A QoS-Aware BPEL Framework for Service Selection and Composition Using QoS Properties

Chiaen Lin and Krishna Kavi
Department of Computer Science and Engineering
Unitersity of North Texas
Denton, TX 76203 USA
chiaen@unt.edu, kavi@cse.unt.edu

***Abstract*—The promise of service oriented computing, and the availability of web services in particular, promote delivery of services and creation of new services composed of existing services – service components are assembled to achieve integrated computational goals. Business organizations strive to utilize the services and to provide new service solutions and they will need appropriate tools to achieve these goals. As web and internet based services grow into clouds, inter-dependency of services and their complexity increases tremendously. The cloud ontology depicts service layers from a high-level, such as Application and Software, to a low-level, such as Infrastructure and Platform. Each component resides at one layer can be useful to others as a service. It hints the amount of complexity resulting from not only horizontal but also vertical integrations in building and deploying a composite service. Our framework tackles the complexity of the selection and composition issues with additional qualitative information to the service descriptions using Business Process Execution Language (BPEL). Engineers can use BPEL to explore design options, and have the QoS properties analyzed for the design. QoS properties of each service are annotated with our extension to Web Service Description Language (WSDL). In this paper, we describe our framework and illustrate its application to one QoS property, performance. We translate BPEL orchestration and choreography into appropriate queuing networks, and analyze the resulting model to obtain the performance properties of the composed service. Our framework is also designed to support utilizations of other QoS extensions of WSDL, adaptable business logic languages, and composition models for other QoS properties.**

Keywords*—*WSDL; WS-BPEL; Quality of Services; Non-functional Properties; Service Composition.

I. Introduction

Service oriented architecture (SOA) is a flexible and scalable design methodology to seamlessly integrate and cooperate services in distributed software and systems. As more services are on the web and in the cloud, it becomes easier to create new services dynamically by composing existing services, customized to meet the needs of customers [1]. Before invoking a service, a service requester has to query the functionality as well as the interaction protocols defined to access the service. Web Service Description Language (WSDL) [2] is a widely accepted standard from World Wide Web Consortium (W3C) for describing functionality of web services. The Universal Description, Discovery and Integration (UDDI) registry serves as a repository for the services with WSDL descriptions. Users can query the UDDI and find services meeting their needs since the functionality of the services can be obtained from their WSDL specifications [3].

Once the services are selected, interactions among the services are achieved using messaging protocol defined in WSDL. Even with ever increasing number of services, it may still not be possible to find the ”right” service, and in such cases, one has to either create a new service from scratch, or compose the service using existing services. Tools and frameworks are becoming available to aid in the dynamic composition of services [4], [5], [6], [7], [8]. Another issue that needs to be addressed is related to selecting the appropriate services that takes part in a composition, particularly when multiple services with the same functionality are available. In such cases, non-functional or Quality of Service (QoS) properties, such as performance, security, reliability become the delimiters [9], [10], [11], [12].

While standard WSDL describes the functionality of a service, it does not specify QoS or non-functional properties. In the previous work [13], we augmented the WSDL to permit specification of non-functional properties of a service. The additional information can help distinguish between services with the same functionality, and these properties can be used while composing new services to ascertain the QoS properties of the composed service.

Enterprise software systems or cloud computing often use business logic to refine their design and regulate the behavior of services according to business processes [14]. Business Process Execution Language (BPEL) has become the standard for describing the architecture of a service process [15]. It contains control constructs for the orchestration of component services in a workflow style. While tools and frameworks are available to use BPEL orchestrations in composing services, they are not suitable to evaluate the QoS properties of specific orchestrations [16], [17]. In this paper, we expand our framework to adapt the notion of BPEL to describe QoS-aware services for their selection and composition. We argue that based on our previous QoS-extension framework, BPEL is compatible for use of QoS extensions. The expansion is also backward compatible with the SOA in general and web services in particular. It is suitable for the incorporation of any tools that facilitate QoS extensions and models for analyzing QoS properties. We illustrate how to create queuing models for various BPEL orchestration logic compositions. In the

case study, we also demonstrate our framework for composing performance properties using stochastic models, such as the Layered Queueing Network (LQN) model.

The main contribution of our work is the framework that permits the description of QoS properties with services (using our QoS extensions to WSDL, or other suitable QoS extensions), reasonable composition processes of these services, and the generation of models for deriving QoS properties of composed services that use BPEL. While we focus on stochastic models, our framework can also be used to incorporate models for reliability, availability, as well as security, provided tools for deriving these properties for composed services from properties of component services are available.

The rest of the paper is organized as follows. Section 2 describes our QoS-extension framework and its suitability for use with BPEL. Section 3 gives details of BPEL service composition approaches. Section 4 presents the rules for translating BPEL logic constructs into queuing network models for use by the Layered Queuing Network (LQN). Section 5 illustrates our framework with a case study. Section 6 describes research that is closely related to ours. And, Section 7 contains conclusions about our work and what we propose to do in the near future.

II. QoS-aware Framework in SOA

In this section, we present the QoS-aware framework in the SOA environment which is an extension to our previous work [13]. We first describe essential components of the framework, and its operational processes. Then, we consider the quality awareness extension to the business processes, specifically using WS-BPEL [18] to enable the performance evaluation result in service design, selection, and composition which include operational logic choices.

A. Framework Description

In our previous work, we extended WSDL to permit specification of non-functional property elements with services. Each web service can optionally describe QoS properties along with functional properties, in order to distinguish itself from other services providing similar functionality. QoS properties can include performance, reliability, security or other quality metrics. The framework is compatible with traditional SOA for either standard or quality-aware service publication, selection and interaction. The infrastructure of the framework augments the SOA with three elements:

- QoS-Aware WSDL Extension (QoS_WSDL): These are new WSDL elements for specifying QoS properties with services.
- Ontological QoS Modeling (QoS_Ontology): QoS properties and categories are classified by our ontology model allowing different classes of QoS properties and relationships among these categories.
- Testing and Composition QoS Modeling (QoS_TestCompose): QoS properties can be used for selecting services and evaluating QoS properties of composed services using component properties. The composition properties can be evaluated based on different ontological classes and relationships.

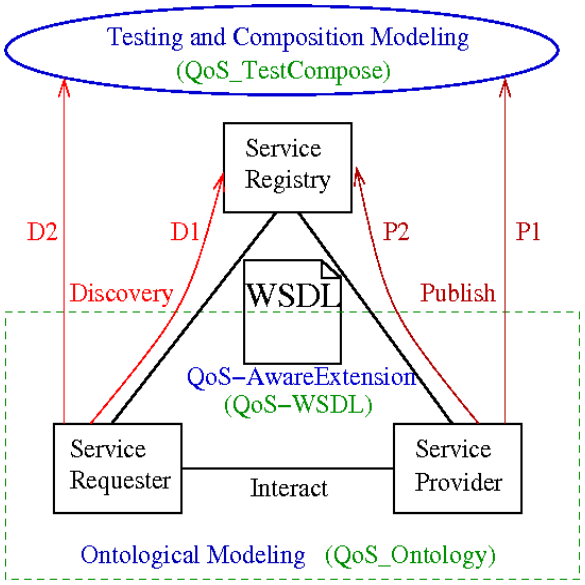

Figure. 1. QoS-Aware Service Oriented Architecture Modeling.

