# Information Modeling for End to End Composition of Semantic Web Services

Arun Kumar, Biplav Srivastava, and Sumit Mittal

IBM India Research Laboratory,
Block 1, IIT Campus, Hauz Khas, New Delhi 110016, India
{kkarun, sbiplav, sumittal}@in.ibm.com

**Abstract.** One of the main goals of the semantic web services effort is to enable automated composition of web services. An end-to-end view of the service composition process involves automation of composite service creation, development of executable workflows and deployment on an execution environment. However, the main focus in literature has been on the initial part of formally representing web service capabilities and reasoning about their composition using AI techniques. Based upon our experience in building an end-to-end composition tool for application integration, we bring out issues that have an impact on information modeling aspects of the composition process. In this paper, we present approaches for solving problems relating to scalability and manageability of service descriptions and data flow construction for operationalizing the composed services.

## 1 Introduction

In many industrial applications such as mobile telephony, service providers face intense competition. In response, they need to continually develop compelling applications (e.g., movie recommendation system) to attract and retain end-users, with quick time-to-market. Much of this service/application development is currently done manually in an ad hoc manner, without standard frameworks or libraries, thus resulting in poor reuse of software assets. When a new service is needed, the desired capability is informally specified and then, an application developer must create this capability using component services available in-house or from known vendors. A component-oriented software development approach to application integration where each software is wrapped as a web service would offer substantial benefits in creating new services by leveraging web services composition.

Web services composition involves concepts from the AI domain as well as software engineering/programming domain. When viewed as a *program*, input and output parameters become important whereas when viewed as an *action*, the preconditions and effects become dominant [19]. However, most of the work in semantic web services community has focused on the AI approach of formally representing web service capabilities in ontologies like OWL-S [15], and reasoning about their composition using goal-oriented inferencing techniques from

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 476–490, 2005.
© Springer-Verlag Berlin Heidelberg 2005

planning[11]. An end-to-end view of web service composition starting from new service specification to an executing instance of the composed service is missing. To address this view, we have developed a prototype of a tool for facilitating new service creation and application integration for telecom service providers [1].

Our solution takes an end to end view and synergistically combines the AI approach of reasoning about web services functionality based on their preconditions and effects, and the distributed programming approach of selecting instances to optimize end-to-end runtime metrics, currently adopted by semantic web community and the industry, respectively. The solution drives the composition process right from specification of the business process in OWL-S, through creation of desired functionality using planning techniques into an abstract plan (workflow), through generation of a deployable workflow by selection and binding of appropriate service instances (specified using WSDL[1]), to finally deploying and running the composite service (specified using BPEL [5]).

The web service modeling efforts in the semantic web community, however, fall short of expectations of real world applications. During the course of our prototype development, we ran into several important modeling issues. We find that existing OWL-S service support is insufficient for the end-to-end composition vision because (a) the modeling does not allow for best knowledge engineering practices of modularity, conciseness and generality and (b) composed web services cannot be automatically operationalized due to lack of contextual information associated with input and output parameters.

A few alternative formalisms have been proposed to address OWL-S deficiencies but they focus more on foundational frameworks to overcome representational weaknesses [14, 21] rather than address ways for efficient, automatic, end-to-end composition. In this paper, we investigate information modeling issues for end-to-end composition of web services and propose related pragmatic extensions to OWL-S. Most specifically, our contributions through this work are:

- We differentiate web service types from service instances. This helps in organizing the expected thousands of web services into categories, allows the scaling of OWL-S ontology for inferencing and permits systematic treatment of non-functional requirements.
- We define support for context to disambiguate intended meanings of input and output (i/o) parameters. In other words, we introduce semantics for i/o parameters to construct the data flow after the control flow has been worked out by planning.

Representation and reasoning go hand-in-hand in any application. While we mainly focus on information modeling when using contingent planning for web service composition, we clarify at the outset that the proposed solution is also applicable if a different planner or a more complex planning formalism were to be used. More details of planning is given in the related works.

The rest of the paper is organized as follows. Section 2 describes a motivating scenario, highlighting the problems that surface while using OWL-S for end-to-

[1] http://www.w3.org/TR/wsdl

end composition. Section 3 discusses the issue of scaling of services ontology. Section 4 deals with generation of data flow to operationalize the composite service. Section 5 briefly describes our implementation. Section 6 gives some related work and Section 7 concludes the paper.

## 2 A Motivating Scenario

Suppose a telco wishes to offer its telecom and IT infrastructure to enterprise clients, by creating and deploying services that would enable automation of the client's business processes. An example of such a business process is a simple Customer Order Management System for a *FlowerDelivery* service. Assume that the registry of available services in the telco's infrastructure consists of a *Directory service*, one or more flower selling services - *FreshFlowerShop service, FragrantFlowerShop service,* etc., multiple credit card services such as *VisaCard service, MasterCard service* etc. and a *Dispatch service*. Figure 1 shows the ≺Inputs, Outputs, Preconditions, Effects≻ (IOPEs) of these services.

