RI 05001 02 February 2005 Computer Science

IBM Research Report

Information Modeling for End to End Composition of Semantic Web Services

Arun Kumar, Sumit Mittal and Biplav Srivastava
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

Information Modeling for End to End Composition of Semantic Web Services

Arun Kumar, Sumit Mittal and Biplav Srivastava
IBM India Research Laboratory
Block 1, IIT Campus, Hauz Khas
New Delhi 110016, India
Email: {kkarun,sumittal,sbiplav}@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 web 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 in industrial setting, we bring out issues that have an impact on information modeling aspects of the composition process. We present pragmatic solutions for problems relating to scalability and manageability of service descriptions, data flow construction for operationalizing the composed services and representation of non-functional requirements.

1 Introduction

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 [7], and reasoning about their composition using goal-oriented inferencing techniques from planning. Web services pose challenge to existing planning methods in representation of complex actions, handling of richly typed messages, dynamic object creation and specification of multi-partner interactions. Moreover, web services composition (WSC) can not be seen as a one-shot plan synthesis problem defined with explicit goals but rather as a continual process of manipulating complex workflows, which requires to solve synthesis, execution, optimization, and maintenance problems as goals get incrementally refined [11]. This is because WSC 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 [10]. Due to this, the web service modeling efforts in the semantic web world fall short of the expectations of real world applications.

We have developed a prototype of end-to-end web service composition tool for facilitating new service creation and application integration for telecom service providers[1]. Mobile telephony service providers, due to intense competition, 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.

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. For the AI component, we represent services using OWL-S markup language [7] that is being defined for facilitating the creation of web service ontologies. To create deployable and executable instances of those services, we use Business Process Execution Language (BPEL) [2]. The solution drives the composition process right from specification of the business process in OWL-S, through creation of desired functionality using planning techniques, through generation of a deployable workflow by selection and binding of appropriate service instances, to finally deploying and running the composite service. This integrated solution achieves the best of both worlds and provides scalability to the composition process.

However, we ran into important modeling issues not currently addressed by the semantic web services community. The existing OWL-S service modeling 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. Moreover, how to work with non-functional requirements is unclear. A few alternative formalisms have been proposed to address OWL-S deficiencies but they focus more on foundational frameworks to overcome representational weaknesses [6, 12] rather than address ways for efficient, automatic, end-to-end composition.

We investigate information modeling issues for end-to-end composition of web services and provide pragmatic extensions to OWL-S. Our contributions

[1]Reference omitted to facilitate blind review process.

are as follows:

- We differentiate between web service types and instances. This helps in organizing the expected thousands of web services and scaling the OWL-S ontology for inferencing.
- We introduce support for context to disambiguate intended meanings of input and output parameters. In other words we add semantics to input and output parameters that helps in determining the data flow after the control flow has been worked out by planning.
- We discuss representation of functional and non-functional requirements and capabilities attached to web services. We also characterize the role of each of these specifications at each stage of the composition.
- We discuss usage of the proposed information model in an end to end web service composition tool.

The rest of the paper is organized as follows. We describe an example scenario and highlight the three problems that surface while using OWL-S for end-to-end composition - scaling of services ontology, data flow generation and specification of service capabilities. We then propose pragmatic solutions for them and apply it in the example, and give concluding comments.

2 End-to-end Service Composition

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, say, *FreshFlowerShop service, FragrantFlowerShop service, CheapFlowerShop 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 some of these services.

The figure also shows a feasible plan for the new service obtainable with an AI planner. It consists of invocations to directory service for obtaining addresses of the sender and the recipient. This is followed by invocation to a flower shop service for obtaining desired flowers. An order receipt is sent to sender. The pricing details are passed to the credit card payment gateway and on successful authorization, the shipping details and the flower packet are passed onto dispatch service for delivery. A delivery receipt is then sent to the sender. The modeling problems we faced are presented below.

Service Types Vs Instances: The scenario described above has multiple flower shop services. These services could be offering different kinds of flower packages (e.g. bouquets, decoration packages, etc.) but essentially they are

New Service Requirement

Name: FlowerDeliveryService
Input: PersonName, PersonName, FlowerName, NumOfFlowers, CreditCard
Output: OrderReceipt, DeliveryReceipt
Precon: PersonName **notNull**, 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: 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: CheapFlowerShop Service
Input: Address, Address, ItemCode, NumOfItems
Output: Acknowledgment, Packet, Charges
Precon: Address **available**, Address **available**,
ItemCode **oneOf** ITEMLIST, NumOfItems <= 100
Effect: Acknowledgment **sentTo** Address, Amount **available**, Packet **available**

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

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

Figure 1: *The FlowerDelivery Service composition scenario*

all flower shops. This fact could be very useful for efficient representation of these services. Unfortunately, OWL-S does not capture the notion of *types* and *instances*. Each OWL-S description pertains to a single instance of a service. It consists of the *ServiceProfile* that describes the interface of the service, a *ServiceModel*, that describes the details of its operation and the *ServiceGrounding* that provides information about interoperation with that service using messages.

