, , 1–28 ()

Using Semantics for Policy-based Web Service Composition *

SOON AE CHUN[1,2], VIJAYALAKSHMI ATLURI[2] AND NABIL R. ADAM[2]

{soon,atluri,adam}@cimic.rutgers.edu

[1] *CDS Department, Seton Hall University, South Orange, NJ 07709*
[2] *Center for Information Management, Integration and Connectivity (CIMIC)*
and
MSIS Department, Rutgers University, 180 University Ave., Newark, NJ 07102

Editor:

Abstract. Proliferation of Web technologies and the ubiquitous Internet has resulted in a tremendous increase in the need to deliver one-stop Web services, which are often composed of multiple component services that cross organizational boundaries. It is essential that these Web services, referred to in this paper as *service flows*, be carefully composed in a *dynamic* and *customized* manner to suit to the changing needs of the customers. This composition should be conducted in such a manner that (i) the composed service flow adheres to the policies imposed by the organizations offering the component services, (ii) the selected component services are *compatible* with one another so that the entire composition would result in a successful service flow, and (iii) the selected component services *most closely* meet the customer requirements. In this paper, we propose a policy-based Web service composition that utilizes the semantics associated with the component services.

We consider policies imposed by different entities while composing service flows, which include *service policies* (imposed by the organizations offering component services), *service flow policies* (associated with the entire service flow), and *user policies* (the user requirements expressed as policies). In addition to these *policies*, one may consider rules at the *syntactic* and *semantic* levels that can be used to select relevant component services in order to compose customized service flows, by considering the notions of *syntactic, semantic* and *policy compatibility*. We model the different policies and the service topic ontology using OWL, DAML-S, RuleML and RDF standards.

Keywords: Workflow, Policies, Composition, Web Services

1. Introduction

With the need to offer business services through the Internet as a one-stop shopping, traditional businesses and commerce activities are undergoing unprecedented transformation. Traditionally, the composite services are offered through forming alliances with other companies, as in the supply chain collaborations and the B2B interactions. These alliances with partners and suppliers involve inter-organizational business processes that are established based on cooperative agreements between firms, and involve exchange and sharing of business information and transaction data, or co-development of products.

* The work is partially supported by the National Science Foundation under Digital Government program grant EIA-9983468 and the Meadowlands Environmental Research Institute.

Often, the collaborations start with manually searching for partners and establishing agreements on business data formats and communication network, as well as transaction exchange steps. For many large organizations, there is a set of pre-established partner organizations for regular business operations. For automation of data exchange and communication, companies may utilize EDI or XML documents over a private network (VAN) or the Internet. For automating the coordination among systems of partner organizations such as when and how to exchange data, there has been industry-wide standards specification such as RosettaNet's Partner Interface Protocol [3, 18].

In addition to these static, agreed upon cooperation, recently businesses have started to use the WWW as a vast repository of resources for communication such that businesses are now able to link people and resources across organizational boundaries dynamically, thereby creating temporary, ad-hoc enterprises, known as *virtual enterprises.* This is propelled primarily due to the demands from customers, including customized one-stop shopping of goods and services. The services of different organizations need to be combined together on-demand to form seamless business processes as if the individual service providers are part of a whole enterprise.

The grand vision of the Semantic Web [8] is to achieve the formation of these loosely coupled, dynamic business processes by automated agents interacting with individual services provided by different organizations, rather than by users manually interacting with (or pulling) the static Web resources (documents and services). Web services from multiple autonomous organizations in different locations will be identified, selected and composed together to perform tasks, provide information, transact business, and take action on behalf of users. These inter-organizational Web services, called *service flows*, are created on demand, by discovering the "right" component Web services that meet the inter-organizational business goals and tailored to satisfy the customers' needs. We refer to this process as *Web service composition problem*, which requires both inter- and intra-organizational business processes be composed in a coordinated manner. Our approach to the composition problem considers the business policies, i.e. terms and conditions that an organization imposes to govern the way business services operate, and the user's personal policies, i.e. terms and conditions that an individual user faces in consuming services.

A service flow is thus comprised of component Web services, where each participating organization has a set of services to offer. To easily identify the relevant services, we assume that these are organized as a service topic ontology. However, service composition requires to first select the services that are compatible. We utilize policies imposed by different entities in the composition process, including *service policies* (imposed by the organizations offering component services), *service flow policies* (associated with the entire service flow), and *user policies* (the user requirements expressed as policies). In addition to these *policies*, one may consider rules at the *syntactic* and *semantic* levels that can be used to select relevant component services in order to compose customized service flows, by considering the notions of *syntactic, semantic* and *policy compatibility.*

Under syntactic compatibility, the service discovery and selection will verify whether a service is compatible with respect to its specifications, such as matching of the output of one service with the input of the subsequent service. On the other hand, semantic compatibility considers domain- specific guidelines, and checks whether the service composition follows the "correct" combination specified in a standard operating procedure, such as manufacturing a specific drug that requires right kind and amount of ingredients and their combinations. The policy compatibility requires that the different policies that each organization imposes on the services offered by them should be compatible with those by other organizations. For instance, the delivery day for one service is on every Thursday, thus, the composition needs to select other services that conform to this company policy. The policy level compatibility considers both organization's internal policies as well as regulatory rules that are imposed on the industry level business practices. Figure 1 illustrates these different levels of compatibility rules in a service composition. Prior work [10, 12] on Web service composition considers syntactic and semantic compatibilities, whereas, in this paper, we consider all the three types of compatibilities.

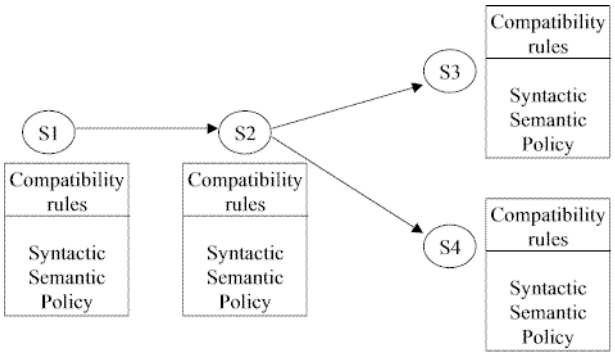

Figure 1. Service composition based on compatibility rules

The standard XML markup languages such as WSDL, UDDI, RDF, DAML-S, and OWL, facilitate the Web service discovery and composition. They allow the discovery and composition based on syntax and semantics of described Web services. We argue in this paper that this is inadequate to achieve a fully automated Web Service composition. More pragmatic rules need to be considered for a software to determine whether a specific Web service can be used in a composite business service flow. These pragmatic rules include the geographic location covered by a Web service, the range of time period during which a service is relevant, quality of service, delivery policy, product cost structure, trust of hosts (e- Vendors), and relationships among hosts (competitors), regulatory constraints, payment methods, etc. Many pragmatic rules are specified as company policies, imposing constraints and conditions in rendering services. Many of the policies also specify rules on how the actual business contracts and negotiations need to be handled within the organization, with other partner organizations and with customers. In addition, the customers also have some constraints or conditions to meet, such as preference in the

payment methods or delivery dates. These individual user's conditions on selecting a particular service over others are also the user's pragmatic rules representing personal policies. We focus on organizational policies as well as user specified policies, and their role in composition. We present the specification of these policies as well as user policies that specify complex service requests. These policies affect not only the proper selection of services and service providers, but also the composition of the entire service flow itself. In this paper, we present a service flow composition approach that considers the user policies as well as the organizational policies that takes into account the compatibility at syntactic, semantic and policy level.

This paper is organized as follows. In Section 2, we present the different categories of Web service compatibility rules. Section 3 presents our policy-based composition model, called the service flow model. In Section 4, we present the service compatibility model and the compositional rules adopted by our policy based composition approach. In section 5, we present the policy-based composition approach. In section 6, we explore the policy negotiation by relaxing the rules when true compatibility does not exist. Section 7 presents the details of system implementation. Section 8 reviews related research. Section 9 provides conclusions and an insight into future research.

2. Compatibility Rules

The inter- and intra-organizational processes are subject to various business rules that govern the way a business operates. Business rules include various policies and procedures internal to a particular unit or organization as well as various business protocols between units within an organization and among different organizations. Examples include: "Pay a supplier invoice only if it has been approved." "Only good customers may obtain credit orders." "Overdue invoices occur 30 days after statement." "Many payments can be made per invoice." "Only one invoice should be generated for one order." "Credit balance should be greater than or equal to order value to accept order, otherwise reject the order." In addition, certain business processes also need to follow the inherent semantic rules to fulfill a service. For example, a semi-conductor chip making process or an auto manufacturing process needs to follow certain technical specifications. They also are subject to external regulations and codes of business conduct imposed by regulatory and auditing authorities.