The QoS-aware framework for SOA is depicted in Figure 1. For a QoS-aware service to work, QoS_WSDL has to be prepared by instantiation of the QoS_Ontology model of the service. The value of the QoS properties can be obtained by using QoS-TestCompose as a testbed for analytical modeling or testing (P1). Once the Service Provider equips the service with the QoS extension, the provider can register the service specification on the Service Registry (P2). A Service Requester can query the Service Registry to discover qualified candidates by examining both functional and non-functional properties (D1). Functional properties are interpreted by reading WSDL, while non-functional properties are referenced by the QoS extensions. In case multiple services are selected with the same functional properties, the requester can use QoS properties to differentiate between the services. The selected services can be used to create new composed services and the QoS properties of the new services be obtained using the QoS-TestCompose Modeling(D2).

For those services that have not used our QoS extensions, our framework uses conventional selection and composition processes (P2 and D1).

B. QoS-Aware Framework with WS-BPEL Extension

Using our previous work, we now extended the QoS-Aware framework to use business processes. We discern service declaration types with atomic and process descriptions. An atomic service (AS) is the one whose provider offers functionality with design details but implementations hidden. Access to the service is achieved with required message exchange pattern (MEP) and binding of ports as shown in WSDL. In other words, an atomic service is opaque and represents the standard web services. We illustrated service selection and compositions with respect to service performance in [13]. However, at that time, we did not use any specific logic for the composition.

A process service (PS) is a service that facilitates collaborations between services controlled by business logic. A PS may be composed by multiple AS and/or PS. PSes can be nested. At the lowest layer of the hierarchy, a PS should only consist of ASes. We will show that both AS and PS can be modeled and analyzed in our QoS-aware framework.

There are many well-known business process modeling languages available to formally describe the interactions among different service components with business logic [19]. These languages rely on well-defined workflow formats. In some cases they use meta-data that can be used for management purposes. In this paper we use WS-BPEL or BPEL for short, to demonstrate the service selection and composition capabilities of our framework.

In the web service context, BPEL can be treated as a layer on top of WSDL [20]. BPEL provides the description of behavior and interactions of a process instance with its partners and resources through Web Service interfaces. Both the process and its partners are exposed as WSDL services [18]. Furthermore, BPEL follows WSDL model of separation between abstract information, such as message and port type, and concrete information, such as binding and endpoint. The two use cases for modeling BPEL processes are abstract and executable. Abstract processes describe the protocol that specifies the message exchange between parties without revealing their underlining implementations. While abstract processes may hide some of the required operational details, executable processes are fully specified and can be executed. Both abstract and executable processes share all the BPEL constructs, except that the former has additional opaque mechanisms for hiding operational details [18].

To include a PS in our framework, we assume that WSDL descriptions for all services are available. WSDL files describe how to use services, while BPEL describe collaborations among the services or tasks. In accordance to our previous design of the framework, only concrete WSDL is relied upon in our QoS-aware framework. Quality of Services with concrete bindings provides more specific range of values, derived from actual tests or analyses. The service that is extended for use in our framework can be viewed as an AS with a concrete WSDL. Or an abstract AS can be included in our framework, provided the QoS properties are derived through a concrete binding (as shown by process P1 in Figure 1).

Now we consider if the assumptions can be applied to the cases of acquiring a PS in the framework. For an executable PS, it is natural to assume that the services involved in the PS have concrete WSDLs, since an executable process is assumed to be concrete. For an abstract PS to be included in the framework, it must first be transformed into an executable PS. The transformation is called Executable Completion [18] in the web services context of WS-BPEL. The main algorithm of the transformation and related issues concerning QoS properties will be addressed in later sections.

With the adaptation of PS into the framework, we now consider the process of publishing and discovery operations for processes. Once again, an executable PS can be observed as services with concrete WSDL in the framework. To publish a PS service, it applies the same process P1 and P2, shown in Figure 1; for service registration of ASes, additional service meta data is added to the QoS_Ontology compartment for describing management related information. Since the framework differentiates a PS from an AS, the ontology model notes the service identification and service type classification when a service is instantiated. The additional information includes identification of a PS, the business process structure, and its sub-components. Note that the additional information of a PS is stored in the framework and is independent of the data minted in a Service Register of the SOA triangle. To discover a PS service, it makes no difference as to discovering any AS with QoS annotations in its registered WSDL (D1). Re-discovery of a PS service is required to first discover its sub-services as ASes, and submit the business process to QoS_TestCompose for updating QoS values (D2).

Although only concrete AS and executable PS are allowed in the framework, abstract processes can still be included. An abstract process can be viewed as embedding multiple use cases. The use cases are differentiated by their usage profiles. From the abstract processes, one can analyze the profiles to obtain specific values for QoS properties of the processes. To this end, we suggest that records of abstract processes be kept so as to facilitate QoS-aware compositions using different business process operations. We will discuss how PSes can be used in our framework in Section III.

III. Service Compositions with WS-BPEL in the QoS-Aware Framework

SOA enables a flexible and adaptable web service discovery and service composition. To allow for selection and composition based on QoS properties within our framework, we need to devise processes to guide QoS-aware business process selection and compositions. Since WS-BPEL is an established standard to describe business processes in the web services context, we will use BPEL to describe business processes in our framework.

Orchestration and choreography are two aspects of creating businesses from composite web services [20]. Orchestration refers to an executable process that interacts with internal and external web services. Since the executable process may include business logic and task execution order, it represents the control flow among the participating services. On the other hand, choreography refers to the interactions (or data flow) among participants who cooperate to achieve the objectives of the composed services. Choreography coordinates message exchanges occurring among services. For our purpose, we adapted BPEL4Chor [21], an extension of BPEL to address service composition, as the choreography framework. Engineers can use the language and available tools to readily model service interactions. We will show how this BPEL choreography can be used within our QoS-aware service composition framework.

The following subsections include discussions of the applicability of quality awareness to both orchestration and choreography compositions, and their operational processes. Note that the focus of the service composition here concerns non-functional properties while assuming the functional semantics in the selection and composition has already been

completed. We use the term service candidates to refer to the services already selected for composition based on functional requirements.

Our framework is designed to permit the use of many different approaches for specifying QoS properties, provided appropriate tools for selecting services meeting specific non-functional properties are also available.

A. Business Process Service Orchestration

A service orchestration is to organize the sub-services of a PS, and the message exchange with other services to achieve its service purposes. The PSes considered here are executable with their sub-services are also executable. Since the PS and its component services are all executable, they are eligible to apply the QoS extension when registering the service in the framework. Service composition from the perspective of orchestration involves sub-services selection. The PS selection for orchestration comes down to two scenarios: a fixed process organization, and process candidates of the same functionality with alternative design.

A PS may consist of m sub-services whose organization is based on the business process logic and how the tasks are ordered. Candidates for the m component sub-services are selected based on both functional and non-functional (QoS) properties. Since candidate services are assumed to use QoS extended WSDL, QoS references can be obtained for services and appropriate service components can be selected based on QoS properties.

In the case of multiple candidates of the same services with alternative business processes, they can be further classified as fixed or non-fixed sub-services. If the sub-services are fixed, the whole PS can be treated as AS. Then, the service selection only involves comparing the QoS values of the targeted non-functional attributes.

If the sub-services can be changed dynamically, each of the process candidates may be evaluated using multiple use cases. Each use case that belongs to a process candidate must be re-discovered for its QoS values. The result of QoS criteria for each candidate can be obtained, which can be used to match the requirements in order to make the decision.