The figure also shows a feasible plan for the new service obtainable with an AI planner. The plan consists of invocations to directory service for obtaining

**New Service Requirement**

*Name*: FlowerDeliveryService
*Input*: PersonName, PersonName, FlowerName, NumOfFlowers, CreditCard
*Output*: OrderReceipt, DeliveryReceipt
*Precon*: PersonName **notNull**[a], PersonName **notNull**, NumOfFlowers <= 1000, FlowerName **oneOf** FLOWERLIST, CreditCard **hasBalance**
*Effect*: Packet **deliveredTo** PersonName, OrderReceipt **sentTo** PersonName, DeliveryReceipt **sentTo** PersonName, CreditCard **debited**

**Services in Registry**

*Name*: FreshFlowerShop Service
*Input*: Address, Address, FlowerName, NumOfFlowers
*Output*:OrderReceipt, Packet, Amount
*Precon*:Address **available**, Address **available**, FlowerName **oneOf** FLOWERLIST, NumOfFlowers <= 1500
*Effect*: OrderReceipt **sentTo** Address, Amount **available**, Packet **available**

*Name*: Directory Service
*Input*: Name
*Output*: Address
*Precon*: Name **notNull**
*Effect*: Address **available**

*Name*: FragrantFlowerShop Service
*Input*: Address, Address, FlowerName, NumOfFlowers
*Output*:OrderReceipt, Packet, Amount
*Precon*:Address **available**, Address **available**, FlowerName **oneOf** FLOWERLIST, NumOfFlowers <= 1200
*Effect*: OrderReceipt **sentTo** Address, Amount **available**, Packet **available**

*Name*: VisaCard Service
*Input*: Amount, CreditCard
*Output*: Authorization
*Precon*: Amount **available**, CreditCard **hasBalance**
*Effect*: CreditCard **debited**, Authorization **available**

*Name*: CheapFlowerShop Service
*Input*: Address, Address, ItemCode, NumOfItems
*Output*:Acknowledgment, Packet, Charges
*Precon*:Address **available**, ItemCode **oneOf** ITEMLIST, Address **available**, NumOfItems <= 100
*Effect*: Acknowledgment **sentTo** Address, Amount **available**, Packet **available**

*Name*: Dispatch Service
*Input*: Authorization, Address, Address, Packet
*Output*: DeliveryReceipt
*Precon*: Authorization **available**, Address **available**, Address **available**, Packet **available**
*Effect*: DeliveryReceipt **sentTo** Address, Packet **deliveredTo** Address

**A plan for FlowerDelivery Service**

// I is for input and O is for output
*Step 1*: Directory Service1(I:N1, O:A1)
Directory Service2(I:N2, O:A2)
*Step 2*: FreshFlowerShop Service(I:A3, I:A4, I:FN, I:NUM, O:ORCPT, O:PKT, O:AMT)
*Step 3*: VisaCard Service(I:AMT, I:CC, O:AUTH)
*Step 4*: Dispatch Service(I:AUTH, I:A5, I:A6, I:PKT, O:DRCPT)

[a] The string in **boldface** is the predicate and the associated strings are its parameters

**Fig. 1.** The FlowerDelivery Service composition scenario

addresses of the sender and the recipient. This is followed by invocation to one of the flower shop services for obtaining desired flowers. An order receipt is sent to the sender. Pricing details are passed on to the credit card payment gateway and on successful authorization, shipping details and the flower packet are handed over to the dispatch service for delivery. A delivery receipt is now sent to the sender. In generating an end-to-end deployable plan for this scenario, we faced the modeling problems listed below.

**Service Types Vs Instances**: The scenario described above has multiple flower shop services. These services can offer different kinds of flower packages (e.g. bouquets, decoration styles etc.), but essentially they are all flower shops. This fact can be very useful for efficient representation of such services. Unfortunately, OWL-S does not capture the notion of service *types*. Each OWL-S description currently pertains to a single instance of a service.

There are several drawbacks with this approach. First, given the requirements of the new *FlowerDelivery service* as shown in Figure 1, the composition tool needs to consider and evaluate each and every instance of such FlowerShop kind of service available. This seriously affects the performance and scalability of the composer (planner) since there may be hundreds of such FlowerShop service instances available whereas for obtaining a feasible functional composition, different instances of similar kinds of services need not be considered.