The different types of rules, therefore, govern the composition of services by restricting certain combinations of services or by imposing constraints on the selection of a service over another service. We have defined three levels of rules - syntactic, semantic and policy, which serve as basis for determining three levels of compatibility - syntactic level, semantic level and policy level.

Syntactic(operational) Rules: The composability of services depends on the preconditions, input, and output requirements. For instance, the output of a service s_1 needs to be compatible with the input of another service s_2. The syntactic rules ensure that the composition of the services yields a composite service that is operationally or syntactically possible.

Semantic Rules: The service composition should make sense according to the standard business practices. The semantic composition rules often require domain expert knowledge, such as trade laws, State legislations, federal environmental regulations, formal company policies, experimental procedures, etc. For example, according to the New Jersey law, any land development close to the coastal area forces the agent to select the coastal permit application service. The selected service is required to be part of the service flow. These rules enforce the expertise knowledge in a domain. In a commerce domain, the product type, e.g. Apple Computer software, determines (or limits) a set of license or update services that are compatible with the product type. When purchasing computer components to build a computer, a SCSI device purchase will limit a set of controllers that should be compatible to the device. The product type of one service should be semantically compatible with the other purchase service. Thus, the application of semantic rules plays a significant role in service composability to ensure the composed services are compatible at the semantic level.

Policies: In real life it is often the case that several services possess the same profile and provide the same functionalities. The automated selection of one service among these functionally equivalent services may require policy level compatibility. For example, when buying a book and flowers as a combined gift, the delivery of the book and flowers need to be coordinated. Therefore, when composing this service, the combination of service providers who can deliver the book and flowers at the same day is a preferable choice. In other words, the delivery policies of the bookstore and that of the flower shop in terms of the day of the delivery for the same destination should be compatible. Therefore, even though there may exist a number of service providers that deliver books and flowers, only few are policy compatible. As a result, the selection of service providers is further restricted by the policy compatibility requirements.

In summary, syntactic and semantic rules, and business policies allow an agent to restrict the selection and composability of services. These rules allow the service composition to be customized for the user's service requirements as well as the organizational policy level composability.

3. Service Flow Model

As described in section 1, the service capabilities offered by a Web service are often accompanied by certain constraints. In this section, we define the *service flow model*.

3.1. Preliminaries

Definition 1. **[Literals]** A literal $l \in L = \{N \cup AN \cup AR\}$, where N is the set of natural numbers, AN the set of alpha-numeric strings, and AR the arrays of natural numbers or strings.

Definition 2. **[Temporal Constants and Variables]** Let TC be the set of constant time points (e.g., Apr9:2002:17:12:39), and $TV = \{t_b, t_e\}$ be the set of temporal variables, where $[t_b, t_e]$ indicates a time interval.

Definition 3. **[Geospatial Constants and Variables]** Let geospatial constants GC be the set of rectangles, where each rectangle is a 4-tuple $\langle t, g, h, w\rangle$ representing the minimum bounding box covering a geospatial area, such that t represents the latitude, g the longitude for the center of the bounding box, w the width from the center, and h the height from the center. Let $GV = \{address, region, area, place\}$ be a set of geospatial variables.

3.1.1. Web Service Ontology In real world, the services do not exist in isolation. They are created to interact with other services in a meaningful way to achieve certain goals. Although there are some services that overlap across different industries, typically, the services that interact with each other can be grouped together within an industry and differ from those in other industries, e.g. services in the travel industry, services in car manufacturing industry, services in the health care industry, the government services, etc. These services and the inter-relationships among them can be captured using a service part-whole relationship hierarchy, called *service ontology*. An ontology is a formal description of a set of concepts and their relationships to each other. Essentially, service types can be specified using a service ontology. Each service type can have several service subtypes as its components. Each service type is described with a set of descriptors. The whole service type then can be described with a set of descriptors from all its part services. An individual service (a service instance) provided by a particular organization or a business unit belongs to a service type. Each individual service instance can be described using the descriptors associated with the service type it belongs to. The service ontology is formally defined below.

Definition 4. **Web Service Ontology:** A *Web service Ontology*, SO, comprises of a set of Web service types $S = \{s_1, s_2, \ldots\}$ that are related by the *part-of* relationships. Each service type $s_i = \langle tid_i, DC_i\rangle$, such that tid_i is a unique identifier for s_i, and DC_i is the set of service descriptors associated with s_i. Each descriptor $dc_{ij} \in DC_i$ is a 4-tuple $dc_{ij} = \langle name, type, value, mode\rangle$ where $name$ is the descriptor name, $type \in L$ is the valid datatype for the descriptor value, $value \in \{N \cup L \cup TC \cup GC\} \cup AR \cup AN$, and $mode \in \{opt, obl\}$ denotes whether the descriptor is optional or obligatory for this service type.

We use $(s_i \subset_S s_j)$ to represent that s_i is a part-of s_j, and $DC(s_i)$ to denote the set of descriptors associated with a service type s_i. For example, a service type *purchase hard-disk* is part-of *build a computer* service type. Similarly, a hotel booking service is a part-of a travel planning industry, while a business registration service is a part-of government services and a part-of a new business service.

Figure 2 shows a service ontology SO representing the business services S offered by the government. New business registration services consist of local and state gov-

ernment services. The state government services for a new business service consist of a service to obtain a certificate of incorporation, a service for tax registration, etc. Each service type in the ontology has a set of descriptors in addition to the interface parameters such as Input and Output. For example, all services that are part of a new business service type share descriptors such as business type, business owner, business location, and possess their unique descriptors such as employee numbers, etc. The subtype services inherit descriptors from their parent service types.

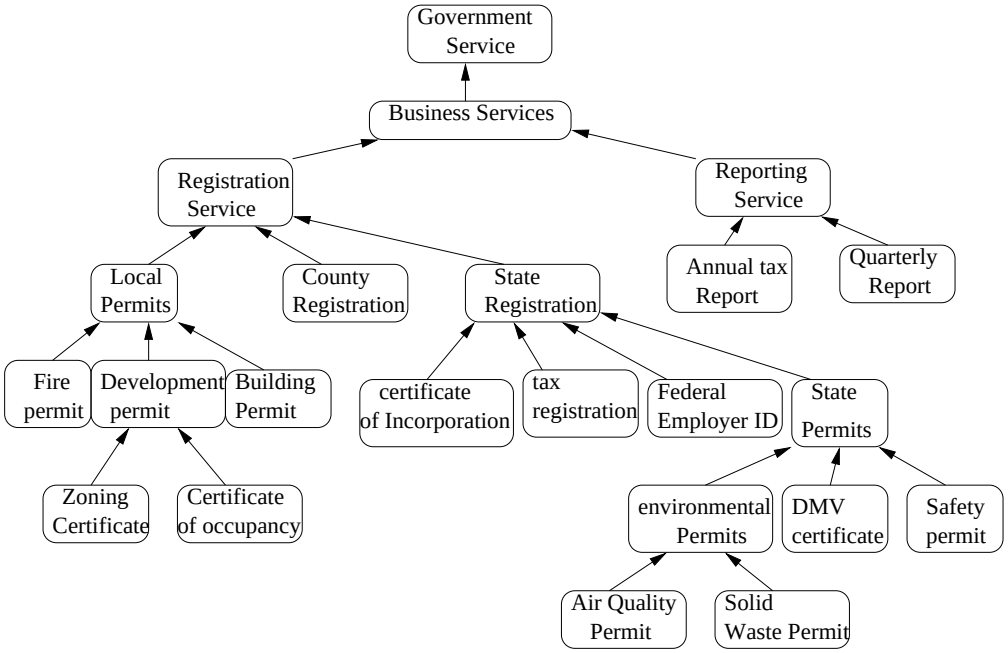

Figure 2. Government Service Ontology

3.1.2. Policies In this section, we develop the formalism necessary to describe the policies.

Definition 5. **[Policy Variables]** Given a Web service ontology, SO, we define the set of *policy variables*, $PV = CV \cup SV \cup TV \cup GV \cup A$, where $CV = \cup\{DC(s_i)|s_i \in SO\}$, SV the set of variables ranging over Web service types $SO = \{s_1, s_2, \ldots\}$, and TV, GV and A represent temporal, geospatial and name variables, respectively.