B. Business Process Service Choreography

As stated perviously, choreography describes the interaction protocols (or data flows) among component services of a business process. While orchestration utilizes executable processes for modeling, choreography uses abstract process to describe the collaboration among service partners.

Since our QoS-aware framework requires concrete services with binding so that non-functional properties can be measured, the abstract nature of choreography in describing service interactions is not a direct fit for our framework. Thus, we need to extend the abstract interactions with appropriate concrete annotations of QoS attributes. Since our framework adapts BPEL as the descriptive language for business processes, we adapt BPEL4Chor [21] to model service choreography. We further annotate the interactions to make the choreography QoS-aware.

BPEL4Chor consists of three artifact types:

- Participant behavior description (PBD): It defines control flow dependencies between activities. It uses the Abstract Process Profile to describe requirements on the behavior of a participant. The profile inherits from Abstract Process Profile for Observable Behavior specified in BPEL, with the addition of identifying activities with unique identifiers. The PBD is essentially an abstract process with the additional attributes kept in the profile.
- Participant topology: It defines the collaboration structure of participant types, participants, and message links. The topology describes the communication structure of the service interactions among participants.
- Participant grounding: It defines the actual configuration of the choreography, and shows the connections to the concrete WSDL of the service participants. For each message link defined in the participant topology, a port type and its operation are specified. After the grounding, every PBD of the service can be transformed to an executable BPEL process based on their profiles.

An initial high level mapping from the modules of BPEL4Chor to our framework is straightforward. Although our framework requires concrete service data, abstract process is included in the framework. And, it is feasible to use abstract processes during the composition process before the new grounding of composition is admitted in the framework. The processes of adapting the composition to our QoS framework are presented below. We will refer to the processes shown in Figure 1 in our discussions below.

- From BPEL4Chor to QoS-Aware Extension
  The main product of a service choreography is an executable process. The new service can be included in the QoS-extension framework by first submitting to the QoS_TestComposite for QoS evaluation (D2). Corresponding process data is established with additional specific records for a choreography including PBD for all the participants and the composition topology. Recording a PBD is compatible with storing an abstract process, which is supported in the QoS-Aware extension.
- From QoS-Aware Extension to BPEL4Chor
  The main activity of BPEL4Chor is to identify a set of service participants to create a new service. The process involves selecting the service participants, extracting the PDB, and applying BPEL4Chor processes to compose the new service. The participants are restricted to only PSes since we have to identify the names of the operations. The QoS-aware framework facilitates the selection process by providing QoS values during discovery (D1). The selection process is similar to the selection process of a PS as introduced in Section III-A. Once we select the participants for composition, we will need the PDB for each participant. Since each participant selected is executable PS, there always exits one and only one abstract PS in the framework that belongs to the PS. Transforming a PS to a PDB is straightforward with adding the unique name to the message exchange operations. With the named message links, practitioners

then put together the required participant topology with the design. The grounding information for each linked operation is already available with an executable PS. Then BPEL4Chor composition process is complete, and the new composite service is created. To accommodate the created PS in the framework, the same processes mentioned in Section II-B is followed.

IV. From WS-BPEL to LQN

In this section, we explain the QoS_TestCompose module in our framework to illustrate how QoS properties of services that are composed using BPEL are calculated. Modeling non-functional properties for services and their composition is not always straightforward, since combining QoS properties of different services are based on underlying mathematical models. For example, given a process with the service components executed in sequential order, one may assume that the response time of the combined process is the sum of the response times of the component services. However, this will not be accurate because combining performance properties rely on stochastic processes. In other words, for obtaining performance attributes of a composed process we must use stochastic models. In our case, we use the layered queueing network (LQN) for modeling performance. However, other stochastic models and tools can be used to compute QoS properties of processes using BPEL.

The following subsections give a brief introduction to the essential elements of BPEL and LQN. Then we derive the transformation rules for mapping from BPEL compositions to models in LQN, and discuss how LQN can be used to compute performance attributes.

A. WS-BPEL constructs

WS-BPEL [18] is a standard language intended to describe business processes for web services. The idea is to represent collaborations among services or tasks described in the WSDL language. As a descriptive language using XML format to describe workflow of business process, BPEL consists of two types of activities: Basic and Structured.

Basic activities are atomic activities mainly describing service interactions. They include <receive> and <reply> , which represent waiting for a message, and response to a message respectively; <invoke> enables a web service operations offered by a service partner. The invocation enables either a one-way or request-response message exchanges. Other basic activities include <assign> to update a value of a variable, <exit> to end the process, <wait> to delay the execution, and <empty> to express no-op operation. Still others include <scope> , <throw> ,<compensate> , and <validate> that handle from the execution scopes to fault handling operations. New activity creation is also possible through <extensionActivity> .

Structured activities control the process order of activities. They can be nested in other structured activities as well. The constructs include <sequence> and <flow> to express sequential and parallel order of the enclosed tasks. The control flow constructs include <if> that sets a boolean condition for activities, <while> and <repeatUntil> that iterate through their enclosed processes until the condition becomes false; <forEach> controls the number of times the set of enclosed tasks can repeat, either running in sequential or parallel, while <pick> chooses among tasks to be executed depending on the occurrence of the event.

B. Layered Queueing Networks

Layered queueing network [22] is an extended queueing model with the layered structure representing servers at higher levels making requests to servers at lower levels. Each task in the model involves sharing and consuming processing resources. An entry of a task can be modeled as the service operation stub receiving requests and responding with a reply to higher level systems. The entry can be further refined with activities representing the workflow of its sub-components which are organized with precedence operators, such as fork and join. For each task and activities, there are resource requirements specified, as service demand in time. The interactions between different servers and their tasks can be modeled with phases representing message receipt and response in different time slots. The nature of the communication can be defined as synchronous and asynchronous, which model blocking and non-blocking interactions respectively.

As modelers put together the service architecture and information needed for the system integration, a queueing network is created. The system modeling can be subjected to either open or close networks during performance analysis. LQN comes with an analytic solver (lqns) and a simulator (lqsim) to generate the performance indexes such as response time, utilization, and throughput.

LQN models can also be expressed in XML format. A further analysis to explore the design space with different combination of system configuration is also possible with its LQX tool. LQX is a general purpose programming language used for the control of input parameters to the LQN solver system. The language allows a user to put together a wide range of different set of input parameters, and solve the model accordingly.

C. Transition Rules from BPEL to LQN

A structure of business process in BPEL largely consists of activities and their corresponding fault handlers, in addition to variables, correlation sets, and partner links. Since performance evaluation of the business processes is the focus here, the derivation of the transformation rules only focuse on the process activities. For the performance analysis purpose, the activities in the event and fault handlers can follow the same set of rules, and integrated with the activities in the main processes.

The main process activities usually begins with a list of sequential activities. The behavior of the activities, both basic and structured, are described by the control constructs. The main task of the transformation is to maintain the same activity orders as in BPEL when creating the LQN model. For basic activities, the order of the behavior relates to mainly communication protocols. For structured activities, the order can be focused on the mapping of business logic.

TABLE I. MAPPING BPEL BASIC ACTIVITY TO LQN ELEMENTS.

