A Description Language for QoS Properties and a Framework for Service Composition Using QoS Properties

Chiaen Lin, Krishna Kavi, Sagarika Adepu
Department of Computer Science and Engineering
Unitersity of North Texas
Denton, TX 76203 USA
chiaen@unt.edu, Krishna.Kavi@unt.edu, sagarika121@gmail.com

***Abstract*—Web Services Description Language (WSDL) is an XML-based language for describing Web services and how to access them. There are established standards and frameworks for specifying and composing Web services based on the functional properties. A WSDL extension to specify non-functional or Quality of Service (QoS) properties is proposed in this paper. This enables the QoS-aware Web service composition. This paper introduces a framework that adapts publicly available tools for Web services, augmented by ontology management tools, along with tools for performance modeling to exemplify how the non-functional properties such as response time, throughput, and utilization of services can be addressed in the service acquisition and composition process. The framework provides support to achieve specified QoS goals by discovering services based on both functional and non-functional properties, and composing selected services such that the composed system satisfies the overall QoS requirements. The framework can be easily extended to automate the composition of services and update both functional and non-function properties of the combined services.**

***Keywords*-WSDL; Ontologies; Quality of Services; Non-functional Properties; Service Composition.**

I. INTRODUCTION

Service oriented architecture (SOA) offers a flexible methodology for the creation and management of software services. Software services are well-defined business functionalities situated in loosely-coupled and distributed computing settings such as Cloud and Web. Each service provides a specific and well defined functionality. Well defined interfaces permit for the discovery and invocation of services. Web service is a realization of the SOA concept. Available standards allow for the creation, registration, discovery and invocation of Web services. Web Services Description Language (WSDL) can be used to specify the functionality of a service along with its communication protocols. Service providers can register services with Universal Description Directory and Integration (UDDI) or other such registry services. Service repository can be queried by customers to discover needed services. The discovery of a service is based on searching through categories and by matching the specification given in WSDL.

The goal of our project is to discover services based not only on their functionality but also based on non-functional (or quality of service) properties. In addition, our goal includes service composition and specification of non-functional properties of composed services. These goals require the ability to specify non-functional (or QoS) properties with services, and the ability to compute non-functional measures of composed services. Ascertaining certain non-functional properties of composed service require models and tools that are appropriate for the specific property (e.g., stochastic models for performance measures). In this paper, we explore the development of the necessary framework for composing performance properties using queuing models.

WSDL can only be used for specifying functionality of services. Non-functional properties, including several quality of service (QoS) characteristics, are crucial to the success and wider adoption of Web services. Customers would like to use QoS characteristics of Web services for selecting from among several alternate implementations. Each of the potential service provider declares similar functionalities for the same purpose – thus the customer expects more information about services. Typical among QoS properties are security, reliability, and performance [1]. WSDL should be extended in order to provide QoS related information with services. Once non-functional properties of services are specified, it will be possible to develop or extend tools for the discovery of Web services based both on functionality and non-functional properties. Additional tools can be designed for service aggregation, integration and composition based on QoS characteristics.

As we proceed with the quality-aware extension to the specification of services, it will be necessary to define standard metrics for non-functional properties. The Cloud Council that is developing a practical guide to Service level Agreements [2], recommends using ISO definitions [3] for standard metrics. Service composition leading to the computation of QoS properties of the composed services present new challenges. Consider for example "response time" as a non-functional property, and consider the composition of two services with 3ms and 5ms response times. One cannot assume that the response time of the composed service is 8ms, since computation of service times are based on stochastic measures and it may become necessary to use

 ISBN: 978-1-61208-230-1

appropriate models (e.g., queuing theory) for computing the response time of the composed service. The orchestration of services in a composed service plays an important role in modeling QoS properties of the composite service. In the case of performance, additional complexity results from the current workload at a processing node: a lightly loaded node leads to faster response times. This may necessitate specification of performance properties at different levels of workloads (e.g., at low, average and heavy loads). These complexities can be managed using ontologies for the specification of non-functional properties.

Since our motivation is not only the discovery of services meeting QoS requirements, but also to compose services leading to new services and ascertaining the QoS properties of composed services, we felt that available QoS extensions do not fully meet our needs. Hence we propose our own extensions to WSDL to specify QoS properties. To exemplify the utilization of these extensions, we propose a framework for service composition with the assistance of both ontological and performance modeling tools. QoS properties are modeled with the ontological engine that can be expanded in accordance with the service declaration. Properties that are subject to a chosen performance tool can also be noted in the ontology model for further semantic comparisons. In this paper, we will focus on the performance aspect of the service; in particular service response times, utilization, and throughput. As a proof of concept, we demonstrate the process of WSDL extension, along with its corresponding QoS ontology modeling, performance modeling, and service composition using an example.