Definition 6. **[Policy Expression]** Let $PV = CV \cup SV \cup TV \cup GV \cup A$ be the set of policy variables, and $LOP = \{=, \neq, <, >, \leq, \geq\}$ be the set of logical operations. A Web service policy expression pe is defined as follows:

1. If $s_i \in S$, then $s_i(x)$ is a pe, denoting the set of Web services belonging to service type s_i;
2. if $v \in CV$ is a policy variable and $l \in L$ is a literal and $op \in LOP$, then $v\ op\ l$ is a pe;
3. if $v \in \{TV \cup GV \cup A\}$ is a policy variable and $c \in \{TC \cup GC \cup N\}$ is a temporal constant, and $op \in LOP$, then $v\ op\ c$ is a pe;
4. if pe_1 and pe_2 are two Web service policy expressions, then $pe_1 \wedge pe_2$, $pe_1 \vee pe_2$, $\neg pe_1$, and (pe_1) are pe.

EXAMPLE: Examples of policy expressions are as follows.
pe_1 = New-Business-Registration(x) denotes a policy (constraint) that the service offered is of the semantic type New-Business-Registration.
pe_2 = PurchaseOrder(x) is a policy expression and denotes a policy that a Web service offered is a PurchaseOrder type.
pe_3 = (delivery-mode ='FedEx') $\wedge$ (delivery-area = 'NJ') denotes a policy that the delivery is by Federal Express and the area covered by the delivery is New Jersey. □

The policies associated with a Web service may be regarding the service implementation capabilities (e.g. input parameters {price, number of days}), or may be imposed by the service organizations (e.g., service delivery is restricted to weekdays). In addition, each policy can be either negotiable or non-negotiable, which means that the policy constraint can be relaxed or not. Following defines the policy statements for each Web service.

Definition 7. **[Policy Statement]** A policy p is defined as a pair (pe, m), where pe is a policy expression to denote the conditions to be satisfied, and $m \in \{n, nn\}$ is the flexibility mode of the policy, where n denotes negotiable and nn non-negotiable.

Given a policy $p = (pe, m)$, we use $PV(p)$ to denote the set of policy variables involved in pe. Note that, in our model, we limit the type of variables and constants used in the policy expression to a specific well-defined set so that our model lends itself to verifying the compatibility among the services.

Definition 8. **[Policy Range]** Given a policy p, we define the range of $pv \in PV(p)$ as follows: (i) if $pv \in AR$, $[min(value(pv)), max(value(pv))]$, (ii) if $pv \in TV$, $[min(t_b(pv)), max(t_e(pv))]$, and (iii) if $pv \in GV$, [$latitude(pv)$,$longitude(pv)$, $height(pv)$,$width(pv)$].

Note that, since in our model, we support only three types of variables that assume spatial, temporal and numerical values, we define the range of the policy on these three types of variables.

In this paper, we encounter three types of policies: (i) service policies, (ii) service flow policies, and (iii) user policies. Service policies are defined by the individual

organizations offering services, service flow policies are defined by the organizations offering a composite Web service comprising of component services, and user policies are defined by the consumers of the services.

3.2. Component Web Service

Each Web service s_i is comprised of a set of operations it performs, and is associated with a set of interface parameters. It is also associated with a set of policies on how and when the service is applicable, or what conditions may need to be satisfied for it to be invoked.

A service is an instance of a service type in the Web service ontology. This can be specified as an application capable of being defined and located via the Internet protocol, capable of interacting with other software applications, and can be identified by a Uniform Resource Identifier (URI). Each service is stated with a set of constraints using descriptors of the service type, called the set of *service policies*, which are implemented by the service provider (organization) in order to offer the service. A service policy, *sp* is stated in terms of the conditions and contexts under which a particular policy becomes relevant. The consumers of the service (including human consumers, as well as automated agents on behalf of a human consumer) are subjected to these terms and conditions in order to invoke or consume a service.

Definition 9. **[Web Service]** A Web service s is a 4-tuple $\langle\ \Omega, Input, Output, SP\ \rangle$, where Ω is the set of operations in s, $Input$ is the set of objects allowed as inputs to s, $Output$ is the set of objects expected as output from s, and SP is the set of policies that must be adhered to utilize the s.

Each service policy $sp \in SP$ is expressed as a policy statement, as in definition 7. We use $SP(s)$ to denote the set of policies associated with a service.

Definition 10. **[Service Policy]** Given a Web service s, let $SP(s)=\{sp_1, sp_2, \ldots\}$ be the set of policy statements that s is subject to. Each policy statement $sp_i = (se_i, m_i)$ is such that se_i is a policy expression defined over policy variables $PV = CV \cup SV \cup TV$, where $CV = DC(s)$, and $SV = \{s\}$, and m_i is a mode for negotiability.

EXAMPLE: Following are some examples of service policies.
1. $SP(s_1)$ ={(new-business-registration(x), nn) (location=Newark, n)(business-type=autobody, nn)} states that the service s_1 is used for new-business-registration type whose location is in Newark and whose type is an autobody shop.
2. $SP(s_2)$ = {(ReserveHotel(x), nn) (Provider= Sheraton, nn) (Name= Reserve-HotelRoom, nn) (contact= John@sheraton.com, nn) (Purpose= reserves a room in Sheraton Hotel, nn) (uri= http://www.sheraton.com/ReserveHotelRoom.wsdl, nn) (Input= {arrival-date, days-to-stay}, nn) (Output= {reservationNumber, rate}, nn)}. The policy s_2 states that this service is for hotel reservation provided by Sheraton Hotel, and the customer must present the arrival date and duration of stay in order to get reservation and room rate information.

3. $SP(s_3)$ ={(OrderProcess(x), n) (Provider=IBM, nn) (Name= POrderWS, nn) (Contact= smith@ibm.com, nn) (Purpose= process purchase orders, nn) (uri= http://www.ibm.com/POrderWS.wsdl, nn) ((geographic area= NE) $\vee$ (geographic area=South), n)}. The policy statement for s_3 mandates that the order processing service by IBM can be used only for the orders made in North East or South regions. □

Given component services provided by organizations with specific policies, the service flow needs to select suitable component services that meet the user policies and the organizational policies. The selected service that complies with the policies is called a service instance (or instantiated service) as defined in the following.

Definition 11. [**Service Instance**] A service instance si is an instantiation of a service type s such that for each service policy $sp \in SP(s)$, its policy expression is satisfied.

3.3. Service Flow

A *service flow* is comprised of multiple component Web services and a set of policies that must be satisfied to compose the service flow.

Definition 12. [**Service Flow**] A service flow SF is defined as a pair (S, P) where $S = \{s_1, s_2 \ldots s_n\}$ where each $s_i \in SO$ is a service type, and $P = \{p_1, p_2, \ldots\}$ where each p_i is a policy statement to denote a condition to be satisfied to instantiate $s_i \in S$.

Definition 13. [**Service Flow Policy**] Given a service flow $SF = (S, P)$ where $S = \{s_1, s_2 ... s_n\}$ and each policy statement $p_i = (pe_i, m_i)$ is such that that the policy expression pe_i is defined over policy variables $PV = CV \cup SV \cup TV$, where $CV = \cup\{DC(s_i) | s_i \in S\}$ and SV the set of variables ranging over service types $S = \{s_1, s_2, \ldots\}$.

We use $P(SF)$ to denote the set of policies associated with a service flow SF. Note that the service flow policies, defined above, are different from service policies. Service flow policies apply to the entire service flow comprising of multiple services, whereas the service policy is applicable to a specific component service within the service flow.

EXAMPLE: Following are some examples of service flow policies.
P(TravelPlanning) = {(budget $\leq$ \$500,nn)(departure-date='12/20/2004',n) (arrival-date='1/2/05', nn)}. The service flow of travel planning needs to have a cost within \$500 and the travel should be arranged between December 20, 2004 and January 2, 2005, although the departure date may be flexible.
P(computer-assembly-info) ={(total-cost $<$ \$700, nn)(total-weight $<$ 3lbs, nn) (deadline $<$ 2hr, n)}. The computer assembly information service flow should consists of the component services so that the total cost be less than \$700 and the weight should be less than 3 pounds and the information of the assembly should be available within 2 hours. □

Definition 14. **[Compatible Descriptor Set]** Given a service flow $SF = (\{s_1, s_2 \dots s_n\}, P)$, a *Compatible Descriptor Set*, $CDS = \{dc_1, dc_2 \dots dc_m\}$, such that each $dc_i \in \cup\{DC(s_i)|s_i \in SO\}$ and there exist compatibility requirements among them.

We use $CDS(SF)$ to denote the set of compatible descriptors for a service flow SF.