BPEL Basic Activity	LQN	Description
<receive>	Pre-precedence (or a join-list)	Getting a message from a service partner.
<reply>	Request (send-no-reply) : direct reply Request (forwarded) : indirect reply	Sending a message to a service partner.
<invoke>	Request (send-no-reply) : one-way Request (rendezvous) :request-response	Invocation of a service offered by a service partner. It can be one-way or request-response interactions.
<wait>	Activity with a think time	A delay for a timer.
<empty>	Activity with zero service time	A no-op holder which does nothing.
<exit>	N/A	Immediate termination
<assign>	N/A	Assign a value to a variable.
<validate>	N/A	Validate the value of variable defined in WSDL.
<throw>	N/A	Generate a fault from business process. Fault handler needs to be specifically modeled.
<rethrow>	N/A	Regenerate a fault from fault handler. Fault handler needs to be specifically modeled.
<compensate>	N/A	Compensate actions can not be completed. Fault handler needs to be specifically modeled.
<compensateScope>	N/A	Compensate actions can not be completed in a specified scope. Fault handler needs to be specifically modeled.

TABLE II. MAPPING BPEL STRUCTURED ACTIVITY TO LQN ELEMENTS.

BPEL Structure Activity	LQN	Description
<sequence>	Precedence: Sequence	A list of service activities executed in the specific order.
<flow>	Precedence: And-Fork & And-Join	A bag of service activities executed in concurrent and finished in synchronization.
<if>	N/A [Use Or-Fork & Or-Join to emulate the condition with a probability 1 or 0.]	Take different actions depends on the Boolean condition.
<pick>	N/A [Use Or-Fork & Or-Join to emulate the condition with a probability p.]	Activity is chosen depending on the kind of message or timeout events.
<while>	N/A [Precedence: Loop to emulate the number of iteration.]	Iteration on the Boolean condition evaluated to true.
<repeatUntil>	N/A [Precedence: Loop to emulate the number of iteration.]	Iteration will stop on the Boolean condition evaluated to true.
<forEach>	N/A [Precedence:Loop to emulate the number of iteration]	Repeat activities multiple times, activities in each iteration can be modeled with <sequence>or <flow>

The order of the control flow in the transformation is realized using precedence of activity connections in LQN tasks. The precedence can be sub-classed into Join and Fork for modeling synchronization and concurrency of activities. To connect one activity to another, the source activity connects to a pre-precedence (or Join). A pre-precedence in turn connects to a post-precedence (or Fork), and then to the destination activity. More details on precedence types can be found in the LQN User manual [22].

Service requests in LQN can be of three types: rendezvous, send-no-reply, and forwarded. Rendezvous is a blocking synchronous request, while the send-no-reply is an asynchronous request. Forwarded requests are redirected to a subsequent server, which may forward the requests again, or reply to the original client. In the translation, we consider the message exchange pattern to match either blocking or non-blocking, and either one-way or two-way for service invocation.

The summary of mapping of basic constructs are listed in Table I, while the mapping of structured constructs are listed in Table II. For each mapping entry, a brief description is included. For those elements that have no direct LQN semantic counterparts, we use (N/A) with explanation. Since the focus of the transformation is on performance analysis, the corresponding performance models for fault handling activities should be obtained by following the error handling mechanisms designated in the processes. The handling processes can then be subjected to the transformation rules to obtain appropriate performance models. The part of fault handling of the transformation and its performance evaluation is not included in this paper.

D. Data Dependency in Transformation

There is no direct equivalent LQN transformation for the BPEL conditional construct such as if-else. However, an Or-Fork representing a branching point with a given probability p to a selected process path can emulate the semantics of if construct. The probability is set to 1.0 for the if-clause, if the condition should be evaluated to true. On the other hand, the else-clause will be taken with the probability of the if-clause set to zero, if the condition should be evaluated to false. The transformation from <if>in BPEL to LQN can thus be expressed using the semantic of Or-Fork and Or-Join with appropriate probability p. The probability depends on the variables involved in the condition. The frequency of which path is taken depends on the statistical or empirical evaluations. Each sample represents a specific service system configuration that is invoked in a specific use case.

The conditional variables can be related to either service workload or the frequency of the variable assignments. For example, in a sorting algorithm, workload as the size of input list can impact the service time. The business process may

consider splitting the input into smaller sizes, and merging the result later. The condition may also depend on a multivariate function when multiple outcomes are possible. Data profiling and other empirical evaluations can be used to assign probability values with each outcome [23]. Similar approach can be applied to <pick>where the Boolean condition becomes the frequency of the message variable. The sum of the probabilities of each case in the Or-Fork is to be 1.

For the loop control statements, such as <while>and <repeatUntil>, the Boolean condition should be analyzed using the number of times the iteration will be executed. The counterpart in the LQN is a loop (for a service) which is executed desired number of times. <forEach>is similar to these iterative controls with the addition of specifying the execution type, in sequence or parallel, of activities in the loop clause.

V. Case Studies

A. Facial Detection and Recognition Example

A building security monitoring system, which uses facial detection and recognition technique, is used as the example to demonstrate how to use our framework. The purpose of the system is to detect intruders, and raise an alarm when intruders are detected, as well as recognizing the intruders using facial recognition software to compare with existing database of stored images.

A general computation of facial detection and recognition is split into multiple tasks – signal processing, image analysis of machine learning algorithms and processes. In our example, the service is divided into three modules: Facial Detection (FD), Image Converter (IC), and Facial Recognition (FR).

- FD receives video frame input and detects if there are faces appearing in the image. If no face is detected, no action will be taken. However, if faces are detected, alarm messages will be sent and image frame will be the output for further processing.
- IC receives image frames with faces detected, and prepare the normalized file formats for each face. The output consists of the images that can be compared against images stored in the databases.
- FR receives the normalized face images as input, and sets the connections to databases containing images of faces for identification. Once there is a match, a report is sent to human operators with information about the persons identified.

The three modules will be considered as web services, and our goal is to create a new web service that will combine these component services, using sequential composition in the order of FD, IC, and FR. The process sequence of the three services in BPEL is shown in Listing 1, Listing 2, and Listing 3 respectively.

Each BPEL is transformed into a LQN model for analysis. To submit the service into the framework, the LQN model is analyzed with the result of the performance indexes obtained from QoS values of individual services. The transformation of the LQN models are shown in Figure 2, Figure 3, and Figure 4 respectively.

Listing 1. Facial Detection BPEL (FD).

```
<sequence>
  <opaqueActivity name="DetectFacialProcess" />
  <if>
    <condition opaque="yes" />
      <flow>
        <invoke wsu:id="SubmitICReq"/>
        <opaqueActivity name="SubmitAlarm"/>
      </flow>
  <else>
      <opaqueActivity name="SubmitNoResult" />
  </else>
  </if>
</sequence>
```

Listing 2. Image Converter BPEL (IC).

```
<sequence>
  <receive wsu:id="ReceiveICReq" createInstance="yes
      " />
  <if>
    <opaqueActivity name="SplitImageFrame"/>
    <forEach name="splitFile" wsu:id="
        NormalizeFrameSize" parallel="yes">
      <startCounterValue>1</startCounterValue>
      <finalCounterValue>2</finalCounterValue>
      <scope>
        <opaqueActivity name="NormalizeMultipleImage
            "/>
      </scope>
    </forEach>
  <else>
    <opaqueActivity name="NormalizeNormalImage"/>
  </else>
  </if>
  <invoke wsd:id="SubmitFRReq"/>
</sequence>
```