The key contributions of our work are (a) WSDL extensions for specifying nonfunctional properties (b) ontology for classification of non-functional properties (c) framework for the discovery of services that meet both functional and non-functional requirements (d) a framework for computing performance (stochastic) measures of composed services.

The layout of the paper is as follows. Section 2 overviews research that is closely related to our work. Section 3 describes how we extended WSDL to include QoS properties. Section 4 introduces the creation of QoS ontology and performance modeling to be used in the Web services. Section 5 uses a case study to demonstrate QoS based Web service composition framework. Section 6 includes our conclusions about the study.

II. Related Works

The description of non-functional properties related to SOA operational management has been described in [4]. In addition to adding some QoS criteria, semantic interpretation to the extensions have been realized in various frameworks [5] [6] [7]. An approach to describing service lifecycle information and QoS guarantees offered by a service based on OWL-S can be found in [8]. Here, service profiles are appended with QoSCharacteristics to generate a corresponding service description repository. The OWL-S based repository can automatically cover the traditional UDDI registry by mapping its elements. In [9], WSDL is extended to X-WSDL where non-functional criteria are added in service definition. Following its predecessor X-UDDI [10], the Web service registration and publication can be queried on the basis of this criteria. In [11], a unified semantic Web services publication and discovery framework is proposed with a QoSMetrics extension to WSDL using PS-WSDL, USQL for service query, and UDDI mapping suites. In this paper, we focus on a proof of concept for WSDL extension and its correspondent non-functional semantic model engineering, but not on the service registration. With our framework, it should be straightforward to apply well-defined UDDI extension tools such as mentioned in [12], or other registry tools.

To enable semantic description of service extensions, several ontological languages have been proposed. An overview of some of these languages can be found in [13]. They focus on the semantic modeling and mapping ontology applied to service descriptions. Our framework focusses on the engineering of ontology model and its references to the performance modeling tools. With the help of ontology mapping, different service description and advertisement standards should be easy to adapt in our framework.

Service composition methods and their languages can be broadly categorized into different types: Orchestration, Choreography, Coordination, and Assembly [14]. While emphasizing from different aspects to approach the issue, composition methods use ontology to annotate QoS attributes that provide common ground for service synthesis, execution, and adaptation [15]. In QoS-aware service composition, services are selected based on inter and intra task constraints. They can also be grouped into deterministic and non-deterministic depending on when these attributes were made known [14]. Various researches are hoping to gain optimal results by using detailed descriptions of QoS values of services during composition [16] [17]. In [18], a quality-driven middleware serves as a composition manager that model multidimensional QoS attributes with utility functions, and optimizes them by local selection and global planning for different quality criteria.

In [19], requested and provided QoS properties are expressed as required specification documents and service specification documents respectively in the open dynamic execution environment. The framework serves as a broker for service compositions that utilizes QoS model in its own ontological language. Service selection algorithms and metrics based on the ontology are utilized by the service broker. Its objective is to support ad-hoc service collaborations, while ours is to facilitate the description of QoS properties of existing and new composite services. The work is similar to ours with the emphasis on using ontology model as the tool to reason QoS attributes semantically. When monitoring

 ISBN: 978-1-61208-230-1

the execution condition, the ontology model can facilitate the selection of correct set of QoS values according to the execution environment. The QoS-aware ontology modeling framework we propose can serve the same purpose.

The advantage of our framework is in its facilitation of stochastic performance evaluation during service composition. The above mentioned related works do not consider the use of ontology and performance models working closely to address the evaluation of QoS properties of service composition. In addition, we consider the use of different performance tools with the model-related elements in the ontology, facilitating the usage of QoS attributes based on context and selecting appropriate models and tools for ascertaining properties during composition.

III. QoS-Aware WSDL

WSDL is the standard language suggested by World Wide Web Consortium (W3C) for service specification. It can be read as a conceptual model consisting of components with attached properties, which collectively describe the service [20]. A WSDL specification contains abstract and concrete descriptions of the service. At abstract level, it describes the interface to the service: operations with message exchange patterns (MEP) and parameter types. At the concrete level, a binding specifies the transport type that the interface uses. An endpoint then associates a real network address with the binding, which forms the service. The service is invoked by supplying the declared signature to the interface through its endpoints.

