWS. (2004) 446–453
7. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education (2003)
8. for WSDL, S.A., working group, X.S.: Semantic annotations for wsdl and xml schema. Technical report, W3 Consortium (2007)
9. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A.P., Verma, K.: Web service semantics – wsdl-s. Technical report, LSDIS Lab and IBM Corporation (2004)
10. Duan, Z., Bernstein, A.J., Lewis, P.M., Lu, S.: A model for abstract process specification, verification and composition. In: ICSOC. (2004) 232–241
11. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of composite bpel4ws web services. In: ICWS. (2005) 293–301
12. Gomadam, K., Ranabahu, A., Ramaswamy, L., Sheth, A.P., Verma, K.: A Semantic Framework for Identifying Events in a Service Oriented Architecture. In: ICWS. (2007) 545–552
13. Verma, K.: Configuration And Adaptation of Semantic Web Processes. PhD thesis, University of Georgia (2006)
14. working group, W.S.P.: Web services policy 1.2 - framework (ws-policy)
15. van der Aalst, W., Hofstede, A.: Yawl: Yet another workflow language (2002)
16. Biundo, S.: Present-day deductive planning. In Bäckström, C., Sandewell, E., eds.: Current Trends in AI Planning: Proceedings of the 2nd European Workshop on Planning (EWSP-93), Vadstena, Sweeden, IOS Press (Amsterdam) (1994) 1–5
17. Stephan, W., Biundo, S.: Deduction-based refinement planning. Technical Report RR-95-13 (1995)
18. Levesque, H.J.: Planning with loops. In: IJCAI. (2005) 509–515
19. Nagarajan, M., Verma, K., Sheth, A.P., Miller, J.A., Lathem, J.: Semantic interoperability of web services - challenges and experiences. In: ICWS. (2006) 373–382