The second drawback of the current approach is related to standardization. Since there is nothing common defined for similar services, a composition tool cannot infer anything about the degree of similarity or dissimilarity of these services. In our scenario, each of the FlowerShop services have different profiles even though their underlying process model may be the same. In Figure 1 the FreshFlowershop Service has a different profile than that of CheapFlowerShop Service. We can try to rectify this by adding some relations in the ontology such as *OrderReceipt* **isEquivalentTo** *Acknowledgment*, but it does not always work as in the case of FlowerName and ItemCode. Since the profile model is used to advertise a service, these two would appear as different kinds of services.

The third drawback relates to the service grounding part. Since grounding is specific to each service instance, a composition taking that into account is less likely to be stable - changes at the level of individual service instance operation take place much more frequently than those at the level of service functionality. The composition becomes prone to small implementation changes made to the service instance. For example, if the VisaCard Service originally supported 64-bit encryption protocol (specified in its grounding, not shown in figure) then the plan in Figure 1 may break if VisaCard service upgrades to 128-bit encryption.

**Support for Data Flow Construction**: When we seek to operationalize composed plans, we are in fact generating programs. A program contains the specification of both its control flow (the dependence among activities) and the data flow (the dependence among data manipulations). Planning techniques can be used to easily generate the control flow for the composite service given the pre-

condition and effect information for available service types, but generating the complete data flow needs reasoning with contexts of inputs and outputs.

For the full composition, data flow has to be produced between dependent services to make the plan executable. In Figure 1, the FreshFlowerShop Service accepts two addresses - one of these addresses is that of the sender and the other is that of the recipient. Even if this distinction of their semantics is not necessary for generating the control flow, it could be important for the data flow. Specifically, the two addresses can have different meanings and different data (message) types. In the FlowerDelivery scenario, to determine the relation between input/output of component services, we must (automatically) figure out things such as the following:

**DF1:** CreditCard information from the user goes directly to the VisaCard Service.

**DF2:** distinguish the semantics of the two Address inputs each to the FreshFlowerShop Service and the DispatchService.

**DF3:** map the Address outputs from invocation of the two DirectoryService instances to the Address inputs of both the FreshFlowerShop Service as well as the DispatchService.

**DF4:** DeliveryReceipt from DispatchService and OrderReceipt from FreshFlowerShop Service together constitute the output for the user.

In programming languages this issue is resolved by specifying an ordering among the parameters of a function or procedure. A human developer could then look at the language specification and specify the parameters accordingly. However, in the web service composition scenario, software programs cannot automatically derive and interpret semantics of all parameters just from the available ordering. The context for the inputs and outputs need to be made explicit. One provision to model the semantics associated with the input/output parameters is by creating new concepts in the domain ontology. But this will make the ontology large and brittle. The latter consequence is well understood in knowledge engineering[18] and that is the reason very specific terms are not recommended in an ontology.

In the following two sections, we will look into these modeling issues more closely and propose approaches for resolving them.

## 3 Scaling Services Ontology

In order to work with large collections of web services – categorizing them, supporting multiple views [10], standardization and for stable functional compositions – we need to support web service *types* that are described independent of individual web service *instances*. The approach of separating type definitions from instance definitions has been used successfully in data models for distributed systems management [13] and has various modeling benefits. In this section, we first delve into the classification of services into types and instances, then look at the support for this classification in OWL-S and finally discuss the issue of modeling non functional service capabilities.

### 3.1 Classifying Services into Types and Instances

Our proposal raises the question of what kind of relationship a web service type has with its various instances. A web service type captures the core functionality of a class of web services. Individual instances belonging to that class of services must adhere to the basic type definition but may be allowed to offer minor variations under some constraints. An important desiderata is that any composition which is produced with the web service type should be still valid when any of its web service instance is selected. This is ensured if the precondition of a web service type is more specific than precondition of its instances and its effect is more general than effect of any of its instances. We can summarize the relationship as:

$$\textit{If } S^{instance} \textit{ is of } S^{type},$$

$$1.\ S^{type}_{precondition} \vdash S^{instance}_{precondition} \textit{ and}$$

$$2.\ S^{type}_{effect} \dashv S^{instance}_{effect}$$

The above relationship states that the precondition of the service type entails the precondition of the service instance so that the latter is satisfied whenever the former is. For effects, the reverse is true. With this, given a web service request $R$, when the request matches a service type ($R \bowtie S^{type}$), the relationship between $R$ and the web service instances would be:

$$1.\ R_{precondition} \bowtie S^{type}_{precondition} \Rightarrow (\forall S^{instance} : S^{type})\ R_{precondition} \bowtie S^{instance}_{precondition} \textit{ and}$$

$$2.\ R_{effect} \bowtie S^{type}_{effect} \Rightarrow (\forall S^{instance} : S^{type})\ R_{effect} \bowtie S^{instance}_{effect}$$