Although the syntactic specifications provide information about the structure of input and output messages, and the functional descriptions of the service, WSDL does not address non-functional properties. To fully utilize Web services, non-functional information, along with functionality, is needed in the service description. To augment any proposed extensions, backward compatibility and its extension level must be considered. Since WSDL description model addresses abstract and concrete components with services, the non-functional extensions to WSDL should be considered accordingly. It should be compatible with the original Web services mechanism in that the addition may be considered optional. Web service engines and operations should be able to freely ignore the QoS information as they choose to operate in the conventional environment. For applications that adapt our framework, the QoS-aware extensions are extracted easily. We decided that the extensions should be established at service level rather than at interface level, since the WSDL interfaces are bounded by the message exchange patterns and considered abstract models. At service level, an endpoint is where the abstract service binds to a concrete port type, where the overall service performance can be noted.

WSDL2.0 Core standard provides element-based extensibility that can be used to specify technology-specific

```
<xs:complexType name="criteriaService">
  <xs:complexContent>
    <xs:extension base="qwsdl:ExtensibleDocumented">
      <xs:sequence>
        <xs:elementname="performance" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="performanceType">
  <sequence>
    <element name="ResponseTime" type="qwsdl:ResponseTimeType" minOccurs="1"
maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Offered" type="boolean"/>
</xs:complexType>
<xs:complexType name="ResponseTimeType">
  <attribute name="value" type="float"/>
  <attribute name="unit" type="string" fixed="sec"/>
  <attribute name="category" type="qwsdl:CategoryType"/>
</xs:complexType>
```

Figure 1. QoS-Aware WSDL schema for Performance parameters

binding. We create an element in WSDL to represent QoS property specification. Then we use the element as the extension element to the endpoint. The service with the endpoint is therefore being annotated by the extended properties. For a QoS property extension element, we use complex type in the XML schema to accommodate the data structure of the QoS. As depicted in Figure 1, the QoS-aware extension schema exemplifies a non-functional property of performance. Within the performance criteria, response time is noted with its value, unit, and category. The extension can also be further referenced by importing latest XML schema version which can be updated on-the-fly as revising the QoS ontology model, thus conforming to the latest XML standards.

IV. Performance Service Composition

Service composition decisions have to be made from considerations of both functional and non-functional requirements. To manage the semantics of both aspects and facilitate the automatic selection of service components that meet the service level requirement, an ontology engine is proposed to efficiently and flexibly classify both functional and non-functional attributes. Services and their components can be further classified according to the application domain, using the category scheme, to facilitate the retrieval and management of corresponding services. The availability of desired service depends on the discovery resulting from querying the ontology model. In case of no services meet the requirements, existing services can be acquired and composed. The newly composed services can then be added to the ontology engine for future selections.

In some cases, computing the non-functional properties of composed services requires stochastic models. Consider performance properties of services such as response time. These performance attributes are used to filter and rank services so that service selections can be made. As new services are composed from its constituent service components, performance indexes can be generated by modeling the new

 ISBN: 978-1-61208-230-1

composition through the stochastic model. The performance evaluation results along with the new composed service are added back to the ontological model for future reference.

The backend of the composition framework provides interfaces in utilizing ontology model and models for the evaluation of QoS properties for Web service composition. The two modules are independent and any potentially compatible models and tools can be plugged in. In this paper, we illustrate the process of creating the ontological model, and use a queuing model for composition of performance related properties of services.

A. Ontological Property Model

Ontologies offer more accurate and flexible cataloging of entities than taxonomies. While the latter uses hierarchical and branched static structures to group entities and manage information using structural organizations, ontological model annotates semantics with meta-data, relating properties and attributes with more complex organizations than branching or tree like structures. Ontology model therefore provides more flexible organization and semantic interpretation of data with entities.

The quality of service property of a Web service can be inferred by its performance attributes. Criterion of service selection can be formulated by the configuration of these attributes to indicate levels of service importance. Due to the dynamic nature provided by service oriented mechanism, even the meaning of performance metrics should be adapted to fit the context of the service domain. For example, a Web service component qualified for soft real-time application may be considered if they have reasonable response times; however they may not be suitable for hard real-time environments, unless the response times can be bounded. Different contexts impose different semantic interpretations on the same non-functional properties. However the ontology model is highly flexible and thus multiple semantic interpretations can be associated with properties associated with services.