Listing 3. Facial Recognition BPEL (FR).

```
<sequence>
  <receive wsu:id="ReceiveFRReq" createInstance="yes
      "/>
  <forEach wsu:id="queryDatabase" parallel="yes"
      opaque="yes">
    <startCounterValue>1</startCounterValue>
    <finalCounterValue>3</finalCounterValue>
    <scope>
      <opaqueActivity wsu:id="
          FacialRecognitionProcess" />
    </scope>
  </forEach>
</sequence>
```

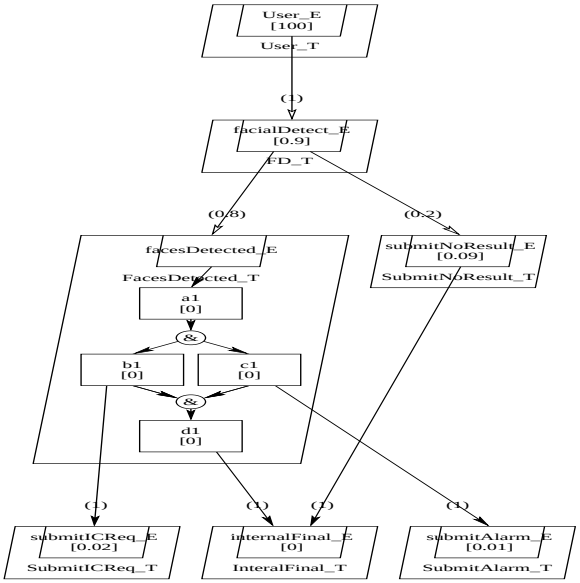

Figure. 2. Facial Detection LQN Model.

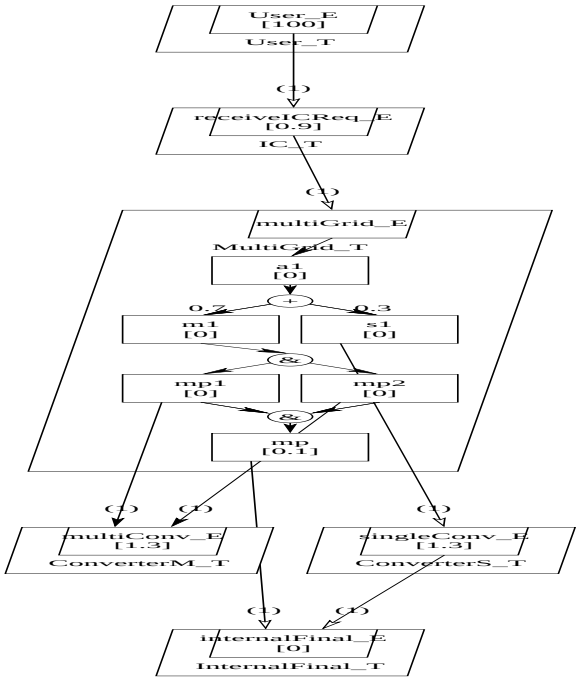

Figure. 3. Image Converter LQN Model.

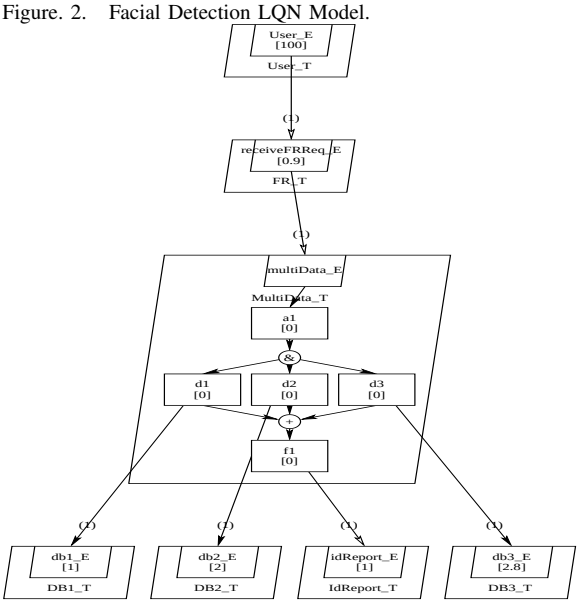

Figure. 4. Facial Recognition LQN Model.

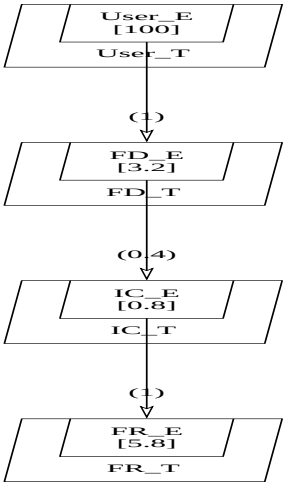

Figure. 5. FD IC FR Sequential Composition LQN Model.

The entire composition for the building security application can be sought in different ways depending on the approaches the engineers use. We demonstrate two example scenarios to show how the framework facilitates compositions. In a simplified scenario, all services can be considered as atomic services, while in a more flexible scenario, the composition utilizes the workflow processes to leverage the service choices in order to gain a better performance.

To compose the the system in the simplest case, service discovery process (D1, shown in Figure 1) is applied. For FD, IC, and FR, QoS values such as service execution time are obtained from their QoS extended WSDL files. A simple version of the sequential BPEL expression is created in Listing 4.

Listing 4. Atomic Composition of FD.

```
<sequence>
  <opaqueActivity name="FD_Process" />
  <opaqueActivity name="IC_Process" />
  <opaqueActivity name="FR_Process" />
</sequence>
```

The transformation steps along with the quality attributes obtained from the WSDL extension of each services, together create the LQN model of the composition. The LQN model is depicted in Figure 5. The new composition along with the performance indexes resulting from analyzing LQN models can be published using service publish process (P2, shown in Figure 1).

A more flexible way to consider the composition is to observe the web services components as processes. We first retrieve web services along with their processes. Applying BPEL4Chor processes, a topology file is created to build the service interactions. A snapshot of the topology configuration is shown in Listing 5. The result of the composition along with the derived BPEL and corresponding LQN model is shown in Figure 6.

Listing 5. Choreography Topology for Process Service Composition of FD

```
<?xml version="1.0" encoding="UTF-8"?>
<topology name="
     example_facialDetectRecognizetopology"
   targetNamespace="http://agentmode.com/choreography
       /facial/topology"
   xmlns:fd="http://agentmode.com/choreography/facial
       /detecter"
   xmlns:ic="http://agentmode.com/choreography/facial
       /converter"
   xmlns:fr="http://agentmode.com/choreography/facial
       /recognizer"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance">

  <participantTypes>
    <participantType name="FD"
        participantBehaviorDescription="fd:detecter"
         />
    <participantType name="IC"
        participantBehaviorDescription="ic:converter
        " />
    <participantType name="FR"
        participantBehaviorDescription="
        fr:recognizer" />
  </participantTypes>

  <participants>
    <participant name="detecter" type="FD" selects="
        converter" />
    <participant name="converter" type="IC" selects=
        "recognizer" />
    <participant name="recognizer" type="FR" />
  </participants>

  <messageLinks>
    <messageLink name="icRequest" sender="detecter"
        sendActivity="SubmitICReq" receiver="
        converter" receiveActivity="ReceiveICReq"
        messageName="icRequest" />
    <messageLink name="frRequest" sender="converter"
         sendActivity="SubmitFRReq" receiver="
        recognizer" receiveActivity="ReceiveFRReq"
        messageName="frRequest" />
  </messageLinks>
</topology>
```

