

## SOFTWARE ENGINEERING FOR WEB SERVICES WORKFLOW SYSTEMS

M. BRIAN BLAKE

*Department of Computer Science, Georgetown University, 3<sup>rd</sup> Floor St. Mary's Hall,  
Washington, DC, 20057-1232, USA  
blakeb@cs.georgetown.edu  
<http://www.cs.georgetown.edu/~blakeb>*

LISA SINGH

*Department of Computer Science, Georgetown University, 3<sup>rd</sup> Floor St. Mary's Hall,  
Washington, DC, 20057-1232, USA  
singh@cs.georgetown.edu*

Received (11 May 2007)

Revised (31 December 2007)

Accepted (30 January 2008)

Service-oriented computing (SOC) suggests that many open, network-accessible services will be available over the Internet for organizations to incorporate into their own processes. Developing new software systems by composing an organization's local services and externally-available web services is conceptually different from system development supported by traditional software engineering lifecycles. Consumer organizations typically have no control over the quality and/or consistency of the external services that they incorporate, thus top-down software development lifecycles are impractical. Software architects and designers will require agile, lightweight processes to evaluate tradeoffs in system design based on the "estimated" responsiveness of external services coupled with known performance of local services. We introduce a model-driven software engineering approach for designing systems (i.e. workflows of web services) under these circumstances and a corresponding simulation-based evaluation tool.

*Keywords:* Web service evaluation; workflow management system; software engineering and modeling

### 1. Introduction

Commercial, academic, and government organizations alike may be able to expand their capabilities by sharing their underlying software services. The notion of thousands or even millions of universally accessible service-based capabilities is not only promising to the individual looking for a specific consumer-based service, but also to organizations hoping to enhance their own capabilities by incorporating the services of external entities. With respect to the latter, *service-oriented computing (SOC)* promotes an open environment where network-accessible services, or web services are universally-available across organizations leveraging various supporting technologies (i.e. Simple Object Access Protocol (SOAP) for messaging, the Web Services Description Language (WSDL) for service specification, and Universal Description, Discovery, and Integration (UDDI) for service storage and discovery) [21][22].

There are several specific issues in the strategy-based composition of web services. Software designers require an adequate list of evaluation factors in order to determine the most efficient system configurations. Variables that affect business processes include service dependencies, sub-process arrival rates, service reliability, network delay, etc. Many of these factors are not constant and therefore need to be considered by software engineers when making specific design decisions. In some cases, business process factors, such as the sequence of services and the arrival rate of processes, vary from process to process and domain to domain. For example, the arrival rate of requests to a travel reservation process may differ from that of an online product acquisition process. Therefore, modeling and approximating these business process factors can help software engineers make more informed design decisions. In this paper, we introduce a performance model that incorporates both process-oriented (i.e. workflow-oriented) and system-oriented factors for business processes consisting of web services. In addition, this model is integrated with industry-standard software modeling approaches such that practicing engineers can visualize system designs. This performance model is evaluated using a new semi-automated simulation-based evaluation software that facilitates side-by-side analysis of multiple service-oriented system configurations. This paper extends previous work [6] by including a performance model with formalized equations and additional experimentation.

The paper proceeds in the following section with a discussion of the model-driven software engineering process. This process is discussed in terms of the underlying models. In the next section, we discuss the design of the supporting simulation software. In the subsequent sections, we detail and evaluate two experimental cases. In the final sections, we discuss the merits of this work with regards to related work and future investigations.

## **2. A Model-Driven Spiral Design Process**

In this work, we suggest that software engineers for service-oriented computing must assimilate a new role when designing systems for new business processes. Software engineers must analyze requirements for building system components, while concurrently analyzing the ability of a composed set of inter-organizational services to fulfill those requirements. Figure 1 summarizes this approach to software engineering within service-oriented computing. The software engineer will have two interleaving cycles. In the first cycle, the software engineer will elicit requirements from domain experts or stakeholders to determine their specific need. At the same time, the software engineer will collect information about local known services. The software engineer will develop business process views (i.e. a high-level, workflow-oriented view defining the process consisting of inter-organizational web services) and system interaction views (i.e. a low-level view describing complex implementation protocols). Both of these views are defined in detail in later sections. The software engineer models these views consisting of the best set of known information while incorporating feedback from the stakeholders.