To create an ontological model for Web services, leading to service composition, we demonstrate the process of establishing performance as non-functional property of Web services, such as response time, server utilization, and throughput. Further context related performance indicators can also be easily added with similar considerations. In order to create, update, and query the performance properties during the Web service composition process, we need to establish records of each and every services. We adapt the Protege Editor [21] as the editing tool to help create an ontology model. Protege Editor has a GUI interface [22]. Users can specifically define Entity and Class as first-class elements in the schema along with their Object Properties. Instance of object can be initialized as an Individual and its Data Property can be appended. Visual tools are provided by the editor's plug-ins to facilitate various aspects views. There are also several reasoners available that can be invoked to check and infer the ontological derivations automatically.

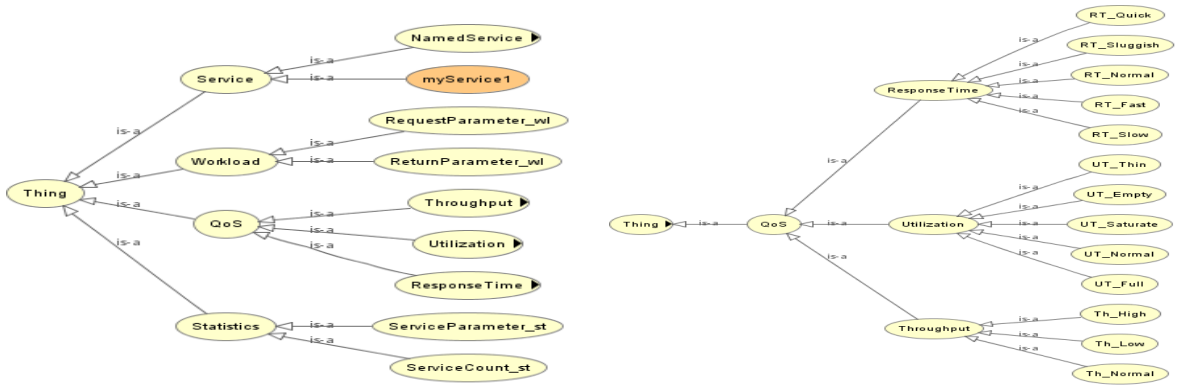
We differentiate the first-class elements from base performance and model relevant ones. The model refers to the performance modeling used in the compostion. The base ones serve as the mandatory performance attributes that all the Web services are required to specify as performance indicators. The model-related properties serve as the supplement to the application-specific modeling approaches, thus can store additional attributes for use by specific methods and tools. In our example, the base performance classification is represented by Quality-of-Services (QoS). The QoS subclasses ResponseTime, Throughput, and Utilization are base performance indicators, to quantify performance property. Model-related attributes include Workload and Statistics. Workload here is used as an attribute to evaluate the significance of base QoS properties. The attributes allow for recording the criteria under which the performance properties were derived and thus allow for adjustments when new running environments differ from these values.

For each of the base and model-related first-class elements, classification can also be refined into detailed subclasses. For instance, a response time can be ranked into subcategory such as Fast, Quick, Normal, Slow and Sluggish. Each of the rank can also be noted with its values that represents the class, based on specific context. As the new service composition emerges, the new service can easily be accommodated in the ontological model, establish quantity and corresponding semantics, and is ready for further queries and reasoning. The example model described here includes base and model-related classes and depicted in Figure 2(a). The refined QoS rank subclasses example is depicted in Figure 2(b).

To be able to interface with Protege Editor so that we can update and query the ontology model for service composition processing, we further convert the ontological process into programming. Protege-OWL provides the capability to convert API to equivalent GUI functions and the mechanism for plugging reasoners [23]. We follow the process steps of creating the ontology model, and make the process programmable. The base QoS schema can serve as the building block for the extension of the ontology model. The automation enhances the flexibility to experiment on the first-class cataloging and their refined properties. It also provides a convenient facility to plugin a specific model-related ontology for performance modeling.

The automation process can be further extended with a reasoner to the ontology model that enhances the reasoning ability while interacting with the model. A reasoner implementing the reasoner plugin programming interface will be accessible in the same way that the built-in reasoners are. We choose the Jena Framework from Apache as the reasoner mechanism for our running example. It is well-known and an open source tool. Its query and storage architecture can

 ISBN: 978-1-61208-230-1



(a) Performance Ontology Model

(b) QoS Ontology Model with Refined Ranks

Figure 2. Ontology Models

enable more flexible online usage of the ontology engine.