According to it, if a request $R$ matches a web service type, where matching can be exact or defined over a range as in [16], all the instances of the web service type will also match. While this relationship would guarantee that compositions are valid when the abstract plan is concretized, it can be overly restrictive because the precondition of the web service type is required to be more specific than all its instances. We will call this as the *strict* relation. To relax the restriction, we use the insight that eventually each web service type referred in the abstract plan will be instantiated by only one web service instance. Therefore, as long as we could guarantee that if a request $R$ matches a web service type, some but at least one instance of the web service type will also match, the abstract will be successfully concretized and the composition will succeed. That is,

$$1.\ R_{precondition} \bowtie S^{type}_{precondition} \Rightarrow (\exists S^{instance-i} : S^{type})\ R_{precondition} \bowtie S^{instance-i}_{precondition} \textit{ and}$$

$$2.\ R_{effect} \bowtie S^{type}_{effect} \Rightarrow (\exists S^{instance-j} : S^{type})\ R_{effect} \bowtie S^{instance-j}_{effect} \textit{ and}$$

$$3.\ \textit{instance-i} = \textit{instance-j}$$

The decision of whether to follow the *strict* or *relaxed* relationship during domain modeling is one of balancing tradeoffs. With the former, the abstract plans can be automatically concretized because all its service instances are guaranteed to

preserve composition. With the latter, a service type has to be more specific than at least one of its instance and this would simplify building of the services ontology (e.g., more instances for a type). During the concretization of abstract plan, all instances might need to be explored for say, optimality. In that case, additional constraints will have to be checked for instances whose preconditions are more specific than that of their type. Checking these additional constraints may require the intervention of a developer.

We adopt the *relaxed* relation above as the guideline for our domain modeling. In Figure 2, FlowerShopService type captures the category of flower shop services whose instances are Fresh, Fragrant and Cheap Flower shop services. The preconditions of FlowerShopService type is the same as Fresh and Fragrant service instances (disregarding the non-functional requirement of *NumOfFlowers*, see below) but different from precondition of Cheap Flower service which has a restriction on *ItemCode*.

**New Service Requirement**
*Name*: FlowerDeliveryService
*Input*: PersonName *(From)*, PersonName *(To)*, FlowerName, NumOfFlowers, CreditCard
*Output*: OrderReceipt, DeliveryReceipt
*Precon*: PersonName *(From)* **notNull**, PersonName *(To)* **notNull**, CreditCard **hasBalance**, FlowerName **oneOf** FLOWERLIST, NumOfFlowers <= 1000
*Effect*: Packet **deliveredTo** PersonName, OrderReceipt **sentTo** PersonName, DeliveryReceipt **sentTo** PersonName, CreditCard **debited**

**Service *Types* in Registry**

*Name*: FlowerShop Service Type
*Input*: Address *(From)*, Address *(To)*, FlowerName, NumOfFlowers
*Output*:OrderReceipt, Packet, Amount
*Precon*:Address *(From)* **available**, Address *(To)* **available** FlowerName **oneOf** FLOWERLIST
*Effect*: OrderReceipt **sentTo** Address *(From)*, Amount **available**, Packet **available**

*Name*: Directory Service Type
*Input*: Name
*Output*: Address
*Precon*: Name **notNull**
*Effect*: Address **available**

*Name*: Dispatch Service Type
*Input*: Authorization, Address *(From)*, Address *(To)*, Packet
*Output*:DeliveryReceipt
*Precon*:Authorization **available**, Address *(From)*, Address *(To)* **available**, Packet **available**
*Effect*: DeliveryReceipt **sentTo** Address *(From)*, Packet **deliveredTo** Address *(To)*

*Name*: CreditCard Service Type
*Input*: Amount, CreditCard
*Output*: Authorization
*Precon*: Amount **available**, CreditCard **hasBalance**
*Effect*: CreditCard **debited**, Authorization **available**

**Service *Instances* in Registry**
— Same as the services in Registry of Figure 1 —

**A logical Plan for FlowerDelivery Service**
// I is for input and O is for output
*Step 1*: Directory Service Type(I:N1, O:A1)
Directory Service Type(I:N2, O:A2)
*Step 2*: FlowerShop Service Type(I:A3, I:A4, I:FN, I:NUM, O:ORCPT, O:PKT, O:AMT)
*Step 3*: CreditCard Service Type (I:AMT, I:CC, O:AUTH)
*Step 4*: Dispatch Service Type (I:AUTH, I:A5, I:A6, I:PKT, O:DRCPT)

**Fig. 2.** Proposed modeling for FlowerDelivery scenario with svc. types, roles & NFCs

### 3.2 Support for Service Types

As previously noted, currently OWL-S is designed to model a single web service instance [15]. It consists of a *ServiceProfile* that describes the interface of the service, a *ServiceModel* that describes the details of its operation and a *ServiceGrounding* that provides information about interoperation with that service using messages.