EXAMPLE: The Compatible Descriptor Set (CDS) for a service flow of computer assembly include: a processor chip should be compatible with video controller, network driver. Thus, $CDS(SF)$ = {Processor chip, Operating System, video controller, network driver, hard disk, disk driver}. □

Definition 15. **[Compatible Descriptor Value Set]** Given a Compatible Descriptor Set, CDS, a *Compatible Descriptor Value Set*, $CDVS = \{value(dc_1), value(dc_2) \dots value(dc_m)\}$, such that each $dc_i \in CDS$.

EXAMPLE: Compatible descriptor value set (CDVS)={Intel Pentium 4, Windows OS, Adaptec-7880, Ultra-Wide, SCSI controller}. □

Definition 16. **[Service Flow Instance]** A service flow instance SFI is defined as a set of service instances $\{si_1, si_2 ... si_n\}$ where each si_i is an instance of a service type $s_i \in SO$, and for each service flow policy $p \in P(SF)$, its policy expression is satisfied.

3.4. User Policy

The semantic Web envisions that the user goals are specified in simple terms, and the software agent figures out sensible means of achieving the user goals. However, in reality, the user has various constraints and preferences in selecting particular services. These preferences, constraints and goals are considered as user specified policies, so-called user's service policies that should be considered in selecting, composing and executing Web services. Examples include: "if s_1 and s_2 provide the same products, but s_1 offers a discount, then select s_1," or "if the service provider of s_1 can deliver its product within 2 days, then also add s_1 to buy a product d from the same vendor to be delivered together." User policies in turn are of two types – local user policies and global user policies, where the former is defined over component services and the latter is defined over the service flow.

Definition 17. **[Local User Policy]** Given a user u_i and a service type s_j, we define a set of local user policies $LUP_{ij} = \{lup_{ij_1}, lup_{ij_2}, \dots\}$, where each local user policy statement $lup_{ij_k} = (lue_{ij_k}, m_{ij_k})$, such that lue_{ij_k} is a policy expression defined over policy variables PV.

Definition 18. **[Global User Policy]** Given a user u_i and a service flow $SF = (S, P)$, where $S = \{s_1, \dots s_n\}$, we define a set of global user policies $GUP_i = \{gup_{i_1},$

$gup_{i_2}, \ldots\}$, where each global user policy statement $gup_{i_j} = (gue_{i_j}, m_{i_j})$, such that gue_{i_j} is a policy expression defined over policy variables PV

In the above definitions the constraint attributes in the policy expression are derived from service descriptors for each service type s in SO. For instance, a user $John$ can specify his service requirements as the following user policy statements.

- $gup_{John,registeranewbusiness}$ ={(duration<10 days, n), (location=NJ, nn)}
  $lup_{John,CertificateofIncorporation}$ ={(business structure=incorporated, nn) (business name =Car Care, n) (business type= Autobody shop, nn) (location= "80 Mercer Street, Newark, NJ 07052", n)}
  $lup_{John,AirQualityPermit}$ ={(business type= Autobody shop, nn) }
  $lup_{John,EmployerID}$ ={(employee_number$\geq$ 3, n)}

  John's global policy for registration of a new business states that the business is opened in NJ, and the registration process should not exceed 10 days. His policies on component services include business structure should be incorporation, business type is an automobile repair shop with more than three employees.

- $gup_{Mary,travel-planning}$ ={ (price $\leq$ \$400, nn) (begin-date='July 23, 2004", n), (end-date="August 7, 2004", nn)) }
  $lup_{Mary,hotel-booking}$= { (hotel-price < \$90, n)}
  $lup_{Mary,car-rental}$= {(car size= small, nn)}
  $lup_{Mary,airline-booking}$= {(airline type= 'Delta', n)}

  Mary's policy states that she wants to achieve travel-planning service that has a hotel price of less than \$90 per night, a rental car be a small size car, and airline tickets from 'Delta' to utilize her mileage benefits.

4. The Service Compatibility Model

In ad-hoc, dynamic Web services, the component services not only adhere to the service specific and service organizational policies, but also the inter-service constraints. In the static environment, companies set up contracts to specify the terms and conditions through the human (or semi-automatic) interactions. In ad-hoc dynamic composition of services, the contracts pertaining to specific services are not specified in advance. On the other hand, there are rules that govern valid interactions of services crossing organizational boundaries, whose validity is checked for contractual relationships. These protocols or rules on how to compose valid services are specified as a set of general compatibility rules. The compatibility rules refer to syntactic, semantic and policy aspects of the service contracts, further categorizing the compatibility into syntactic compatibility, semantic compatibility as well as usage compatibility. The semantic and policy compatibility rules may be governed by regulatory rules.

Definition 19. **[Syntactic Compatibility]** Given two Web services s_i and s_j, we say s_i is *syntactically compatible* with s_j, denoted $s_i \propto_{syc} s_j$, if Output(s_i) $\subseteq$ Input(s_j).

Note that the above compatibility relationship is not commutative. In other words, $s_i \propto_{syc} s_j$ does not necessarily imply $s_j \propto_{syc} s_i$.

If Input(tax-register) ={FEIN, Business Address} and Output(Federal-Employer-IdNumber) = {FEIN}, then the tax registration service is syntactically compatible with the service for generating a federal employer identification number, since the input and output relationships between these two services satisfy the syntactic compatibility rule.

Definition 20. **[Semantic Compatibility]** Given two Web services s_i and s_j, we say s_i is *semantically compatible* with s_j, denoted $s_i \propto_{smc} s_j$, and s_j is *semantically compatible* with s_i, denoted $s_j \propto_{smc} s_i$, if there exist two descriptors dc_{ik} of s_i and dc_{jl} of s_j such that $\{dc_{ik}, dc_{jl}\} \in CDS_m$, where CDS_m is a compatible descriptor set, and $\{value(dc_{ik}), value(dc_{jl})\} \in CDVS_m$.

If a service, s_1= purchasing a CPU, resulted in a value "Intel Chip" for CPU, and a service s_2 = selecting an operating system, resulted in a value "WindowsNT" for OS, then s_1 and s_2 are semantically compatible. On the other hand, if s_2 resulted in "OS X", s_1 and s_2 are not semantically compatible.

Definition 21. **[Policy Compatibility]** Given two Web services s_i and s_j, we say s_i is *policy compatible* with s_j, denoted $s_i \propto_{poc} s_j$, and s_j is *policy compatible* with s_i, denoted $s_j \propto_{poc} s_i$, if there exist two policies $p_{ik} \in P_i$ and $p_{jl} \in P_j$ such that $(pv_m \in PV((p_{ik}) = pv_n \in PV((p_{jl})) \wedge range(pv_m) \cap range(pv_n) \neq \emptyset$.

EXAMPLE: Assume you are shopping for a gift which involves scheduling delivery services. Your preference is to have both a book and flowers to arrive at the same time.
(1) If a book delivery service s_1 covers an area of NY and NJ, and a Web-ordered flower service s_2 covers an area of NY, then these services are policy compatible, since their spatial ranges have an overlap area.
(2) If a book delivery service s_1 has a policy that states all deliveries are done on Monday through Wednesday every week and if a flower delivery service, s_2 has a policy to deliver their flowers on Tuesday and Thursdays every week, then these two services are considered policy compatible, because there exist a temporal overlap. □

In summary, the syntactic compatibility checks that the input, output, and preconditions between individual services from different sets are compatible. The semantic compatibility check ensures that the domain specific semantic relationships among services are compatible or the user requirements are met. The policy level compatibility ensures the rules imposed by one organization allow its service to be composable with another service organization's policy. These compatibility checks examine the relationships of policies and rules to filter any incompatible services in the composite service.

In order to make sure these compatibilities are verified, a set of compositional verification rules are introduced in our model, which are specified as follows.

Definition 22. **[Compositional Rule]** A compositional rule cp is a tuple (ce, a, dg) where ce is a conditional expression, a is an action, and dg specifies the degree of the strength for the rule $dg \in$ {must, possible}, where must denotes the compositional rule has no exceptions, and possible denotes that the compositional rule is recommended.

Following are types of compositional rules supported under our model.

1. **General Composability:** cp_1=($s_i \propto_{syc} s_j \wedge s_i \propto_{smc} s_j \wedge s_i \propto_{poc}, s_j$, $SF = SF \cup \{s_i, s_j\}$, must).

   This compositional rule cp_1 states that a service s_i is composable with s_j, denoted by $s_i \bowtie s_j$, iff $s_i \propto s_j$, i.e. they are syntactic, semantic and policy compatible.

2. **Sub-type service inclusion:** cp_2 = ($(s_i \subset_S s_j)$ mode(s_i)=obligatory, $SF = SF \cup \{s_i\}$, must).