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 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 those services. For instance, each of the FlowerShop kind of service may have different profiles even though their underlying process model may be 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, they appear to be different kinds of services.

The third drawback relates to the service grounding part. Since the grounding is specific to each service instance, a composition taking that into account is less likely to be stable. This is because changes at the level of individual service instance operation take place much more frequently than at the level of service functionality. The composition becomes prone to small implementation changes made to the service instance and this is highly undesirable. For instance, 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). One of the main differences between knowledge engineering and programming, as described in [9][2], is that while logic sentences in the former tend to be self-contained, the statements in a program depend heavily on surrounding context. Planning techniques can be used to easily generate the control flow for the composite service given the precondition and effect information for available service types, but generating the complete data flow needs reasoning with contexts of inputs and outputs.

[2]Chapter 8, Page 222

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[9] and that is the reason very specific terms are not recommended in an ontology.

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

- CreditCard information from the user goes directly to the VisaCard Service.
- the Address output from the second DirectoryService instance goes to both FreshFlowerShop Service as well as DispatchService
- DeliveryReceipt from DispatchService and OrderReceipt from FreshFlowerShop Service together constitute the output for the user.

Non-functional Service Requirements & Capabilities: The end-user requirements for the composite service, like that of any software program, can consist of functional as well as non-functional requirements (FRs and NFRs, respectively). The first part deals with the desired functionality of the composite service while the second part relates to performance, reliability and other user-acceptance issues. In terms of representation, the NFRs can be expressed as qualitative or quantitative properties. In terms of usage, the NFRs are expressed across the composite service (e.g., flower delivery time should be less than 24 hours). Like requirements, the capability of existing web services can also be characterized along functional and non-functional features.

Currently, OWL-S supports specification of FRs through IOPE and NFRs through profile attributes. NFRs are soft constraints and they distinguish individual instances but do not reflect on the nature of their functionality. The challenge is to model and reason with them efficiently by characterizing their role in the composition process.

3 Scaling Services Ontology

As mentioned in Section 2, currently OWL-S is designed to model a single web service instance [7]. However, in order to work with large collections of web services – categorizing them, supporting multiple views [5], standardization and for stable functional compositions – we need to support web service *types* that are described independent of individual web service *instances*.

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. 2). 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 a *Service* could then point to the *ServiceProfileType* of *ServiceType* 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 *ServiceType* 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*.

This approach has various modeling benefits. A new kind of service can be easily introduced by adding to the ontology an object of type *ServiceType*. This 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 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 *ServiceProfileType* of the corresponding *ServiceType*. This approach of separating type definitions from instance definitions has been used successfully in data models for distributed systems management [1].

Now, service composition can happen in two phases:

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 requirements, that would then be bound together for deploying the newly created composite service.

Discussion: 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

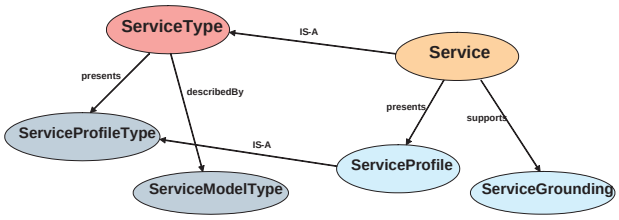

Figure 2: *Modified OWL-S* upper ontology

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 the strongest precondition from that of its instances and its effect is the weakest effect from those of its instances. Formally, $\forall S^{instance}$ of S^{type}, $S^{type}_{precondition} \supseteq S^{instance}_{precondition}$ and $S^{type}_{effect} \subseteq S^{instance}_{effect}$. We adopt it as the guideline for domain modeling. In Figure 3, FlowerShopService type captures the category of flower shop services whose instances are Fresh, Fragrant and Cheap Flower shop services. Now IOPEs of the types and instances can be related, e.g., the *ItemCode* in the last instance corresponds to *FlowerName* in the service type.