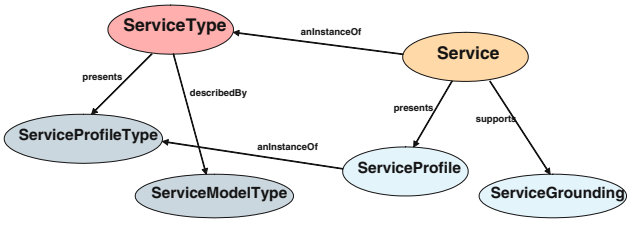


**Fig. 3.** *Modified OWL-S* upper ontology

We propose to separate the representation of web service type definitions from instance definitions. This means that the OWL-S upper ontology needs enhancements to have a *ServiceType* class hierarchy in addition to the *Service* hierarchy (see Fig. 3). The *ServiceProfile* model of the current OWL-S *Service* hierarchy is essentially a type definition and can be moved to the *ServiceType* hierarchy. The *ServiceProfile* of an instance will now point to the corresponding *ServiceProfileType* for structure, and contain the actual values of the Inputs, Outputs, Preconditions and Effects (IOPE) parameters applicable for that service instance.

*ServiceGrounding* is a concept that applies to instances rather than types and can stay as it is. *ServiceModel* should ideally be encapsulated inside the service interface and not exposed to the external world. Making the model visible outside the service is useful only if it describes the conversational aspect of the web service that would be needed to interoperate with it. In such a case, it should be included in the *ServiceType* hierarchy since a common conversation model should be applicable to all instances of a service type. In other words, we propose to have an ontology for service types that consists of *ServiceProfileType* and *ServiceModelType* model. This would be in addition to an ontology for service instances that consists of a *ServiceProfile* and a *ServiceGrounding*.

With this representation, a new kind of service can be specified in the ontology by adding an object of type *serviceType*, without having to create an actual running instance first. This is not possible in the current OWL-S ontology. Creating an object of ServiceType would include defining the parameters in its profile by populating the *ServiceProfileType* model, and describing the conversation model by populating the *ServiceModelType* model. Each actual running instance of this web service would be represented by an object of type *Service* and include a reference to its *ServiceType* object. Its *ServiceProfile* model would contain the actual values of the parameters listed in the corresponding *ServiceProfileType*.

### 3.3 Modeling Non Functional Service Capabilities

The functional capability (FC) of a web service describes its core functionality. It is expressed through IOPEs that capture the transformation performed by this service. The non-functional capabilities (NFCs), on the other hand, help

in characterizing the service further by capturing its optional features, such as cost, QoS etc. OWL-S has provision to represent NFCs through profile attributes which may contain parameters other than the functional IOPEs.

Since NFCs inherently capture properties of service instances (and not of types), they are not needed during functional composition. In contrast, FCs form the core of the functional composition process. NFCs play an important role during selection of appropriate service instances in order to meet the end-user requirements. The current OWL-S only deals with service instances and therefore all the functional as well as non-functional attributes are in the ServiceProfile. In our modified OWL-S upper ontology (presented in Figure 3), the FCs get represented in ServiceProfileType. The ServiceProfile of an instance inherits these FCs from the ServiceProfileType and adds the NFCs to it.

In some domains it may be desirable to model certain service features as mandatory for all instances of a service type. In military applications, for example, it may be necessary to make all service instances secure. For such domains, it seems logical to model NFCs such as security in the service type itself. These non-functional capabilities now form a part of the core functionality. They are included in ServiceProfileTypes and are utilized in selecting service types during the logical composition phase.

Table 1 shows the NFCs for the FlowerShop service instances. Note that NumOfFlowers is not modeled in FlowerShopService type in Figure 2, as opposed to Figure 1 where it is included as a part of precondition of the services. The NumOfFlowers requirement in the desired composite service was for $\prec 1000$ and this would be ensured while picking instances.

**Table 1.** Representing Instances for FlowerShop Services

| Instance Name | Security Level | Response Time | Max #Flowers |
|---|---|---|---|
| FreshFlowerShopService | Restricted | 70 sec | 1500 |
| FragrantFlowerShopService | Confidential | 240 sec | 1200 |
| CheapFlowerShopService | Public | 30 sec | 100 |

## 4 Generation of Data Flow

One of the main differences between knowledge engineering and programming, as described in [18][2], is that while logic sentences in the former tend to be self-contained, the statements in a program depend heavily on surrounding context. To operationalize the workflow of the composite service, we need support for incorporating context with IO parameters of component web services. One option is to introduce specific terms in the domain ontology, one for each possible concept and each valid context. However, this makes the ontology large and brittle. The consequence of this is well understood in knowledge engineering [18] and that is the reason very specific terms are not recommended in an ontology.