   The rule cp_2 states that if the user's goal (service flow) is a service s_j, then its obligatory subtype service s_i should be also part of the service flow SF.

3. **Satisfaction of User Policies:** cp_3 = (eval($P(s_i), uc$)=T, $SF = SF \cup \{s_i\}$, possible).

   The compositional rule cp_3 states that if user's constraints specified over s_i are all met with the service policies provided by a service provider, then the service can be added (i.e. candidate service) in the service flow SF.

4. **Ordering relationships:** $cp_4 = (s_i \propto_{smc} s_j \wedge \text{value}(s_i) \prec \text{value}(s_j)$, $s_i \prec s_j$, must)
   $cp_5 = (s_i \propto_{syc} s_j \wedge \text{Ouput}(s_i) \subseteq \text{Input}(s_j)$, $s_i \prec s_j$, must).

   The rule cp_4 states that if s_i is of semantic type, such as OrderProduct, and s_j is of service type that semantically follows, such as Delivery, then the composition should put the ordering relationship, s_i before s_j, e.g. the product order should precede the delivery.
   cp_5 states the syntactic compatible services will have an ordering relationship according to their input and output relationships.

5. Policy-based Service Composition

In this section, we describe the process of service composition using the service policy statements and user policy. We have developed an automatic service flow composition algorithm and implemented a prototype in the E-government domain [10, 12, 11]. In this domain, the services discovered based on the rules are usually unique, i.e. there are as many competitive services as in E-commerce. In the electronic commerce domain, the services with the same functionalities can be provided

by many service providers. The problem is to select the one service that is best suitable for composition. Our approach is to automatically generate a service flow with candidate service sets, S= $\{S_1, S_2, \ldots\}$ and some ordering relationships among them. Unlike our previous approach, now each candidate service $S_i \in S$ is a set of functionally equivalent services, S_i =$\{s_1, s_2, \ldots\}$ where each $s_j \in S_i$ is a functionally equivalent service to s_k, but s_j and s_k are provided by different vendors and implements different policy rules. In order to select one appropriate service s_j from each S_i, we use the compositional rules including the compatibility among s_k in S_i and s_l in S_j at syntactic, semantic and policy level.

For example, consider an e-catalogue company's business policy on delivery to satisfy the customers, which states that: if purchase items are books and/or CDs and the delivery destination is within NJ, then use a delivery service that can deliver goods within 1 business day. Let's assume that mail service providers A and B are competitors in delivery. The delivery policy of A states that the merchandise can be delivered in NJ within 1 day, while B can deliver in NJ within 2 days. A's delivery service is compatible with the e-catalog company's delivery policy, while B's is not.

The process of a Web service composition consists of the following steps.

Step 1: (Identify component services and service flow) Given user's service request (a service flow), its component service types are identified according to the service type ontology.

Step 2: (Assign user policies) The global user policies (constraints) are assigned to the service flow and the local user policies are assigned to each component service in the flow.

Step 3: (Match user policies and service policies) Given a repository of Web services provided by individual organizations, for each component service in the user's service flow, search for services that match the semantic category of component service in the service flow. Each service in the list has its own service policies that it needs to abide. The service policies are matched with the user's local policies for the component service type. All the services whose policies match with user's local policies for the component service are put into a candidate list. This process repeats for all the component service types in the service flow.

Step 4: (Select a service from a candidate list and instantiate) Services in each candidate list need to go through the compositional rules to verify whether they can be composable with services in other candidate lists. When there are pairs or a sequence of services from different candidate lists that are composable according to the compositional rules, then one of these pairs or sequences of services are selected. The compositional rules eliminate the non-composable services from the candidate lists. If there is only one service in the candidate list that meets the user's local policy, then the selection of the service is unique. The selected services are service instances for the service flow.

Step 5: (Verify whether the service instances meet the global user policies for the service flow) When there are global user policies for the service flow, the combination of the selected services must meet the overall service flow policies in order to be a successful composition. The selection process in step 4 should consider different

services from the candidate lists, to meet the global user policies.

In the following, we present the algorithm for the service selection and composition described above.

ALGORITHM 1 **[Service Composition]**
Input ur=(us, gup, lup) a user requested service us, global and local user policies gup, lup
Output SF=$\{si_1, si_2, \ldots\}$: a service flow with service instances that meet the user and service policies.
BEGIN
Given SO /* Service type ontology */
/* select the service types user requires and prefers (obl, opt), and assign user policies */
for each $s_k \in SO$ such that $s_k \subset_{ST} us$ {
  $SF = SF \cup \{s_k\}$ for all services that is subtype of gs_i
  if($\exists lup(s_j) \in ur \vee$(mode($s_k$)=obligatory)/* obligatory or preferred */
    if($lup(s_j) \neq \emptyset$)
      $lup(s_k) = lup(s_j)$ /* assign local user policy */
    else $lup(s_k) = \emptyset$ /* assign no constraints */
  else $SF = SF - \{s_k\}$ }
$gup(SF) = gup(us)$
Given SR /* Web Service Registry */
foreach $(s_i, lup(s_i)) \in SF$ {
  Candidate(s_i) = $\emptyset$
  foreach $(s_k, P(s_k)) \in SR$
    if (type(P(s_k))=s_i) $\wedge$ match-policy(P(s_k), lup(s_i))
    /* if the service instance is the same service type and its policy matches with user policy y */
    Candidate(s_i) = Candidate(s_i) $\cup$ $\{s_k\}$ }
While (found = False) Do {
    foreach Candidate(s_i)
      case(Candidate(s_i)
        $\emptyset$: return(Exception)
        otherwise: { /* either 1 or more candidate services */
          si = ChooseOne(Candidate(s_i)
          SF1 =SF1 $\cup$ $\{si\}$
          Candidate(s_i) = Candidate(s_i) - $\{si\}$ }
    Given RB (composition rule base){
      /* check the selected service against the composition rules */
      result=verify-composition(SF1)
    if (result = True) { /* if the composition is valid */
      SF=SF1
      found = match-policy(P(SF), gup) } }
if (found=TRUE) return(SF)
else return($\emptyset$) /* Return an empty set, if no compatible service flow is found */
END

ALGORITHM 2 **[Match Policy]**
/* Match service policies $SP(s_i)$ and user's local policies $lup(s_i)$ */
Input: $lup(s_i), SP(s_i)$ /* Service Provider policy and local user policy for s_i

Output: T or F /* True if the service policy can satisfy user's constraints,
False otherwise.
if lup(s_i) = $\emptyset$
result = True;
else {
For each up =(upe, m) $\in$ lup(s_i)
foreach sp =(pe, m) $\in$ P(s_i) where pe and upe =(LHS op value)
if (LHS(pe) = LHS(upe))
op =op(upe)
result = (value(pe) op value(upe))
if result=False,
return(False); exit }
if result=True
return(True)

ALGORITHM 3 **[Verify Composition]**
/* checks if the component services abide by the composition rules across
organizational policies. Uses compositional rule base CPB */
Input: SF=$\{s_1, s_2, \ldots\}$
Output: True, if SF is a valid composition according to the composition rules
False, otherwise.

composable=True
for each $s \in SF$ {
for each $cp = (ce, a, dg) \in CPB$ {
if $\exists s_j, s_i \in SF \wedge a = (s_i \prec s_j)$ {
if ce=(Input(s_j) $\subseteq$ Output(s_i), composable$_{syn}$=true
if $\exists$ ($dc \in CDC \wedge dc \in ce$) $\wedge$ {value($dc(s_i)$) value($dc(s_j)$)} $\in CDVS$)
composable$_{sem}$=true
if ce=(area(P(s_i) $\cup$ area(P(s_j)$\neq \emptyset$, composable$_{pol1}$=true
if ce=(time(P(s_i) $\cup$ time(P(s_j)$\neq \emptyset$, composable$_{pol2}$=true }
if composable$_{syn}$ $\wedge$ composable$_{sem}$ $\wedge$ composable$_{pol1}$ $\wedge$ composable$_{pol2}$
composable=True
else composable=False } }
return(composable)

6. Policy Negotiation for Relaxing Compositional Rules

The policy matching process may result in an empty candidate list, when there is no service whose policy matches the user policy perfectly. In order to avoid failure in the service composition, we introduce a flexible matching which considers a negotiation process to match the user and service policies. Match failures may stem from various sources, such as the descrepancies among descriptors as a result of coming from different service ontologies. Thus, when there are discrepancies, the system should be able to check if the descriptors are synonyms. If that is the case, then the algorithm evaluates the values for successful matching. Depending on the degree of matching, we define the following classes of matches.