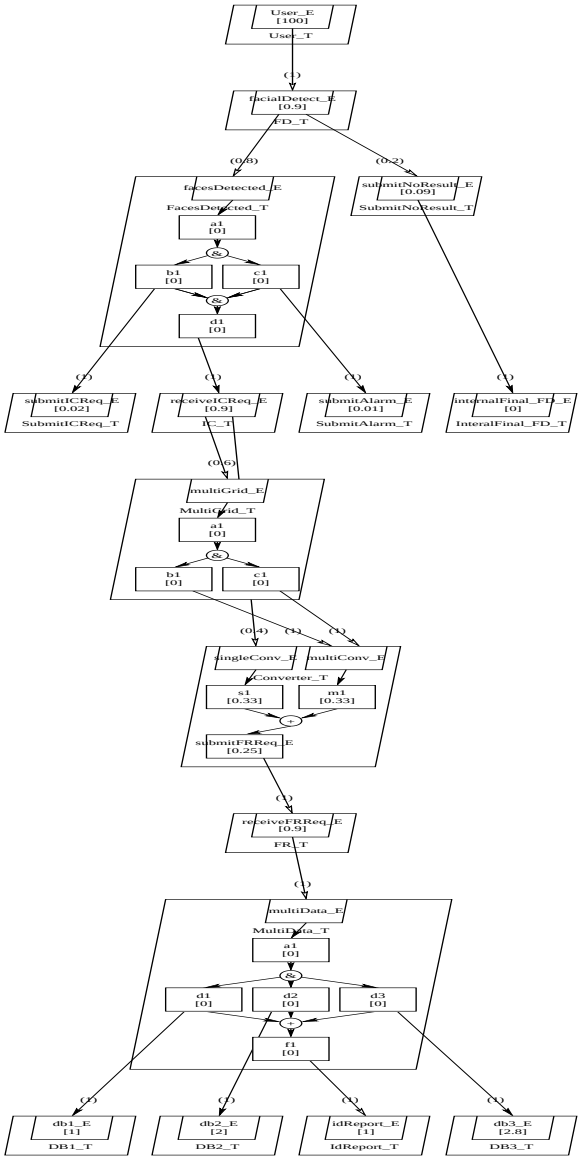

Figure. 6. FD IC FR Choreography Model.

B. Data Dependency Considerations

In the Image Converter (IC) BPEL process, the if-clause distinguishes between single and multiple faces that need to converted, since converting multiple faces increases workload on the processing systems. If the image contains multiple faces, it may be desirable to use multiple processes executing concurrently to improve the speed of IC process. Here we model two identical servers executing the same job by splitting the conversion tasks into two assuming two faces are detected. Each server, which either processes the single task or two tasks, has the same execution performance and same capacity. The service time depends on the probabilities associated with detecting one or two faces. In this example, we vary the if-clause probability from 0.01 to 0.99, and estimate the effective

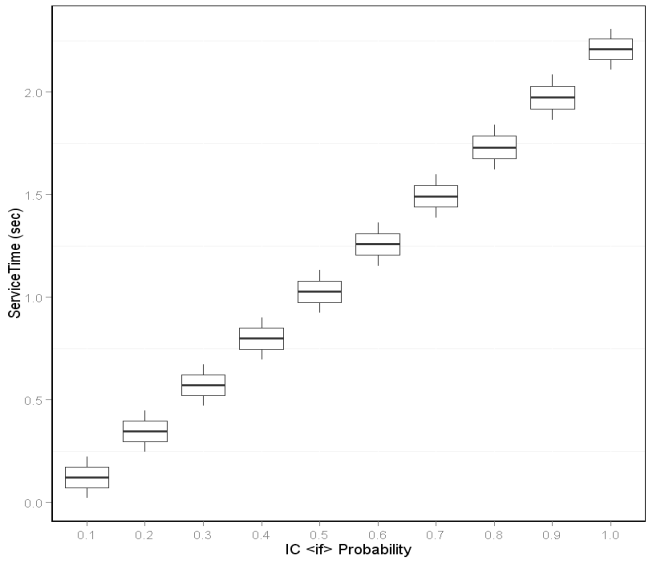

Figure. 7. IC Service Execution Time vs if-clause Probability.

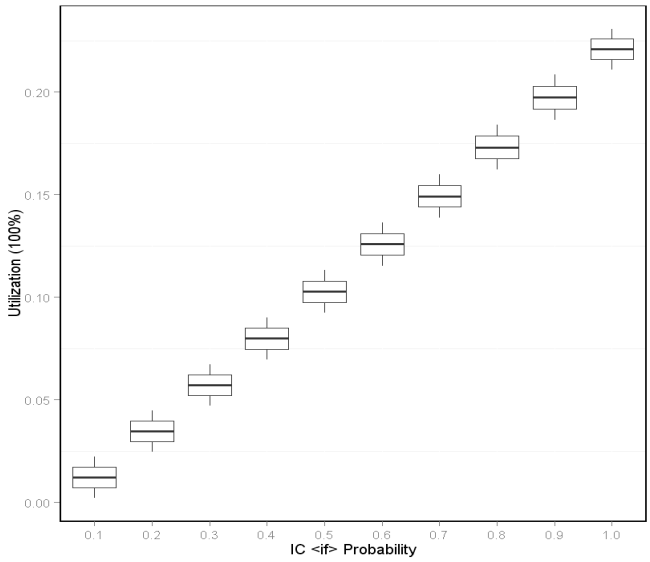

Figure. 8. IC Utilization vs if-clause Probability.

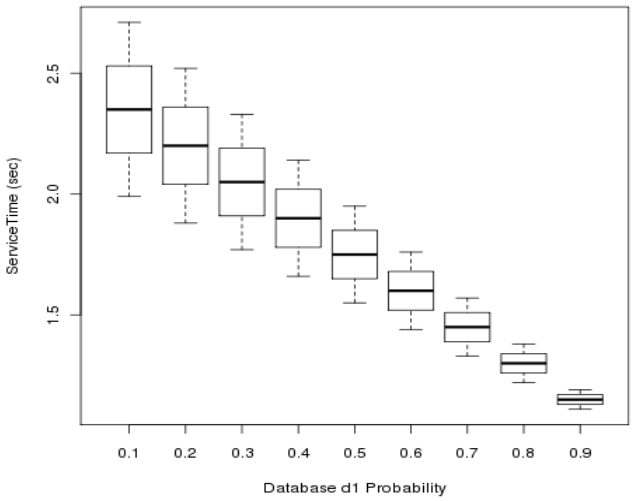

Figure. 9. FR Service Execution Time vs Database Probability.

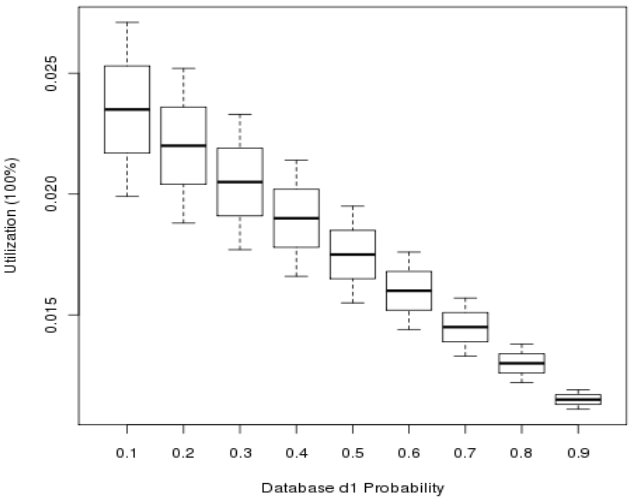

Figure. 10. FR Utilization vs Database Probability.

performance. Figure 7 shows the execution time ranges while the probability with the if-clause is changed. Figure 8 shows the utilization of the image conversion servers.