[2] Chapter 8, Page 222.

The semantics of each input/output parameter can be expressed along two dimensions. The first one specifies the meaning of the parameter as intended by the service designer. For instance, the designer of FreshFlowerShop Service could designate one Address parameter as the *From* address and the other one as the *To* address. The second dimension is dictated by the composition of which this service becomes a component. If in a composition, the input parameter Name to the Directory Service is assigned the label *From*, the output Address should be assigned the same label.

### 4.1 Context Resolution Using Roles

We seek to solve the problem of context resolution by explicitly encoding the context for inputs and outputs using the notion of *roles*. A role is a term that *qualifies* a concept. That is, for any concept $\varphi$, $(\psi\ \varphi)$ specifies that the role played by $\varphi$ is $\psi$. Roles are optionally specified on the inputs and outputs by the service developer. They come from a separate ontology, and are structured and standardized in a domain similar to concepts. Figure 4 shows a sample role ontology for roles that could be played during categorizing offerings, item transfer and expertise lookup. Depending on need, a parameter can have either one, multiple or no specific roles. In Figure 2, the user assigns the roles of *From* and *To* to the two input addresses in the input specification.

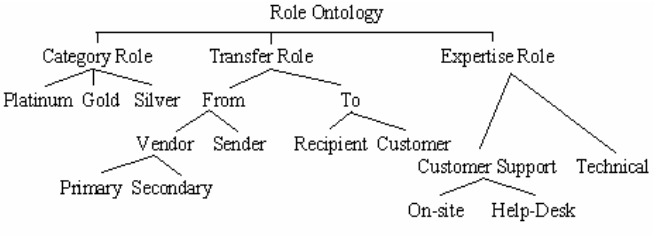


**Fig. 4.** Sample Role Ontology

In [8], the authors give an extensive coverage of how context is handled in knowledge representation in AI. Their solution is to explicitly model context as a resource and they introduce terms to specify *lifting rules* so that propositions could be generalized across contexts to serve their data aggregation application. In comparison to the roles, the context of [8] means that if *ist*($c_i$, $\varphi$), the proposition $\varphi$ is true in context $c_i$. The two usages can be combined - for example, *ist*($c_i$, $\psi\ \varphi$) means that the proposition $\varphi$ has the role $\psi$ in the context $c_i$. Currently OWL-S does not support the notion of roles for service representation.

A key motivation for defining roles is that they should be generic in nature. If a role can be attached with multiple concepts, it reduces brittleness in the ontology by eliminating the need for specific terms. Introduction of roles, however, requires the developer to define the roles played by the input and output parameters in her specification of the service.

### 4.2 Role Propagation

Roles can be propagated so that input or output or both can be associated with new roles in the presence of roles coming from requirement specification and/or those of other services. New roles can be acquired while matching a specification with a service instance or from the input to the output of a service and vice-versa. When a role can or cannot be propagated will be specified by the service modeler for a service using some rule language like SWRL. Some rules are given in Figure 5. Rule 1 says that a role from the specification can be propagated to input of an instance. In Figure 2, using this rule, the roles *From* and *To* from the requirements specification are transferred to the corresponding inputs of the Directory service instances. Rule 2 says that roles can be propagated from the input to the output of a service. In our scenario, using this rule, the role of input *From* or *To* to the Directory service is carried to its output. Rule 3 says that role can be propagated from the output of one service instance to input of a successor, if the successor does not have roles assigned for its inputs. Rule 4 says that if the inputs (or outputs) of a service has parameters of the same type (with no roles assigned), no role is propagated. This is because having the same parameter type introduces ambiguity for propagation. For example, if roles were not assigned to the two Address inputs of the FlowerShop Service by the service developer, no role for them can be inferred automatically from the composition.

Given service $S$ with inputs $I$ and outputs $O$

1. IF $\psi_I^{instance} = \{\}$ and $\psi_I^{spec} \neq \{\}$
   THEN $\psi_I^{instance} = \psi_I^{spec}$

2. IF $\psi_O = \{\}$ and $\psi_I \neq \{\}$
   THEN $\psi_O = \psi_I$

3. IF $\psi_I^{next-instance} = \{\}$ and $\psi_O^{prev-instance} \neq \{\}$
   THEN$\psi_I^{next-instance} = \psi_O^{prev-instance}$

4. IF $((\exists I_i, I_j, i \neq j$ s.t. $I_i^{type} = I_j^{type}) \vee$
   $(\exists\ O_i, O_j, i \neq j$ s.t. $O_i^{type} = O_j^{type})$
   THEN do not propagate

**Fig. 5.** Some rules for role propagation

The rules cannot be generic because role propagation should depend on the way the service processes its inputs to generate outputs, something best known to the service developer. Therefore, we advocate that rules be provided on a per service basis, taking into account the way service has been implemented.