- Perfect Match: When the descriptors and values from service policy expressions and user policy expressions match, it is considered a perfect match. When the synonym of descriptors or synonym of values (in case of string values) are also considered a perfect match.
- Approximate Descriptor Match: When a descriptor of a user policy is a subclass of the descriptor used in a service policy, then the match evaluates their corresponding values. For example, assume a user specifies his hotel booking with a descriptor "room price" while the service used specifies the descriptor as "hotel price." The hotel price in the booking context implies a room price. Thus, their values are matched. This is considered an approximate descriptor match.
- Approximate Value Match: When the value of a descriptor for a user policy has larger range than that of a service policy, it is considered an approximate value match. For example, the user wants a "hotel" service, while a service provider's policy states that its service is "lodging" or "breakfast and bed." This case considers similar semantic nodes that share the common super-categorys, synonyms or instances.
- No match: When there is no match or no relationships exists as described in either one of the approximate matches between descriptors or between values, there is no match.

In case of no match, the system would not be able to find a service instance. On the other hand, for the approximate match, the system should be able to negotiate a match. There are different strategies one can apply. One is to elicit more information from the user to narrow the gap, and another may be that a service provider has a set of policies in different ranking, from the most desirable policy to the least desirable one. In the latter case, when there is an approximate match or no match with the most desirable policy, the next desirable policy is used for matching. The contingent policies are used as a set of negotiation strategies for the system. Similarly, a user could specify his preferences in advance, so that when there is no match, the user's policies are successively used for matching.

In case that there is a negotiable service policy whose conditions do not meet the user policy, the negotiation procedure for the policy is initiated. When the negotiation process results in mutually agreeable conditions, the policy is made to be policy compatible. For instance, suppose a user wants a product delivered within 5 days, but no service provider can deliver within that time period. In this case, if the policy on the delivery day by a service provider is negotiable, then the system should be able to initiate the service with an alternative possibility. If the user's constraints are non-negotiable, then the system should check if the service has an alternative policy to deliver within 5 days or less. The following definition allows a service policy that is flexible for negotiation, successively relaxing the constraint values to accommodate the user policy.

Definition 23. **[Flexible Service Policy]** Given a Web service s and the set of policy statements that s is subject to $P(s)=\{p_1, p_2, \ldots\}$, each policy statement

$p_i \in P(s)$ is defined as an ordered list of policy statements $(p_{i1} \prec p_{i2} \prec p_{i3} \ldots)$. Each P_{ij} is a pair (se, m), where se is a service policy expression to denote the conditions (constraints) the service s can satisfy and $m \in \{n, nn\}$ is a mode to denote the level of flexibility of the constraints in se, n for negotiable and nn for non-negotiable constraints.

The following shows an example of a flexible policy statement.
p_3 ={(DeliverProduct(x), n) (Provider= ExMail, nn) (Name= Deliver, nn) (Purpose= deliver a product, nn) (uri= http://www.ibm.com/deliver.wsdl, nn) ((geographic area= NE) $\vee$ (geographic area=South), n) ((delivery-days =7, n) $\prec$ (delivery-days=5 $\wedge$ fee=$10, n) $\prec$ (delivery-days=2 $\wedge$ fee=$25, nn))}
In this example, the policy statement p_3 for a delivery service provided by ExMail states that a delivery takes 7 days, but it is negotiable to 5 days in case the client is willing to pay a $10 fee, or 2 days with a $25 fee. Given these alternatives, the system first identifies all the services that can satisfy the above, and if no service exists according to the customer's request, i.e. the candidate list is empty, then it checks whether there are alternative offerings of each service provider, and negotiate if the alternative can meet the customer's needs.

This process of matching a customer's preferences and the service policies is viewed as a negotiation process, with additional conditions that need to be satisfied for consumption of the service. For example, assume that there is no service provider that matches a 5 day delivery request by the customer. The system can then inquire whether the customer is willing to pay an additional $10 to match his constraints. This way, the user can either accept or reject the offer to find another service provider's alternatives.

A formal algorithm for searching for a compatible policy and for negotiation can be easily devised as a modification of algorithm 1. This part is out of the scope of this paper and will be reported in a separate paper.

7. System Implementation

Figure 3 shows the architecture for our policy-based service composition system.

Service and Policy Specification and Advertisement: This module allows service providers to define the services they offer, policy statements that govern their uses, and capabilities of each service. The service definition language WSDL and RuleML are used in this component. It also uses service ontologies, defined in a domain, for semantic classifications as well as a set of descriptors for services. Once the service is defined, the service providers can advertise the service in a service registry, such as UDDI.

User service flow model: This component provides tools for users to specify desired services and user policies, using a user policy specification language and service ontologies which allow users to choose available service types and their descriptors in a domain to specify global and local user policies.

Policy Evaluation: This module evaluates the user policies against service policies and retains only services from service providers whose policies can accommodate the user policies.

Policy Negotiation: When there are no services that satisfy user policies, the negotiation module handles the relaxation of service policies or uses a lower ranked service policy.

Service Selection: This module selects services that are evaluated against user policies, and ensures the selected services obey compositional rules, by checking syntactic, semantic and policy-level compatibilities, as well as ordering and other compositional rules in the specific domain.

Service Flow Instance Specification: This component takes service instances in a service flow and combines the selected services, and generates the executable service flow specification, using a service flow language such as BPEL4WS, XLANG or WSFL. The resulting service flow is syntactically, semantically and policy compatible and therefore is sensible to be executed. The composition component requests the specific Web service and policy instances from a service provider.

Service Execution: The specified service flow is executed.

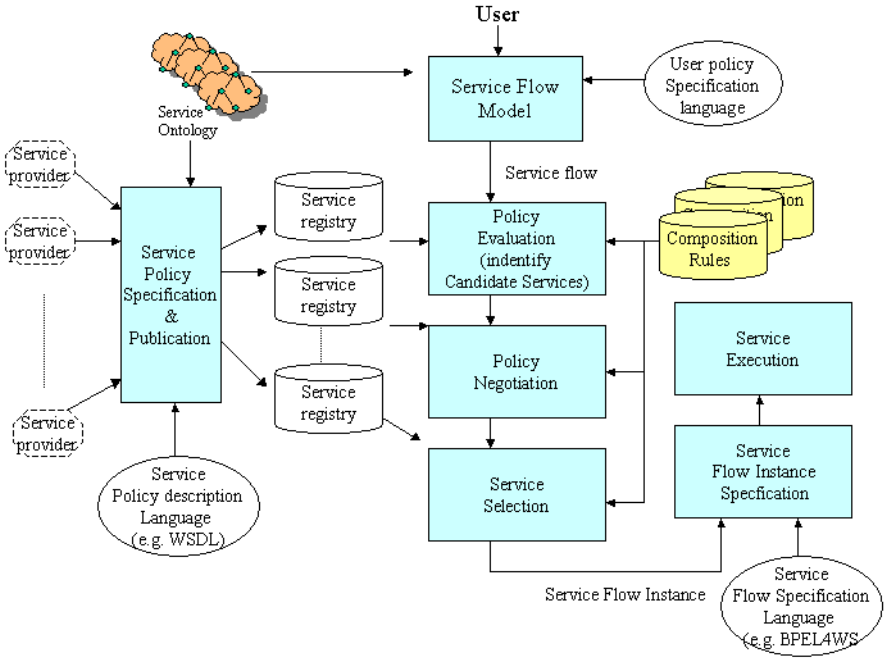

Figure 3. System Components for Automatic Service Flow Composition

```
<damls>
    <damls:class  openCellphoneAccount
<damls SubclassOf  openPhoneAccount > </damls:class>
<damls Provider   uri:http://www.cardissue.com/ >
<damls:function>
 <precondition:exp validCreditcard>
 <input:var  #Address, Name>
 <output:var  #cellphonenumber> </damls:function>
 <damls:prop
    <damls:var #quality-of-service>
    <damls:var  #max-response-time>
    <damls:var #geographic-area-covered> </damls:prop>
</damls >
```

Table 1. DAML-S description of a Web service

7.1. Service Implementation

The service is represented using standard DAML-S ontology notation [1]. While the standards of WSDL provide syntactic annotations for Web services that include tags mostly limited to network connection, input/output specifications, data types and how they bind to concrete ports, DAML-S provides semantic descriptions of Web services, using an ontology to describe service capabilities and properties. Web services in DAML-S are described with profiles, models, and groundings. The service profile describes information on what organization provides the service, what function the service computes, and other characteristics of the service. It includes contact information of a service provider, input and output of the service, and features such as the category of a given service, quality rating of the service, an estimate of the maximum response time, the geographic radius of a service, etc. The service profile provides the type of information about "what the service does." The profile is therefore used for advertising, registry, discovery, and matchmaking for the agent to find the service that meets its purpose.