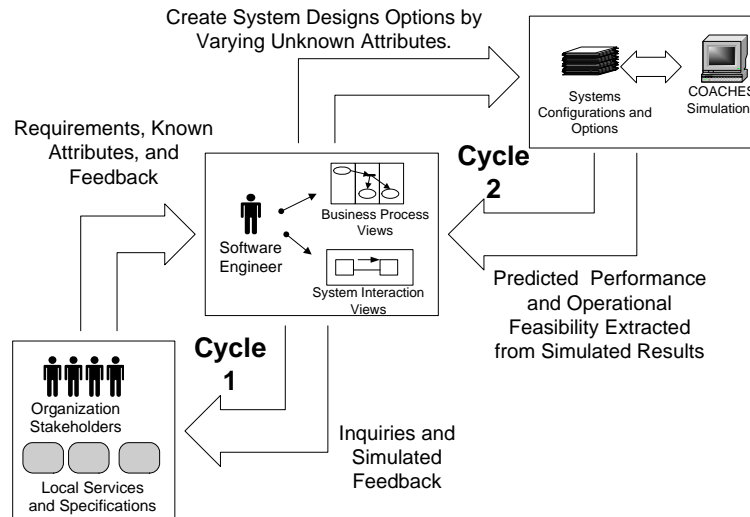


Fig. 1. An iterative SOC-oriented software design process.

In the second cycle, as the software engineer feels comfortable that the views are representative of his/her understanding, he/she then augments the views with predicted or estimated information about services that may lie outside of the organizational boundaries. The mixture of unknown external conditions and known local factors allows the software engineer to generate many optional designs and system configuration alternatives. Using a specialized software simulation application called COACHES [4] [5], the models are simulated and the predicted system operations are presented to the designer. Finally, the software designer uses the resulting simulation information to iterate between cycles to help planners make decisions and create strategies. Software designers must communicate the resulting operational performance information with domain and business subject matter experts. Within the scope of this paper, the business experts must determine if the quality of the processes fulfill their requirements, a binary decision. In future work, we intend to develop automated approaches to align business requirements to the simulated measures. The next section describes the performance attributes important to modeling service-oriented architectures and the following section describes the modeling approach.

### 3. Evaluative Attributes of the SOC Domain

The performance and contextual factors of most interest in our work can be classified into the following categories: *system load/traffic*, *process management*, *system communication*, and *service processing*. System load/traffic involves the rate at which the web services workflow system receives requests. Considering the presence of a web services workflow application that constructs processes from job requests (i.e. process management), we also model workload effects on this module. In most cases, the

workflow is distributed across an enterprise's intranet, thus system communication is another important factor for consideration. Finally, software engineers must consider the operating overhead of the underlying services and their variability (i.e. service processing). We consider web service workflow systems where intelligent software agents manage the control points. Intelligent agents are an abstraction for system control, but the use of agents are not mandatory to the effectiveness of this approach. Component, modules, or widgets could also be used in the place of agents. Figure 2 illustrates these categories in the context of a centralized agent-based system. We implement the web services workflow system as a multi-agent system because the use of intelligent agents facilitates the distribution of control required in a service-oriented environment [14]. Job requests are received by the workflow agent in the service-based modules. The workflow agent then considers process management and system communication costs associated with the job requests. Together with the service agents requests are made to different web services as needed and the results are coordinated.

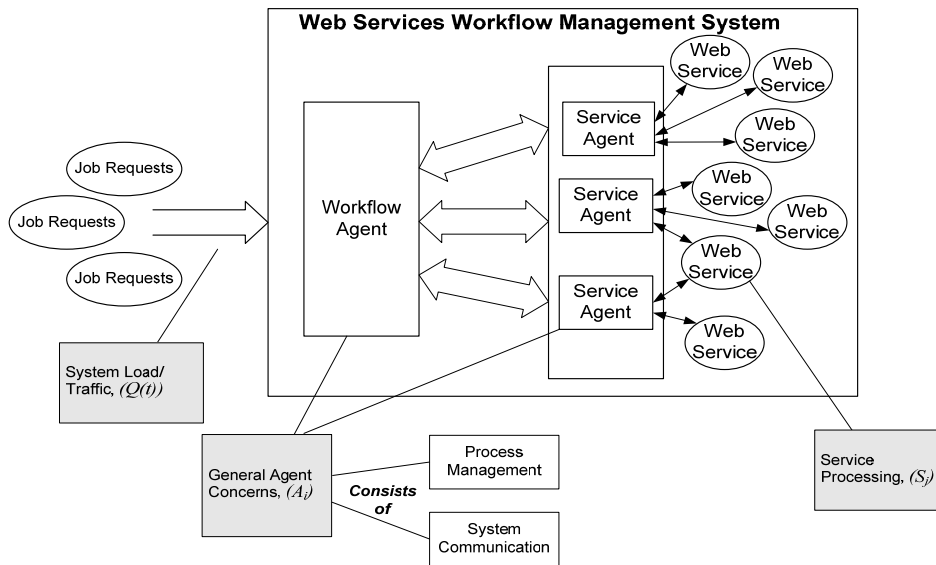


Fig 2. Selected Evaluative Criteria in the SOC Domain.