### 4.3 Data Flow Construction Using Roles

Going back to the data flow problems raised in Section 2, credit card and receipts can be deduced from the IO data types of the services in the composition. For address, the role propagation rules can be used with Directory Service to automatically deduce the data flow as follows. Rule 1 will associate different roles

*From* and *To* to the two directory Service instances and Rule 2 will propagate them to the outputs. Now, using the roles on outputs of Directory Services, the data flow with the next services - FlowerShop Service and Dispatch Service - is found by simple role alignment.

Assigning roles has two benefits - on the one hand, role disambiguates between multiple instances of the same concept in a service profile thus clarifying the intended usage of the concept in the service. On the other hand, it enables the creation of a context using which the data flow from other service to this service. Association of roles with parameters of a web service also provides an extra dimension for matching requirements. A match-making tool would try to search services for which the input parameters have roles that fit the description of the requirement.

### 4.4 Discussion

In [8], the authors point out that while the aim of a context mechanism is to qualify information, in the semantic web, the mechanism should additionally be able to handle its large scale and distributedness, i.e., the mechanism should yet be easy to inference with and easy to use. They argue that the best way to approach a context mechanism is by focusing on the needs of a specific application, and they focus on data integration. In this section, we presented roles as a pragmatic approach for resolving ambiguities during service composition.

However, there are other kinds of situations where roles are not sufficient. To illustrate, in situation DF3 mentioned in Section 2, the data flow could be constructed using roles under the assumption that an output from a service is unconstrained. For example, the solution assumed that an output can serve as input for more than one service later in the plan. Modeling of such assumptions is not handled. Full automation of data flow construction needs full support for contextual reasoning.

## 5 Implementation

We have implemented a prototype tool that allows end to end composition of web services [1]. The composition is basically done in two stages:

1. **Logical Composition:** This phase provides functional composition of *service types* to create new functionality that is currently not available.
2. **Physical Composition:** This phase enables the selection of component *service instances* based on non-functional (e.g. QoS) requirements, that would then be bound together for deploying the newly created composite service.

In the absence of tools to support our modified OWL-S ontology (introduced in Section 3), we used the current OWL-S Profile model to represent the service types. The combination of SNOBASE[3] and PSME [6] was used as registry for service type definitions. The logical phase uses service types for creation of

[3] http://www.alphaworks.ibm.com/tech/snobase

abstract composition(s) delivering the desired functionality. Once an abstract composition is obtained then the physical phase uses a registry of web service instances to select appropriate instances satisfying the user's non-functional requirements. We used Web Service Matchmaking Engine (WSME) [7] as a service instances registry to contain Web Service Description Language (WSDL) specifications for each advertised service along with its non-functional capabilities. The tool produces deployable and executable composite service represented in BPEL language.

We have implemented a Role Ontology for the flower shop scenario and modified our OWL-S service descriptions to incorporate role information along with each input/output parameter. The roles help in guiding the data flow construction as discussed.

## 6 Related Work

Information modeling for end-to-end web services composition is challenging. The composition problem poses challenge to existing planning methods in representation of complex actions, handling of richly typed messages, dynamic object creation and specification of multi-partner interactions [20]. We are using *limited* contingent planning for generating the control flow during web service composition [12, 1]. Contingent planning (CP) deals with planning for domain with incomplete knowledge and sensing. In the case of web services, the value of all logical terms may not be known in the initial state but they can be found at the runtime using sensing actions. Our planner can also take input about selective conditions from the user and then uses it to efficiently focus search. But the modeling solutions are applicable independent of the form of planning used.

Many authors have raised issues about OWL-S. For example, [14] gives a list of problems with OWL-S: conceptual ambiguity (e.g., what is a service?), poor axiomatization (there is no firm concept or relation hierarchy and several relations take placeholders in the domain or range), loose design (support for multiple views is needed at different levels of granularity), and narrow scope on information systems which does not make distinction with real world objects and events. However, it does not deal with scalability, service quality and assessment which is essential for end to end composition.

The end-user requirements for the composite service, like that of any software program, can consist of functional as well as non-functional requirements. The non-functional attributes relate to performance, reliability and other user-acceptance issues. [4] describes how such requirements can be qualitatively arranged as goal structures and used to design systems. Their framework allows treating requirements as potentially conflicting or synergistic goals to achieve during the software development process. A middleware for composing web services with QoS in mind is presented in [22]. We characterize how to model non-functional capabilities for the composition process.