4 Generation of Data Flow

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. For instance, in Figure 1, if both PersonName and FlowerName were concepts derived from Name then it would be difficult to figure out whether FlowerName is a valid input to the Directory Service. The problem is to encode the fact that the address obtained from a single invocation of Directory Service should not be supplied to both the input parameters of succeeding FlowerShop Service. Furthermore, we have the problem of specifying which address value (out of the two invocations) goes to which input parameter.

We seek to solve the above problems by explicitly encoding the context for inputs and outputs. In [3], the authors give an extensive coverage of how context is handled in knowledge representation in AI. They note that the contextual need in OWL-S is to qualify information at a large scale but the solutions in AI focus on generalizations to support for nested contexts, ephemeral contexts and the ability to transcend contexts, which are not needed in the semantic web. Their

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**
Effect: OrderReceipt **sentTo** Address *(From)*, Amount **available**, Packet **available**

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

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

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)

Figure 3: *The proposed modeling for FlowerDelivery scenario with service types, roles and NFRs.*

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.

To associate semantics with input/output (IO) parameters, we propose the notion of *roles*. A role is a term that *qualifies* a concept. That is, for any concept φ, (ψ φ) specifies that the role played by φ is ψ. Roles are optionally specified on the inputs and outputs by the service developer. They come from a separate ontology, and are standardized and structured in a domain similar to concepts. Figure 4 shows a sample role ontology for roles that could be played during information processing, item transfer and expertise lookup. Depending on need, an input/output parameter can have either one, multiple or no specific roles. In comparison to the roles, the context of [3] means that if *ist*(c_i, φ), the proposition φ is true in context c_i. The two usages can be combined - for example, *ist*(c_i, ψ φ) means that the proposition φ has the role ψ in the context c_i. In Figure 3, the user assigns the roles of *From* and *To* to the two input addresses in the input specification.

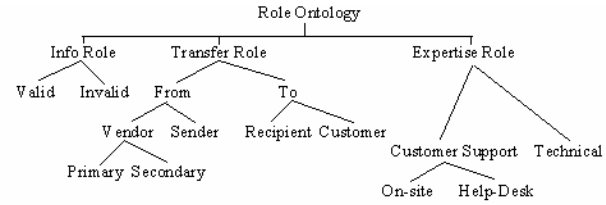

Figure 4: *Sample Role Ontology*

Roles can be propagated so that input or output or both IO can be associated with new roles in the presence of IO roles of requirement specification and/or other services. They can be propagated while matching a specification with a service instance or from the input to the output of a service and vice-versa. How a role can or cannot be propagated will be specified by the service modeler for a service using a rule language like SWRL. The rules cannot be generic because the service implementation may not allow propagation - a service which takes a *From* address as input and gives the address of where the flowers were delivered, cannot propagate the input role. Some rules are given in Figure 5. Rule 1 says that if the inputs (or outputs) of a service has parameters of the same type, no role is propagated. Rule 2 says that the input role of the specification can be propagated to input of an instance, Rule 3 says that they can be propagated from the input to the output of a service while Rule 4 says that it can be propagated from the output of one service instance to input of a successor.

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. Rule 2 will associate different roles to the two directory Service instance and Rule 3 will propagate them to the outputs. Now, using the roles on outputs of Directory Services, the data flow with the next service - FlowerShopService - is found automatically.

Given service S with inputs I and outputs O

1. If $((\exists I_i, I_j, i \neq j \text{ s.t. } I_i^{type} = I_j^{type}) \vee$
   $(\exists\ O_i, O_j, i \neq j \text{ s.t. } O_i^{type} = O_j^{type})$
   Then do not propagate

2. If $\psi_I^{instance} = \{\}$ and $\psi_I^{spec} \neq \{\}$
   $\psi_I^{instance} = \psi_I^{spec}$

3. If $\psi_O = \{\}$ and $\psi_I \neq \{\}$
   $\psi_O = \psi_I$

4. If $\psi_I^{next-instance} = \{\}$ and $\psi_O^{prev-instance} \neq \{\}$
   $\psi_I^{next-instance} = \psi_O^{prev-instance}$

Figure 5: *Some rules for role propagation.*

Discussion: 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 and vice-versa can be constructed. Association of roles with parameters of a web service 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, thereby incorporating both the syntax as well as semantics of the available services. Our solution is related to [4] who describes an environment for building reusable ontologies based on the concept of roles which they informally define 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. An alternative proposal to OWL-S is the SESMA[8] 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.