Similar method is also be used with Facial Recognition (FR) BPEL process, using either a single or multiple tasks to compare the faces with those in the database. To further speedup the process, the database may be organized into frequently accessed faces and less frequently accessed faces. In this example, we separated the facial detabases into three separate databases, d1, d2, and d3. Rather than concurrently querying all three databases, modelers can select just one representative database based on the likelihood of finding a match. Figure 9 shows the execution time of the FR service while adjusting the probability of success with d1. In Figure 10 shows the utilization versus the probabilities.

C. Performance Space Considerations

Various design topologies that yield different service performances can also be considered. In this example, system structure and server capacity are explored. Consider the IC example for multiple image conversion. Instead of running two converters concurrently, suppose we want to explore the alternatives that execute them sequentially as a two-step

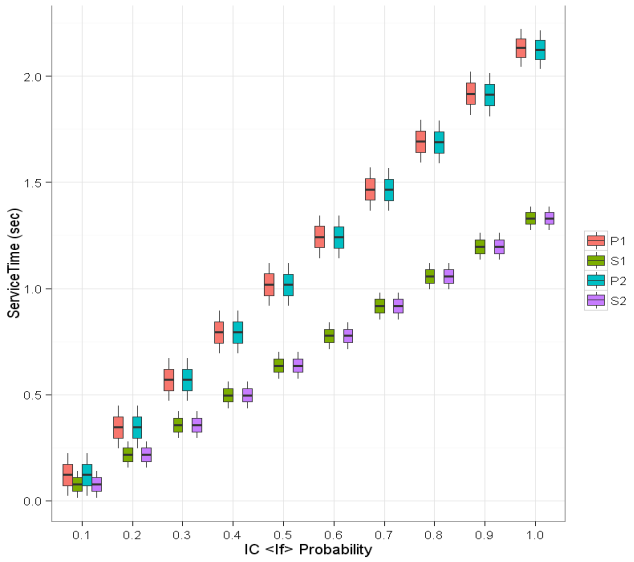

Figure. 11. IC Service Execution Time vs if-clause Probability on Service Design Alternatives.

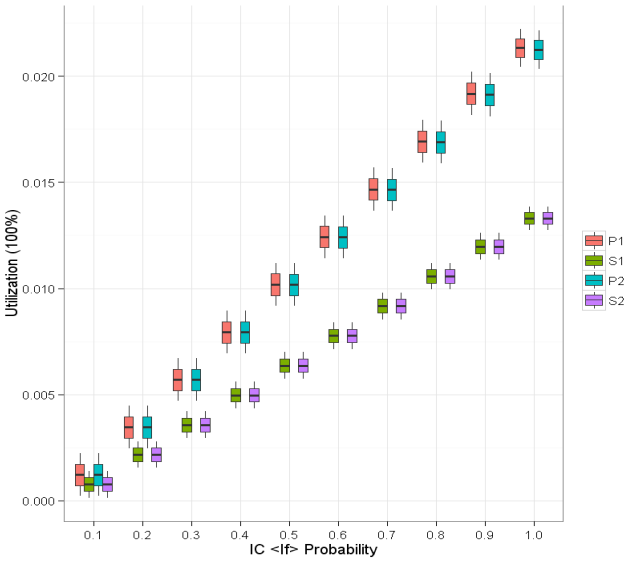

Figure. 12. IC Utilization vs if-clause Probability on Service Design Alternatives.

pipeline, where the execution time of each step is only one forth compared to the parallel ones. There are also options that the server can be equipped with single or multiple processors (e.g., multicore systems) to speed up the service. Together these options can be analyzed by the LQN. Figure 11 shows the service times of the converter compared to the previously shown split workflow; we use S to represent the sequential flow and P to represent parallel workflow and the suffix indicates the number of processors.

VI. Related Works

The promise of service oriented computing, and the availability of web services in particular, promote delivery of services and creation of new services composed of existing services [24] – service components are assembled to achieve integrated computational goals. Business organizations strive to utilize the services and provide new service solutions and they will need appropriate tools to achieve these goals [25]. As webs and internet based services grow into clouds, inter-dependency of services and their complexity increases tremendously. The cloud ontology suggested in [26] depicts service layers from a high-level, such as Application and Software, to a low-level, such as Infrastructure and Platform. Each component resides at one layer can be useful to others as a service. It hints the amount of complexity resulting from not only horizontal but also vertical integrations in building and deploying a composite service. Our framework tackles the complexity of the selection and composition issues with additional qualitative information to the service descriptions in BPEL. Engineers can use BPEL to explore design options, and have the QoS properties analyzed for the design. QoS properties of each service are annotated with our WSDL extension for future references.

There have been several works on QoS-awareness for BPEL services. In [27], a service broker offers composite service with multiple QoS classes to the users. The selection scheme optimizes aggregated QoS of incoming request flows using linear programming. In [28], business workflow is parsed into a tree structure, where heuristic algorithms are applied for selecting service candidates based on QoS properties. In [29], QoS is acquired by constructing activity graph and reasoning the dependencies among them for the QoS parameters, including response time and cost. A declarative approach is proposed in [30] by creating the policy-based language QoSL4BP to specify QoS constraints and logic over scopes of the orchestration. QoS planning, monitoring, and adaptation of the BPEL can be expressed to model the service behavior. An extension to BPEL for specifying QoS and non-functional requirements is proposed in [31]. The extension point is at the service invocation of a partner web service. Our framework is able to provide compatible SOA infrastructure to test on different approaches surveyed above and others, however, the foundation to address QoS properties for BPEL relies on the WSDL extension at the service level [13]. The benefit is that modeling business services to annotate QoS properties is compatible with standard WS-BPEL without the need to introduce other artifacts.

Performance evaluation on BPEL often involves analytical model construction by transforming the business logic into appropriate model logic. In [32] and [33], BPEL processes are translated into stochastic petri nets by a set of rules to model waiting queues and their performance distributions. In [34], a formalism for the SYNTHESys framework [35] is generated by the translation from BPEL to PerfBPEL models. The PerfBPEL serves as the performance annotation to the BPEL workflow, and a Markov chain for the model can be generated. Then multi-formalism modeling technique enables

the use of other tools for analysis. In [36], BPEL is annotated with a performance metadata for operations and resources. A queueing model can be derived from these annotations to generate the bounds of throughput and response times. While the translation is similar to ours, our framework uses an ontology for QoS data management, and use LQN to keep the original mapping of service architecture. In [37], support from abstract to executable processes for service orchestration is proposed according to three levels: needed functionality, expected QoS, and composition flow. Process realization, discovery, classification, and selection steps lead to the composition. The expected QoS is reasoned by a classification method to select services for composition. While our framework can also rank services using ontology models and plug in different selection filters, the QoS prediction for service composition is based on the result of modeling analysis.

A feature-completed Petri net semantic counterpart for BPEL has been established in [17]. As mapping from BPEL is easily obtained, Petri net can be subjected to formal model checking [38] and workflow performance analysis [39].