The service model tells "how the service works," guiding a service-seeking agent to compose a service as a process, using inputs, outputs, preconditions, effects, and sub-processes. This information is used to perform a specific task, to coordinate the activities of the different participants, and to monitor the execution of the service. A service grounding specifies the concrete details of how an agent can access a service, such as communications protocol (e.g., RPC, HTTP-FORM, SOAP), and port numbers used in contacting the service.

Once an agent has located a service appropriate for its need, the service model and service grounding give enough information for an agent to make use of the selected service. Table 1 is an example of a service description using DAML-S. The service "openCellphoneAccount" is a subclass of "openPhoneAccount." It has a provider, and it computes a cell phone number as its output. It also specifies properties such as quality of the service, maximum response time, as well as geographic area covered by the service.

7.2. Service Policy Implementation

To accommodate the policy expressions, we introduce a policy ontology and policy class which has subclasses such as simple policies and composite policies. All policy classes have policy expression and mode. The policy ontology is linked to services by defining policy to be a subclass of ServiceProfile. Policy expression classes are classes in which we constrain the range of one of the service descriptors or properties such as "delivery delay", "delivery area", "input", etc. Table 2 shows the simple policy statements on delivery area, and delivery delay days. From these two simple

```
<daml:policy rdf:ID="Deliveryarea">
  <daml:onProperty rdf:resource="#delivery-area">
  <daml:hasValue rdf:resource="#New Jersey">
  <daml:hasProperty rdf:resource="#negotiable">
</daml:policy>

<daml:policy rdf:ID="Duration">
  <daml:onProperty rdf:resource="#delivery-duration">
  <daml:hasValue rdf:resource="#3days">
  <daml:hasProperty rdf:resource="#non-negotiable">
</daml:policy>
```

Table 2. Examples of policy statements

policy statements, one can define a composite policy expression class with logical operators, as shown in Table 3.

Alternatively, we can use OWL and RuleML to represent the policy statements for Web services as shown in Tables 4 and 5. Table 4 shows an extension of OWL (Web Ontology Language) [2, 16] with proposed tags to represent the policy rules and ontology. The actual body of the rule is linked by the resource link in the description and is represented in RuleML [4] as shown in Table 5.

7.3. Service Flow Implementation

A Web service flow is a composite service, consisting of individual atomic Web services. The service flow specifies the coordination among these component Web services. We adopt the formal workflow model developed in [7, 11] to represent a Web service flow, and use the Business Process Execution Language for Web Services [13] (BPEL4WS or BPEL for short), an XML-based workflow definition

```
<daml:policy rdf:ID="DeliveryPolicy">
  <daml:intersectionOf rdf:parseType="daml:conjunction">
    <daml:class rdf:resource="#Deliveryarea">
    <daml:class rdf:resource="#Duration">
  </daml:intersectionOf>
</daml:policy>
```

Table 3. An example of a composite policy expression

```
<policy rdfID="#P2">
  <owl:Topic rdf:ID="AutoloanAmount">
      <rdfs:resource="http://www.myloan.com/bizrules" />
      <rdfs:SubtopicOf Auto Loan />
  </owl:Topic>
  <hasRestrictRule rdfs:resource=
      http://www.myloan.com/amount_rule.ruleml/>(see Table 5)
  <hasProperty ref:resource="#negotiable"/ >
  <hasNegotiation  rdf:resource=http://service.com/negotiate"/>
  <belongTo  rdf:resource="#ABC Insurance, Inc."/>
 </policy>
```

Table 4. Proposed OWL extension for Policy Ontology

```
<damlRuleML:ind>amount_rule</damlRuleML:ind>
   </damlRuleML:_rlab>
<damlRuleML:_body
        <damlRuleML:atom>
           <damlRuleML:_opr>
              <damlRuleML:rel>Greater than $2000  <damlRuleML:rel>
           </damlRuleML:_opr>
         <damlRuleML:var>LoanAmount</damlRuleML:var>
        </damlRuleML:atom>
    </damlRuleML:_body>
<damlRuleML:_head>
    <damlRuleML:atom>
       <damlRuleML:cterm>
           <damlRuleML:_opc>
             <damlRuleML:ctor>Insert<damlRuleML:ctor>
           </damlRuleML:_opc>
           <damlRuleML:var>VerifyEmpolyment Status
           </damlRuleML:var>
      </damlRuleML:cterm>
    </damlRuleML:atom>
</damlRuleML:_head>
<damlRuleML:imp>
```

Table 5. Example of RuleML representation for a policy rule: If the amount of loan is greater than $2000, then invoke a service to verify the applicant's employment status.

language, to implement the composite services. BPEL documents are executable scripts that can be interpreted by business process engines to implement the described process. Each step in the process corresponds to a Web service provided by a business organization. It also provides tags to specify Web service coordination with the information necessary to link the various tasks in a process. Its transaction specification provides a framework to monitor the success or failure of each coordinated activity, and ways to cancel the process in case of failure. Table 6 shows the BPEL representation of the business process of a credit information provision service .

```
<process name="CreditInfoProcess"
    targetNamespace="http://acme.com/simpleloanprocessing"
    xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
    xmlns:lns="http://loans.org/wsdl/creditinfo"
    xmlns:creditdef="http://cimic.rutgers.edu/services/creditservice"
    xmlns:apns="http://cimic.rutgers.edu/services/creditinfo">
  <partners>
     <partner name="customer"
                 serviceLinkType="lns:creditinfoLinkType"
                 myRole="client"/>
     <partner name="provider"
                 serviceLinkType="lns:creditInfoLinkType"
                 partnerRole="provider"/> </partners>
  <containers>
     <container name="request" messageType="creditdef:SSNinfoMessage"/>
     <container name="creditInfo"  messageType="apns:creditInfoMessage"/>
  </containers>
  <sequence> <receive name="receive1" partner="customer"
                 portType="apns:creditInfoPT"
                 operation="provide" container="request"
                 createInstance="yes"> </receive>
     <invoke name="invokeprovider"
                partner="provider"
                portType="apns:creditInfoPT"
                operation="provide"
                inputContainer="request"
                outputContainer="CreditInfo"> </invoke>
     <reply name="reply"partner="customer"
                 portType="apns:creditinfoPT"
                 operation="provide" container="creditInfo"> </reply>
  </sequence>
</process>
```

Table 6. BPEL4WS representation of a service flow of obtaining credit information process

7.4. User Policy Implementation

In the following, we discuss how user specified policies can be implemented. For example, a user can specify that he wants to get some entertainment ticket on Wednesday evening near Washington Square (with likelihood of 50%, if the ticket costs over $20, and with likelihood of 90% if the ticket costs less than $20). Another example would be "I want a service of type X that is reliable (or trustworthy)." In this case, the meaning of what the user means by being reliable (or trustworthy) should be determined. For a software agent to compose services automatically, the agent should understand what types of activities are considered entertainment. This may depend on personal tastes. It also should know what defines the characteristics of being a reliable or trustworthy service.

We represent the user policies with XML-based semantic tags based on a topic ontology, as shown in Table 7. The user policy uses tags from an ontology so that the terms (e.g. entertainment) correctly refer to specific types of activities. The

user constraints expressed in terms of semantic tags help resolving the vocabulary differences between the tags that are used to express user policies and the tags used to express rules or services. We propose to use a user's service policy specification language similar to DAML-S. We call it user service policy language (USPL). This will describe the user policies with the semantic terms derived from an ontology the user picked.

```
<uspl Class:goal="home mortgage loan"
<uspl ref:source="http://www.myontology.com/bizreg">
<uspl Restriction
    <uspl cond:estimatedResponseTime ="#2 weeks" >
    <uspl  ref:source=(some uri where estimated Response Time is defined)>
    <uspl cond:LoanType="#residential mortgage">
    <uspl cond:LoanAmount="#$250,000">
```

Table 7. User Service Policy Language (USPL)

8. Related Work

The Web service technologies, such as the XML-based language WSDL to describe the services, UDDI for service registry, and SOAP for a uniform way of passing XML-encoded data, were developed to discover and determine right component services and compose a Web service. WSDL defines a Web service as collections of network endpoints or ports. DAML-S is a semantic markup for Web services for the properties and functionalities of Web services using a set of ontologies, thus capturing the semantics of what the service does (service capabilities), rather than just a collection of the network ports. The Web service standards facilitate service discovery [5, 17], but fall short on automatic composition of services.