5 Specifying Non Functional Service Requirements and Capabilities

The functional capability (FC) of a web service expresses the core functionality through IOPEs that capture the transformation performed by the service. The non-functional capabilities (NFCs) help in characterizing the service further by capturing its optional features, such as cost, QoS. OWL-S has provision to represent NFCs through profile attributes which may contain parameters other than the functional IOPEs but it is unclear how to reason with these NFCs. In this section, we present our interpretation of NF attributes that may be used for enabling end-to-end composition.

NF requirements (NFRs) and NFCs can be formulated using the qualitative or the quantitative model. In the former, non-functional attributes are expressed using abstract, high level terms and goals. In the latter, specifications are represented using concrete, numeric values. It is also possible to categorize quantitative NFCs into coarse qualitative categories (e.g., fast, medium and slow services for response time $\prec$ 40 sec, 40-80 sec and $\succ$ 80 sec, respectively) and reason about them first at the broad level before working with specific numeric values.

Since NFCs inherently capture properties of service instances they are not needed during functional composition. In contrast, FCs form the core of the functional composition process. However, 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 the modified OWL-S upper ontology (presented in Section 3), the FCs get represented in ServiceProfileType. The ServiceProfile of an instance inherits those FCs from the ServiceProfileType and also 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. For example, in military applications it may be necessary to make all service instances secure. In such domains, it would make more sense to model NFCs such as security in the service type itself. In doing so, we essentially have converted some non-functional capabilities to functional capabilities since as far as that domain is concerned, they form a part of the core functionality. In this case, these NFCs are included in ServiceProfileTypes and act as FCs in selecting the service types during composition.

Table 1 shows the NFRs for the flower services. Note that NumOfFlowers is not modeled in FlowerShopService type in Figure 3. The NumOfFlowers requirement was for $\prec 1000$ and this would be ensured while picking instances.

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

Table 1: *Representing Instances for FlowerShop Services*

Discussion: The end-to-end NFRs can be handled in physical composition module by a two-fold approach - a) breaking the composite non-functional requirements into individual component requirements, and b)choosing the best available component service instance corresponding to the requirements. Some initial work on this is in [13]. Rules can be used to specify priority among NFRs to resolve conflicts during instance selection.

6 Conclusion

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 pertaining to scaling of service ontologies, lack of support for generation of data flow information and modeling of non-functional requirements. We discussed the need for separating service type from service instances, introduced the notion of roles to help disambiguate the semantics of IO parameters and discussed representation and usage of non-functional requirements. We showed that the proposed guidelines are helpful in an end-to-end composition scenario.

References

[1] Common Information Model (CIM) Metrics Model, V 2.7. www.dmtf.org/standards/documents/CIM/ DSP0141.pdf, June 2003.

[2] F. Curbera et al. Business Proc. Exec. Lang. for Web Serv. www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2002.

[3] R. Guha, R. McCool, and R. Fikes. Contexts for the semantic web. In *Proc. ISWC*, 2004.

[4] K. Kozaki et al. Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of "Role" and "Relationship". In *13th Int. Conf. on Know. Engg. and Know. Mgmt.*, 2002.

[5] R. Lara et al. Semantic Web Services: Description Requirements and Current Technologies. In *Intl. Work. on Elec. Commerce, Agents and Sem. Web Serv.*, September 2003.

[6] P. Mika, D. Oberle, A. Gangemi, and M. Sabou. Foundations for service ontologies: Aligning owl-s to dolce. In *Proc. WWW*, 2004.

[7] OWL-S. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/ owl-s/1.0/owl-s.html, Nov. 2003.

[8] J. Peer. Semantic service markup with SESMA - language spec., v0.7. In *elektra.mcm.unisg.ch/pbwsc/docs/sesma_0.7.pdf*, 2004.

[9] S. Russell and P. Norvig. Artificial intelligence: A modern approach (first ed.). In *Prentice Hall Publ., ISBN: 0131038052.*, 1995.

[10] M. Sabou, D. Richards, and S. van Splunter. An experience report on using DAML-S. In *Proc. of 12th WWW Conf.*, May 2003.

[11] B. Srivastava and J. Koehler. Planning with Workflows - An Emerging Paradigm for Web Service Composition. ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services, 2004.

[12] WSMO. Web services modeling ontology. In *http://www.wsmo.org*, 2004.

[13] T. Yu and K. Lin. Service selection algorithms for web services with end-to-end qos constraints. In *IEEE Int. Conf. on E-Commerce Technology*, 2004.