VII. Conclusion

In this paper, we described our framework for web service QoS-aware selection and composition of web services using BPEL. With the foundation of WSDL extension to annotate non-functional properties, web services can be selected based on both functional and non-functional (QoS) properties. We described the process for publishing and discovering services which meet requirements in the standard service oriented architecture. We show that services in BPEL description can be seamlessly accommodated in the framework. By adapting BPEL and BPEL4Chor for service composition, we reason about the feasibility of service orchestration and choreography in our framework. To illustrate the applicability of our framework to derive QoS properties of composed services, we use performance properties such as throughput, response times and utilization. To this end, we described transformation rules for converting BPEL into appropriate queuing networks which can be used by the LQN (Layered Queuing Network) tool that can compute throughput, utilization, and response times. We used a case study to demonstrate this process. Although we focused on performance in the paper, our framework can also be used to compute other QoS properties such as reliability, security, availability, with appropriate rules for converting BPEL logic into corresponding models and tools for obtaining QoS properties from these models.

VIII. Acknowledgements

This work is supported in part by the NSF Net-Centric IURCRC and a grant #1128344. The authors acknowledge help given by Sagarika Adepu.

References

[1] T. Erl, *Soa: principles of service design*. Prentice Hall Upper Saddle River, 2008, vol. 1.

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

[3] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *Internet Computing, IEEE*, vol. 6, no. 2, pp. 86–93, 2002.

[4] 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.

[5] R. Mietzner, C. Fehling, D. Karastoyanova, and F. Leymann, "Combining horizontal and vertical composition of services," in *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–8.

[6] 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.

[7] R. Hamadi and B. Benatallah, "A petri net-based model for web service composition," in *Proceedings of the 14th Australasian database conference-Volume 17*. Australian Computer Society, Inc., 2003, pp. 191–200.

[8] D. Bonetta, A. Peternier, C. Pautasso, and W. Binder, "A multicore-aware runtime architecture for scalable service composition," in *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*. IEEE, 2010, pp. 83–90.

[9] Y. Liu, A. H. Ngu, and L. Z. Zeng, "Qos computation and policing in dynamic web service selection," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004, pp. 66–73.

[10] N. Limam and R. Boutaba, "Assessing software service quality and trustworthiness at selection time," *Software Engineering, IEEE Transactions on*, vol. 36, no. 4, pp. 559–574, 2010.

[11] S. Reiff-Marganiec, H. Q. Yu, and M. Tilly, "Service selection based on non-functional properties," in *Service-Oriented Computing-ICSOC 2007 Workshops*. Springer, 2009, pp. 128–138.

[12] H.-C. Wang, C.-S. Lee, and T.-H. Ho, "Combining subjective and objective qos factors for personalized web service selection," *Expert Systems with Applications*, vol. 32, no. 2, pp. 571–584, 2007.

[13] C. Lin, K. Kavi, and S. Adepu, "A description language for qos properties and a framework for service composition using qos properties," in *ICSEA 2012, The Seventh International Conference on Software Engineering Advances*, 2012, pp. 90–97.

[14] T. Anstett, F. Leymann, R. Mietzner, and S. Strauch, "Towards bpel in the cloud: Exploiting different delivery models for the execution of business processes," in *Services-I, 2009 World Conference on*. IEEE, 2009, pp. 670–677.

[15] F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana, *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR Englewood Cliffs, 2005.

[16] C. Ouyang, E. Verbeek, W. M. van der Aalst, S. Breutel, M. Dumas, and A. H. ter Hofstede, "Wofbpel: A tool for automated analysis of bpel processes," in *Service-Oriented Computing-ICSOC 2005*. Springer, 2005, pp. 484–489.

[17] N. Lohmann, "A feature-complete petri net semantics for ws-bpel 2.0 and its compiler bpel2owfn," *Techn. report*, vol. 212, 2007.

[18] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland *et al.*, "Web services business process execution language version 2.0," *OASIS Standard*, vol. 11, 2007.

[19] H. Mili, G. Tremblay, G. B. Jaoude, E. Lefebvre, L. Elabed, and G. E. Boussaidi, "Business process modeling languages: Sorting through the alphabet soup," *ACM Computing Surveys (CSUR)*, vol. 43, no. 1, p. 4, 2010.

[20] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[21] G. Decker, O. Kopp, F. Leymann, and M. Weske, "Bpel4chor: Extending bpel for modeling choreographies," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 296–303.

[22] G. Franks, P. Maly, M. Woodside, D. C. Petriu, and A. Hubbard, "Layered queueing network solver and simulator user manual," *Real-time and Distributed Systems Lab, Carleton University, Ottawa*, 2005.

[23] D. Ivanovic, M. Carro, and M. Hermenegildo, "Towards data-aware qos-driven adaptation for service orchestrations," in *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 107–114.

[24] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[25] M. B. Blake, W. Tan, and F. Rosenberg, "Composition as a service [web-scale workflow]," *Internet Computing, IEEE*, vol. 14, no. 1, pp. 78–82, 2010.

[26] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE'08*. IEEE, 2008, pp. 1–10.

[27] V. Cardellini, E. Casalicchio, V. Grassi, and F. Lo Presti, "Flow-based service selection forweb service composition supporting multiple qos classes," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 743–750.

[28] D. Comes, H. Baraki, R. Reichle, M. Zapf, and K. Geihs, "Heuristic approaches for qos-based service selection," *Service-Oriented Computing*, pp. 441–455, 2010.

[29] D. Mukherjee, P. Jalote, and M. Gowri Nanda, "Determining qos of ws-bpel compositions," *Service-Oriented Computing–ICSOC 2008*, pp. 378–393, 2008.

[30] F. Baligand, N. Rivierre, and T. Ledoux, "A declarative approach for qos-aware web service compositions," *Service-Oriented Computing–ICSOC 2007*, pp. 422–428, 2007.

[31] V. Agarwal and P. Jalote, "From specification to adaptation: an integrated qos-driven approach for dynamic adaptation of web service compositions," in *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 275–282.

[32] M. Teixeira, R. Lima, C. Oliveira, and P. Maciel, "Performance evaluation of service-oriented architecture through stochastic petri nets," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 2009, pp. 2831–2836.

[33] D. Bruneo, S. Distefano, F. Longo, and M. Scarpa, "Qos assessment of ws-bpel processes through non-markovian stochastic petri nets," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[34] E. Barbierato, M. Iacono, and S. Marrone, "Perfbpel: A graph-based approach for the performance analysis of bpel soa applications," in *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on*. IEEE, 2012, pp. 64–73.

[35] M. Iacono and M. Gribaudo, "Element based semantics in multi formalism performance models," in *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, 2010, pp. 413–416.

[36] M. Marzolla and R. Mirandola, "Performance prediction of web service workflows," *Software Architectures, Components, and Applications*, pp. 127–144, 2007.

[37] Z. Azmeh, M. Huchard, F. Hamoui, and N. Moha, "From abstract to executable bpel processes with continuity support," in *Web Services (ICWS), 2012 IEEE 19th International Conference on*. IEEE, 2012, pp. 368–375.

[38] F. Van Breugel and M. Koshkina, "Models and verification of bpel," *Unpublished Draft (January 1, 2006)*, 2006.

[39] W. M. van der Aalst, "The application of petri nets to workflow management," *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.