We adapt Jena programming API [24] to read the ontology model built from Protege-OWL, and validate the model by the reasoner rules. For inference support, Jena provides a general purpose rule engine that the ontology model can be validated and the application specific rule can be applied to facilitate aspects of Web service composition management. The service composition can be fine-tuned by using the rules from domain experts or engineers that impose application related restrictions. For instance, assume the response time of a quick Web service is defined to be less then five milliseconds, the selection of the candidate Web service can be filtered by the rule :

[print-a-quick-WebService: (?x pre-ws:hasQoS ?a) (?a pre-ws:hasResponseTime ?b) (?b pre-ws:rt_value ?c) lessThan(?c, 5.0) → print(?x, 'has quick QoS:',?c)]

The print-a-quick-WebService rule prints out any service entity that has a QoS property with a ResponseTime value smaller than 5.0 msec. Similarly, other plug-in rules can be used to customize the model to meet the needs of an application, such as performance rank selection in a service category.It should be noted that it is possible to define a reasoner that uses context related information to define fast, slow response times subjectively, instead of using values.

The ontology model automation enables the Web service composition to be processed online. New classes and properties can be created on-the-fly to address the specific needs of applications. Web service processes can also benefit from adaptation to different service domains by interpreting the performance parameters. The online feedback from the analysis is the up-to-date data that enhances accuracy of the reasoning.

B. Performance Modeling for Service Composition

Different methodologies for evaluating performance of software services such as process algebra, queueing networks, and Petri nets come with different analysis tools for example, PEPA [25], LQN [26], and SPNP [27]. Stochastic performance models have been widely used in the performance evaluation community. In the Web services community, it also plays an important role in assuring that the service performance meets service level agreements.

The purpose of our framework is to provide a platform that enables the use of appropriate tools for performance evaluation in Web service composition. According to the approaches the process takes, developers can explore different tools fit the nature of the composition. Appropriateness can also be explored by comparing various tools for their usability. To demonstrate the usability of the framework, we explain the use of a queueing model with services containing mandatory performance attributes.

While composing services, the flow among the component services can be described using a workflow or business logic. Each of the services can be represented as service nodes, and the request flow can be modeled as waiting queues. In front of each service node, requests are waiting in line for the service to process them in order. The composition model is formed with the integration of the coordinated services network. The performance outcome of the queueing network is the performance result of the newly composed service. The mapping is seen as a close fit to both the performance evaluation mechanism and the Web service composition concept. In the example of our case study, layered queueing model [26] is adapted as the tool to demonstrate our framework. We will use an example to illustrate the framework (see Section 5).

Layered queueing model is a conventional queueing model embedded with the architecture of a software system and needed resources [26]. The first class elements are processor, task, entity, and activity. A task represents a resource that has processors and other entities to execute. Each of the entities in turn can invoke other entities on other tasks to fulfill the job. These invocations are modeled in layered fashion, and can be depicted as a directed graph. For further detailed modeling, each entity can be represented

 ISBN: 978-1-61208-230-1

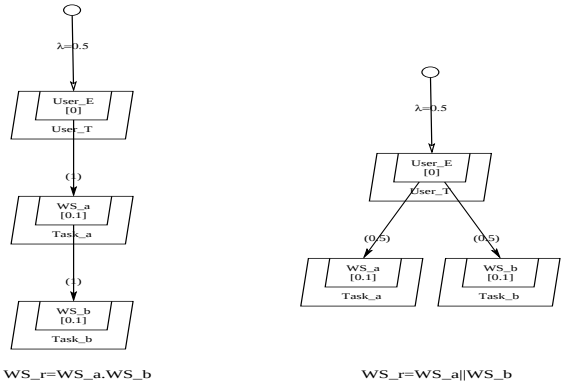


Figure 3. Web Services Layered Queueing Network Modeling

with more specific activities in its own data flow. For each task and activity, there will be resource requirements specified as service time that denotes a performance attribute. And, the mean number of calls represents the average of invocations from one entity to another. With the information for each task and their entities noted, a queuing network can be constructed to represent the integration of all the tasks, leading to workload model using either open or closed queuing models. The former can be modeled with mean arrival and service rates, while the latter can specified using mean value analysis (MVA). As soon as the model is developed, the layered queueing solver can generate reports on the performance indexes such as service time, throughput and utilization for both services and processors. It also generates average waiting times in open queuing model and mean delay in closed models.