Service discovery and composition is an active research area. The industry standards to support Web service composition include the Business Process Execution Language for Web Services (BPEL4WS) [13]. BPEL4WS provides a language for the formal specification of business processes and business interaction protocols that typically includes multiple Web Services composed with each other, i.e. static binding of services in the composition. However, BPEL4WS does not support an automatic discovery and composition of Web services on demand.

[20] proposed an agent technology based on generic procedures and customizing user constraints. They used a first-order logic programming language Golog for programming programming Web services and developed ConGolog interpreter which communicates with Web services written in Golog via the Open Agent Architecture (OAA).

The DAML-S Matchmaker [23] improves current UDDI architecture with semantic service descriptions. The matchmaker supports the service discovery based on the service capabilities. The matchmaker uses the subsumption relation between the classes to support more flexible matchings beyond the capabilities of UDDI.

[15, 14] use OWL and DAML-S for creating ontology for a domain and service description as proposed in our approach to provide semantics for dynamic service discovery and composition, and uses matching for semi-automatic (i.e. interactive) composition. However, the service discovery is made through matching service parameters (such as input, output) first, then through matching the so-called non-parameters (such as service type information) input by user interaction, while our approach starts with filtering first with the policy matching, then with the syntactic level matching. The exact match, generic match or specialization match are considered, where the service parameters can be exactly match, or the generic category concept is used, or the specialized sub-concept is used for matching..

[21] proposes an approach to compatibility and substitutability relationships among e-Services that can be derived by analyzing their interfaces and behaviors. In [6], a two-phase composition is proposed. First, a high level analysis uses service descriptors and semantic information and similarity evaluation to obtain compatible services, and second, a structural level analysis measures similarity by matching incoming and outgoing messages and their parameters. Similarly, [24] has three levels of service composition. First the semantic relatedness between the requested service and available services is used to select services. Second, the capabilities of selected services are considered, and finally syntactic analysis is performed to match the interfaces of services. [9] takes a similar approach using syntactic, operational and semantic similarity metrics for service composition.

In [25], quality of service parameters such as execution price, execution duration, reputation, reliability and availability are considered for Web service composition. A global service selection approach is based on linear programming techniques based on these quality of service parameters to compute optimal execution plans for composite services. [19], proposes an approach for the Web service selection and execution with mobile devices in the ubiquitous computing environment and uses other types of quality of service parameters such as the location of requesters of services, capabilities of computing resources on which services will be executed (e.g., CPU, bandwidth), and so on.

[22] uses the notion of (1) operational semantics composability where purpose and semantic categories of Web services need to be compatible in match making process for composition, (2) message-level compatibility where the data types of Web services are considered for composition, and (3) qualitative composability rules which checks preferences in fees, security and privacy for Web services to be composable. The matchmaking process produces several alternative composite services, called *plans*, and they introduced Quality of Composition (QoC) parameters, such as composition ranking, composition relevance, and completeness to select best plans.

Our approach is similar to the above proposals in using syntactic and semantic parameters, but differ in taking into consideration a service provider's policies as rules not just as parameters. These rules are also semantically related to be easily identified for discovering and matching. Unlike [22], our approach selects the best fitting service among alternative service providers during composition instead of delaying until the whole composition phase is finished. We also provide a formal model of service, service flow, ontology, search and matching algorithms, and

proposed user policy user constraints as personal policies, as opposed to the organizational policies. Our approach also considers the negotiation of services in case of policies among different service providers, or between user policies and service policies do not match exactly.

9. Conclusions and Future work

All real-life business processes (including Web services) have to live by various rules and constraints. In this paper, we have classified these rules into syntactic, semantic and policy rules that play a major role in discovery, selection and composition of Web services. These rules incorporate the knowledge that is necessary to select and compose Web services into a coherent service flow. We presented a formal model for Web service composition, called Service Flow composition model. This model includes the service ontology, service types and their descriptors and service policy statements imposed by service providers. The model also considers user policy. The services not only have to have policies compatible with the user policies in order to be selected in the composition, but they also have to satisfy syntactic, semantic and policy-level compatibility among services to result in a well-formed composition. Semantic-level compatibility considers the compatible semantic relationships among services, and policy level compatibility considers spatial, temporal, and value range compatibilities. We have provided an algorithm to select a service from functionally equivalent services by checking these policy compatibilities. We also introduced an approach for policy negotiations, in cases where no compatible policies exist.

Our approach for implementing Web service composition requires the knowledge (ontology) of Web services in a domain, and policy rules. We use OWL, DAML-S, RuleML and RDF standards and extensions of these standards to incorporate the policy-based composition for implementation.

Future work includes how to handle conflicting rules that are distributed and heterogeneous, described by different organizations. There may be precedence orders among organizational policies that can be more efficiently specified (e.g. in a functional specification) rather than listing them in order. The issue of maintaining the updates of rules and their effects on the composite Web services as well as that of using the rules description for e-negotiations needs to be further investigated. The rules ontology and the ontology for user policy specification need to match for correct identification of applicable rules. The issue of ontology interoperability also needs to be addressed.

References

1. DAML-S. http://www.daml.org/services/daml-s/0.7/CongoService.daml.
2. OWL Website. http://lists.w3.org/Archives/Public/www-webont-wg/2002May/att-0173/01-owl.html.
3. RosettaNet. http://xml.coverpages.org/rosettaNet.html.
4. RuleML Website. http://userpages.umbc.edu/ mgandh1/2002/06/DamlRuleML/.
5. A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. DAML-S: Semantic markup for Web services.

In *Proceedings of the First International Semantic Web Working Symposium (SWWS)*, Stanford, California, 2001.

6. V. D. Antonellis, M. Melchiori, and P. Plebani. An approach to Web Service compatibility in cooperative process. In *Proceedings of Workshop on Service Oriented Computing (SOC)*, Orlando, FL, 2003.
7. Vijayalakshmi Atluri, Soon Ae Chun, and Pietro Mazzoleni. Chinese Wall Security for Decentralized Workflow Management Systems. *Journal of Computer Security*, 12(6), November 2004.
8. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5), May 2001.
9. J. Cardoso and A. Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 12(3), 2003.
10. S. Chun, V. Atluri, and N. R. Adam. Domain knowledge-based automatic workflow generation. In *Proceedings of Database and Expert Systems Applications (DEXA)*, volume 2453 of *Lecture Notes in Computer Science*, Aix en Provence, France, September 2002.
11. Soon Ae Chun. *Decentralized Management of Dynamic and Customized Workflow*. PhD thesis, Department of Mangement Science and Information Systems, Rutgers Universi ty, Newark, 2003.
12. Soon Ae Chun, Vijay Atluri, and N. Adam. Dynamic Composition of Workflows for Customized eGovernment Service Delivery. In *Proceedings of The Second National Conference on Digital Government (dg.o 2002)*, LA, California, May 2002.
13. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, July 2001.
14. Evren Sirin and Bijan Parsia and James Hendler. Composition-driven Filtering and Selection of Semantic Web Services. *IEEE Intelligent Systems*, 18(4), July/August 2004.
15. Evren Sirin and James Hendler and Bijan Parsia. Semi-automatic Composition of Web Services using Semantic Descriptions . In *Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003*, Angers, France, April 2003.
16. J. Hendler and D. L. McGuinness. Darpa agent markup language. *IEEE Intelligent Systems*, 15(6), 2001.
17. M. Klein and A. Bernstein. Searching for services on the semantic web using process ontologies. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, July 2001.
18. M. Lewis. Supply chain optimization: An overview of rosettanet e-business processes. *e-AI Journal*, June 2000.
19. Zakaria Maamar, Quan Z. Sheng, and Boualem Benatallah. On composite web services provisioning in an environment of fixed and mobile computing resources. *Information Technology and Management Journal, Special Issue on Workflow and E-Business*, 5(3/4), 2004.
20. S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. In *Proceedings of the conference on Knowledge Representation and Reasoning*, April 2002.
21. M. Mecella, B. Pernici, and P. Craca. Compatibility of e-Services in a cooperative Multiplatform Environment. In *Procs. Of the 2nd VLDB-TES Workshop*, Rome, 2001.
22. Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *The VLDB Journal*, 12(4):333–351, 2003.
23. M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of The First International Semantic Web Conference*, 2002.
24. Jian Yang and Mike P. Papazoglou. Web component: A substrate for web service reuse and composition. In *CAiSE 2003*, volume 2348 of *Lecture Notes in Computer Science*. Springer, 2002.
25. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *Proceedings of The Twelfth International World Wide Web Conference (WWW'2003)*, Budapest, Hungary, 2003.