Process management and system communication are both handled by agents in this realization throughout the workflow process. These categories become the evaluation factors that are the basis for defining our performance model. The performance model will characterize a business process in terms of sub-process delays with respect to the agent architecture. The evaluation factors shown in Figure 2 are further illustrated in the Table 1.

**Table 1.** Summary of Evaluative Categories for an Agent-Based Web Services Workflow System.

Evaluation Category	Specific Factors	Notation
Agents ( $\mathcal{A}$ )	Data collection	$d$
	Agent-to-agent communication	$a$
	Task execution	$e$
Service ( $\mathcal{S}$ )	Service proximity	$p$
	Service execution	$s$
	Connection speed	$c$
System Load/Traffic ( $Q$ )	Average nbr. of agents per process	$\lambda_{\mathcal{A}}$
	Average nbr. of services	$\lambda_{\mathcal{S}}$
	Average nbr. of service changes	$\lambda_{SC}$

In order to approximate the cost associated with the business process, we first define the delays associated with each step of the process. These delays can be broken down into two categories, agent delays and service delays. The costs associated with agents include *data collection costs*, *agent communication costs*, and *task execution costs*. The data collection cost is the time required to get necessary data from a database (i.e. internal agent collective database or UDDI registry). The agent communication cost is the time it takes to communicate to other agents. The task execution cost is the time needed to complete assigned work. Using these parameters, the overall agent cost,  $\mathcal{A}$ , i.e. agent processing time, associated with a single business process can be expressed as follows:

$$A = \sum_{i=1}^m \mathcal{A}_i = \sum_i^m d_i + a_i + e_i \quad (1)$$

Here  $\mathcal{A}_i$  is the cost of a single agent and  $m$  is the number of agents. As aforementioned, our model has two types of agents, workflow agents and service agents. Both workflow and service agents exhibit these costs although the specific amount of time required for the subtasks will vary based on agent type. Since the execution of tasks associated with workflow and service agents are independent of each other, we define  $\mu_{AW}$  to be the average workflow agent processing time and  $\mu_{AS}$  to be the average service agent processing time.

The costs associated with the web services include the service proximity, the service connection time and the service execution time. The service proximity cost is the time required to reach the web service. The service connection cost is the time needed to connect to the service once a request has been issued to use the service. Finally, the service execution time is the time necessary to complete the service task. The service costs associated with a single business process can be defined as follows:

$$S = \sum_{j=1}^n S_j = \sum_j^n p_j + s_j + c_j \quad (2)$$

Here  $S_j$  is the cost of a single service and  $n$  is the total number of services needed to complete the business process. We define  $\mu_{S_j}$  to be the average processing time for service  $S_j$ .

The overall cost of the business process considers not only the agent and service delays, but also the reliability of the agents, services, and network. We model reliability as random variables involving agents, services and network.

$$\mathbf{R}(t) = \mathbf{R}_A(t) + \mathbf{R}_S(t) + \mathbf{R}_N(t) \quad (3)$$

To determine the reliability of an agent, we measure the probability that all necessary agents execute at time  $t$ . The probability that a single agent executes is then  $p_{A_i}(t) = P(X_i(t) = \text{agent\_executes})$ . Recall that for a single business process,  $m$  agents are needed to complete the task. Since the agents are independent, we can approximate the probability that  $m$  agents execute at time  $t$  as

$$p_A(t) = \prod_i^m P(X_i(t) = \text{agent\_executes}) \quad (4)$$

In other words, we are determining the reliability of the agents by measuring the probability that  $m$  agents execute at time  $t$ ,  $R_A(t) = p_A(t)$ . Similarly, we determine the reliability of a service by measuring the probability that all necessary services execute at time  $t$ . The probability that a single service executes is then  $p_{S_j}(t) = P(Y_j(t) = \text{service\_executes})$ . For a single business process,  $n$  services are needed to complete the task, so we define the reliability of the services as follows:

$$\mathbf{R}_S(t) = p_S(t) = \prod_j^n P(Y_j(t) = \text{service\_executes}) \quad (5)$$