The simplest form of a Web service composition involves two services, say WS_a and WS_b. The possible compositions of the two services can be sequential or parallel composition, say WS_rs and WS_rp. Borrowing the syntax from generic process algebra, the sequential composition can be represented as WS_rs=WS_a.WS_b, and the parallel composition can be represented as WS_rp=WS_a||WS_b. Assume WS_a and WS_b each represents an entity in different tasks say Task_a and Task_b. Each task is assigned to run on its own processor on different hosts say Proc_a and Proc_b. The sequential and parallel composition examples with an open arrival rate 0.5 are depicted in Figure 3. Note that the arrival rate is categorized as workload in the ontology model, and the example just serves as an instance. In the case of similar services encountered same workload but running on different platforms, the selection process have to compare the performance indices such as response time or throughput.

The service composition in both sequential and parallel topology can be scaled by accommodating multiple services at once. Resources can be exclusively owned or shared among services. Service composition can be based on either serial or parallel composition of the services involved. The final layout of the queueing network is the conceptual modeling of the Web service composition. The model can be solved by the analyzer and generate the performance indexes for the composed service.

C. Compositional Semantic Web Services

To export the ontological result that is acquired by the Web service composition mechanism, we use Axis [28] as the service publishing interface for demonstration purposes. The interface also enables the abstraction that Web service ontological engine (WSOE) and performance modeling engine provide.

The WSOE provides for composition of services in the context of Web services management. The utility of the composition services include basic service information maintenance and composition. Service management functions include insertion, update, and deletion. WSOE_Insertion creates a record in the performance ontology model with its name and associated performance properties. The performance properties in our running example is the response time of the service. Other non-functional or QoS properties can also be included within our framework. WSOE_Update and WSOE_deletion are used to update and remove the correspondent services.

New service composition information created by the WSOE can be obtained by the WSOE_Compose_Seq or WSOE_Compose_Par. The former will take the list of Web services in the order specified, and model them as a sequential network in the layered queueing model. The output will be the performance indexes for the composed service. For our simple example, the composition would return the predicted execution time. Likewise, WSOE_Compose_Par will take a list of Web service in the argument, and model them as parallel network in the queueing model. The sequential and parallel compositions can be combined to obtain any general compositions of services. The list of the service interfaces are listed in Table I.

V. CASE STUDY

To demonstrate the web services composition framework, we will use a facial recognition service as an example. We chose this example because of our familiarity with it while working on a related project on service composition. The service collects image data from an attached camera and identifies the presence of human faces. The service consists of Facial Detection (FD), Image Converter (IC), and Facial Recognizer (FR), in that order. First each of the component services are described using our QoS-aware WSDL to denote both functional and non-functaional properties. For each of these services we keep their QoS records in the ontology repository. We will assume that the services are all registered so that search engine can match potential candidates

 ISBN: 978-1-61208-230-1

Table I
WEB SERVICE INTERFACES OF WSOE

Service Name	Parameters	Result
WSOE_Insert	Service_Name, Response_Time	Boolean (True/False)
WSOE_Update	Service_Name	Boolean (True/False)
WSOE_delete	Service_Name	Boolean (True/False)
WSOE_Compose_Seq	$SN_1 ... SN_n$	Service_Name, Response_Time
WSOE_Compose_Par	$SN_1 ... SN_n$	Service_Name, Response_Time

Table II
LAYERED QUEUEING MODEL FOR FACIAL RECOGNITION SERVICE COMPOSTION EXAMPLE

```
#General Section
G
"Web service modeling."
0.00001
100
1
0.9
-1

# Processor Information
P 0
p User_P f i
p FD_P f
p IC_P f
p FR_P f
-1

# Task Information
T 0
t User_T r User_E -1 User_P m
100
t FD_T n FD_E -1 FD_P
t IC_T n IC_E -1 IC_P
t FE_T n FR_E -1 FR_P
-1

#Entry Information
E 0
s User_E 0 -1
y User_E FD_E 1 -1
s FD_E 0.1 -1
y FD_E IC_E 1 -1
s IC_E 0.1 -1
y IC_E FR_E 1 -1
s FR_E 0.3 -1
-1
```

```
Service time:

Task Name Entry Name Phase 1
User_T User_E 50.4902
FD_T FD_E 0.5
IC_T IC_E 0.4
FE_T FR_E 0.3

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name Entry Name Phase 1 coeff of var **2
User_T User_E 7598.29 2.98059
FD_T FD_E 0.75 3
IC_T IC_E 0.34 2.125
FE_T FR_E 0.09 1

Throughputs and utilizations per phase:

Task Name Entry Name Throughput Phase 1 Total
User_T User_E 1.98058 100 100
FD_T FD_E 1.98058 0.990291 0.990291
IC_T IC_E 1.98058 0.792233 0.792233
FE_T FR_E 1.98058 0.594175 0.594175
```