Our solution regarding generation of data-flow is related to [9] which describes an environment for building reusable ontologies based on the concept of

roles. This work informally defines role as a characteristic that a basic domain concept exhibits in a context. We can use their tool to build role ontology in parallel with the domain ontology. Our solution is in the spirit of [3] where a formal framework was proposed for data integration based on dynamic logic. An alternative proposal to OWL-S is the SESMA [17] model which directly handles inputs and outputs. Here, a notion of conversation data set is introduced to hold the input and output variables with values, and these could be evaluated as part of reasoning with the service's preconditions and effects.

Problem of data flow analysis in programs has been studied extensively in the compiler domain. Gathering knowledge of how data flows in a program is conceptualized using the life-cycle and scope of variables [2]. Analyzing the data flow should be differentiated from generation of the flow itself. The former arises when component modules have already been integrated and bounded using variables. The latter presents itself while the developer is trying to integrate the modules, and is therefore the harder issue to resolve.

## 7 Conclusion

Current efforts in the world of semantic web services focus on formally representing service capabilities and on reasoning about composition of services using AI techniques. We presented several issues that arise when we view the composition of web services from an end-to-end perspective. Concretely, we delved into the aspects related to scaling of service ontologies and support for generation of data flow information to operationalize a composite workflow. We discussed the need for separating service type from service instances and introduced the notion of roles to help disambiguate the semantics of IO parameters. We showed that the proposed guidelines are helpful in an end-to-end composition scenario.

## References

1. V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A Service Creation Environment based on End to End Composition of Web Services. In *Proceedings of the 14th International World Wide Conference*, May 2005.
2. A. V. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, 1986.
3. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Principles of Knowledge Representation and Reasoning*, pages 2–13, 1998.
4. L. Chung and B. A. Nixon. Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In *International Conference on Software Engineering*, pages 25–37, 1995.
5. F. Curbera et al. Business Process Execution Language for Web Services. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2002.
6. P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder. Parameterized Semantic Matchmaking for Workflow Composition. Technical Report RC23133, March 2004.

7. C. Facciorusso, S. Field, R. Hauser, Y. Hoffner, R. Humbel, R. Pawlitzek, W. Rjaibi, and C. Siminitz. A Web Services Matchmaking Engine for Web Services. In *Proc. 4th Intl. Conf. on e-Commerce and Web technologies*, Sep. 2003.
8. R. Guha, R. McCool, and R. Fikes. Contexts for the Semantic Web. In *Proceedings of the International Semantic Web Conference*, 2004.
9. K. Kozaki, Y. Kitamura, M. Ikeda, and R. Mizoguchi. Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of "Role" and "Relationship". In *13th Int. Conf. on Knowledge Engg. and Knowledge Management*, 2002.
10. R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, and D. Fensel. Semantic Web Services: Description Requirements and Current Technologies. In *International Workshop on Electronic Commerce, Agents, and Semantic Web Services*, September 2003.
11. S. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web.*, 16(2):46–53, March/April 2001.
12. A. Mediratta and B. Srivastava. User-driven search control in contingent planning and an application. In *IBM Research Report*, 2005.
13. Common Information Model (CIM) Metrics Model, Version 2.7. Distributed Management Task Force, http://www.dmtf.org/standards/documents/CIM/DSP0141.pdf, June 2003.
14. P. Mika, D. Oberle, A. Gangemi, and M. Sabou. Foundations for Service Ontologies: Aligning OWL-S to DOLCE. In *Proceedings of the 13th International World Wide Web Conference*, 2004.
15. OWL Services Coalition. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/ owl-s/1.0/owl-s.html, Nov. 2003.
16. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference, LNCS 2342*, pages 333–347. Springer-Verlag, 2002.
17. J. Peer. Semantic service markup with SESMA - language specification, version 0.7. In *http://elektra.mcm.unisg.ch/pbwsc/docs/sesma_0.7.pdf*, 2004.
18. S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach (First Ed.). Prentice Hall Publication, ISBN: 0131038052., 1995.
19. Marta Sabou, Debbie Richards, and Sander van Splunter. An Experience Report on using DAML-S. In *Proceedings of 12th International World Wide Web Conference, (WWW)*, May 2003.
20. B. Srivastava and J. Koehler. Web Service Composition - Current Solutions and Open Problems. ICAPS 2003 Workshop on Planning for Web Services, 2003.
21. WSMO. Web Services Modeling Ontology. http://www.wsmo.org, 2004.
22. L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. In *IEEE Transactions on Software Engineering*, pages 311–327, 2004.