Table III
LAYERED QUEUEING MODEL FOR FACIAL RECOGNITION SERVICE COMPOSTION EXAMPLE

```
#General Section
G
"Web service modeling."
0.00001
100
1
0.9
-1

# Processor Information
P 0
p User_P f i
p FD_P f
p IC_P f
p FR_P f
-1

# Task Information
T 0
t User_T r User_E -1 User_P m
100
t FD_T n FD_E -1 FD_P
t IC_T n IC_E1 IC_E2 -1 IC_P
t FE_T n FR_E -1 FR_P
-1

#Entry Information
E 0
s User_E 0 -1
y User_E FD_E 1 -1
s FD_E 0.1 -1
y FD_E IC_E1 1 -1
y FD_E IC_E2 1 -1
s IC_E1 0.06 -1
y IC_E1 FR_E 1 -1
s IC_E2 0.06 -1
y IC_E2 FR_E 1 -1
s FR_E 0.3 -1
-1
```

```
Service times:
Task Name Entry Name Phase 1
User_T User_E 82.342
FD_T FD_E 0.82
IC_T IC_E1 0.36
IC_E2 0.36
FE_T FR_E 0.3

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name Entry Name Phase 1 coeff of var **2
User_T User_E 20207.5 2.98037
FD_T FD_E 1.24138 1.84619
IC_T IC_E1 0.3096 2.38889
IC_E2 0.3096 2.38889
FE_T FR_E 0.09 1

Throughputs and utilizations per phase:
Task Name Entry Name Throughput Phase 1 Total

User_T User_E 1.21445 100 100
FD_T FD_E 1.21445 0.995846 0.995846
IC_T IC_E1 1.21445 0.437201 0.437201
IC_E2 1.21445 0.437201 0.437201
Total: 2.42889 0.874401 0.874401
FE_T FR_E 2.42889 0.728668 0.728668
```

meeting both functional and non-functional requirements of the customer.

To create the composed service, a list of qualified candidates of each component services are evaluated. Let us assume that our selection picked FD_E, IC_E, and FR_E as service components. We will now evaluate the non-functional values for response time of the composed service. We model the layered queueing network as follows. The service components are mapped as the entities in the layered queueing network with their correspondent tasks PD_T,IC_T, and FR_T, each of which uses processors FD_P,IC_P, and FR_P. The modeling script and the performance indexes of the example are shown in Table II.

Furthermore, let us assume that Image Converter (IC) service can be composed in parallel to improve performance. The composition engine can be configured to explore the service composition using parallel workflow among the services. The new composition would use two Image Converter (IC) services in parallel named IC_E1 and IC_E2. The modeling script of the example and the performance indexes result are shown in Table III.

Although we only used a simple example and a single property here, our framework is very general and flexible so that it can be easily extended for more complex service discovery based on many QoS properties, and can composed in very complex manner.

VI. CONCLUSION

In this paper, we described a framework for composing Web-services using both functional and QoS properties. We first extended WSDL descriptions of Web-services so that non-functional or quality of service parameters can be associated with the service. We also developed APIs for locating Web-services based on both functional and non-functional properties. We have developed ontologies that can be used to select and compose Web-services. For the purpose of composing non-functional properties of component services, new reasoning engines must be developed. Different non-functional properties may require different reasoning engines. In this paper, we outlined how performance properties can be composed using queuing engines. For the purpose of this paper we demonstrated how services can be composed either in series or in parallel, and used a queuing engine to derive the performance properties of the composed service.

We plan to extend the framework for composing Web-services using other types of QoS properties. While it is possible to use other available tools, in our study we will rely on open source tools.

VII. ACKNOWLEDGEMENTS

This research is supported in part by the Net-Centric Industry/University Cooperative Research Center (Net-Centric IUCRC) and a grant from NSF, #1128344.

 ISBN: 978-1-61208-230-1

REFERENCES

[1] S. Balasubramaniam, G. Lewis, E. Morris, S. Simanta, and D. Smith, "Challenges for assuring quality of service in a service-oriented environment," in *Principles of Engineering Service Oriented Systems, 2009. PESOS 2009. ICSE Workshop on*. IEEE, 2009, pp. 103–106.

[2] "Practical guide to cloud service level agreements," 2012, http://www.cloud-council.org/press-release/04-03-12.htm [retrieved: Oct,2012].

[3] "ISO/IEC 20926," 2009, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51717 [retrieved: Oct,2012].

[4] D. Edmond, J. O'Sullivan, and A. ter Hofstede, "What's in a service? towards accurate description of non-functional service properties," *Distributed and Parallel Databases Journal*, vol. 12, pp. 117–133, 2002.

[5] S. Chaari, Y. Badr, and F. Biennier, "Enhancing web service selection by qos-based ontology and ws-policy," in *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2008, pp. 2426–2431.

[6] H. Muñoz Frutos, I. Kotsiopoulos, L. Vaquero Gonzalez, and L. Rodero Merino, "Enhancing service selection by semantic qos," *The Semantic Web: Research and Applications*, pp. 565–577, 2009.

[7] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 60–67, 2007.

[8] C. Schröpfer, M. Schönherr, P. Offermann, and M. Ahrens, "A flexible approach to service management-related service description in soas," *Emerging Web Services Technology*, pp. 47–64, 2007.

[9] N. Parimala and A. Saini, "Web service with criteria: Extending wsdl," in *Digital Information Management (ICDIM), 2011 Sixth International Conference on*. IEEE, 2011, pp. 205–210.

[10] Parimala, N. and Saini, A., "Decision support web service," *Distributed Computing and Internet Technology*, pp. 221–231, 2011.

[11] T. Pilioura and A. Tsalgatidou, "Unified publication and discovery of semantic web services," *ACM Transactions on the Web (TWEB)*, vol. 3, no. 3, p. 11, 2009.

[12] C. Atkinson, P. Bostan, G. Deneva, and M. Schumacher, "Towards high integrity uddi systems," in *Business Information Systems Workshops*. Springer, 2009, pp. 350–361.

[13] C. Pedrinaci, M. Maleshkova, M. Zaremba, and M. Panahiazar, "Semantic web services approaches," *Handbook of Service Description*, pp. 159–183, 2012.

[14] G. Baryannis, O. Danylevych, D. Karastoyanova, K. Kritikos, P. Leitner, F. Rosenberg, and B. Wetzstein, "Service composition," *Service research challenges and solutions for the future internet*, pp. 55–84, 2010.

[15] G. Dobson and A. Sanchez-Macian, "Towards unified QoS/SLA ontologies," in *Services Computing Workshops, 2006. SCW'06. IEEE*. IEEE, 2006, pp. 169–174.

[16] G. Canfora, M. Di Penta, R. Esposito, and M. Villani, "Qos-aware replanning of composite web services," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005, pp. 121–129.

[17] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A qos-aware selection model for semantic web services," *Service-Oriented Computing–ICSOC 2006*, pp. 390–401, 2006.

[18] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311–327, 2004.

[19] A. Mukhija, A. Dingwall-Smith, and D. S. Rosenblum, "Qos-aware service composition in dino," in *Web Services, 2007. ECOWS'07. Fifth European Conference on*. IEEE, 2007, pp. 3–12.

[20] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (wsdl) version 2.0 part 1: Core language," *W3C Recommendation*, vol. 26, 2007.

[21] H. Knublauch, R. Fergerson, N. Noy, and M. Musen, "The protg owl plugin: An open development environment for semantic web applications," *The Semantic WebISWC 2004*, pp. 229–243, 2004.

[22] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using the protg-owl plugin and co-ode tools edition 1.0," *The University Of Manchester*, 2004.

[23] H. Knublauch, "Protg-owl api programmers guide," *2008-04-22].http://protege.stanford.edu/plugins/owl/api/guide.html*, 2006.

[24] A. Jena, "semantic web framework for java," *URL: http://jena.sourceforge.net*, 2007.

[25] S. Gilmore and J. Hillston, "The pepa workbench: a tool to support a process algebra-based approach to performance modelling," *Computer Performance Evaluation Modelling Techniques and Tools*, pp. 353–368, 1994.

[26] G. Franks, P. Maly, M. Woodside, D. C. Petriu, and A. Hubbard, "Layered queueing network solver and simulator user manual," *Dept.of Systems and Computer Engineering, Carleton University (December 2005)*, 2005.

[27] G. Ciardo, J. Muppala, and K. Trivedi, "Spnp: stochastic petri net package," in *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*. IEEE, 1989, pp. 142–151.

[28] A. Axis, "Apache web services project," *Available HTTP: http://ws.apache.org/axis*, 2010.

 ISBN: 978-1-61208-230-1