The overall cost of a single business process becomes:

$$\mathbf{BP}(t) = [\mathbf{R}_A(t) \times A + (1 - \mathbf{R}_A(t)) \sum_{k=1}^p A_k + \mathbf{R}_S(t) \times S + (1 - \mathbf{R}_S(t)) \sum_{l=1}^q S_l] \times \mathbf{R}_N(t) \quad (6)$$

where  $p$  is the number of times an agent fails and  $q$  is the number of times a service fails. Notice that  $(1-R_A(t))$  represents the probability that an agent will fail. Similarly,  $(1-R_S(t))$  represents the probability that a service will fail. The final reliability related cost is network reliability. In our analysis, network reliability remains constant. This assumption is reasonable since the focus of this research is on the business process-level analysis and not on the network analysis. However, these attributes are included within our formalization for completeness and use in future work.

In our simulation environment, at this point in investigations, the agents do not fail. Therefore, by combining the number of agents,  $\lambda_A$ , the number services,  $\lambda_S$ , and the number of service changes,  $\lambda_{SC}$ , the approximate cost of a single business process can be simplified as follows:

$$BP(t) = \mu_A \times \lambda_A + R_S(t) \times \mu_S \times \lambda_S + (1 - R_S) \times \mu_S \times \lambda_{SC} \quad (7)$$

In an actual web services environment, the workflow agents, service agents and services will be receiving many requests at the same time. Therefore, we need to consider queues. A single queue will exist for each workflow agent. This queue will be modeled having random arrivals and constant service time. Since the focus of the analysis is on the performance of the middleware and not the underlying services, varying the service times of the underlying web services does not affect the evaluation of the middleware. The arrival process is modeled as a Poisson process. This means that the probability density function for the inter-arrival time is exponential and the waiting time for a workflow agent queue is

$$f(t) = (\mu_{AW} - \lambda_{IN}) e^{-(\mu_{AW} - \lambda_{IN})t} \quad (8)$$

where  $\lambda_{IN}$  is the average arrival time of a request into the queue.

For our simulation environment, we make a number of assumptions that further simplifies the model presented in this section. We highlight them now and explain them in more detail in later sections. In our model, we assume that each service agent is associated with a single service. Although multiple agents can be associated with a single service within the supporting framework, this assumption greatly enhances the ability to do predictable side-by-side analysis. Since the service agent and the service are coupled, we maintain a single queue for each service agent / service combination. The queue model parallels the one described for the workflow agent – a Poisson arrival process and a constant service process. A different Poisson process generates arrivals to each queue.

#### 4. A Two-level Software Engineering Modeling Approach

In order to assist software engineers in analyzing and designing these systems, there must be integrated models that bridge traditional software design techniques and the

aforementioned service-oriented approaches. Using extensions to the Unified Modeling Language (UML) [8], the previously mentioned evaluative attributes are coupled with software modeling approaches for capturing the web services business process.

#### **4.1. Business Process Views and System Interaction Views**

When modeling web services-based processes integrating many systems, software engineers may have only high-level details for systems that reside outside of their system boundaries. In other cases, the software engineers may have specific knowledge of certain distributed systems in addition to systems local to their enterprise. Consequently, modeling approaches must support both design at a high-level and the more specific design when system specifications are more detailed. Our modeling approach supports the integration of these two degrees of detail. This two-level modeling approach consists of a business process view and a system interaction view as shown in Figure 3. Our modeling approach combines and enhances fundamental knowledge that exists in this area of modeling. The business process view is a high-level view that describes the workflow-oriented process based on *organizational entities*, *web services*, and *high-level interactions*. Alternatively, when software engineers have more detail and can further define system actions, they use the system interaction view. In the system interaction view, a software engineer can specify the *data collection*, *communication*, and workflow-oriented *task* actions that must occur for a specific system interaction. The system interaction model only covers a subset of possible actions. However, our modeling framework allows for incremental additions as necessary. In this model we use software agent-based methodology to illustrate that the workflow system will be implemented with distributed intelligence (i.e. workflow agents that manage the processes and service agents that act as proxy to web services). Decomposing the organizational entity into modular agents presents a crisp modeling approach; however this is not a requirement of this engineering approach as *agent* could easily be substituted with *component*.

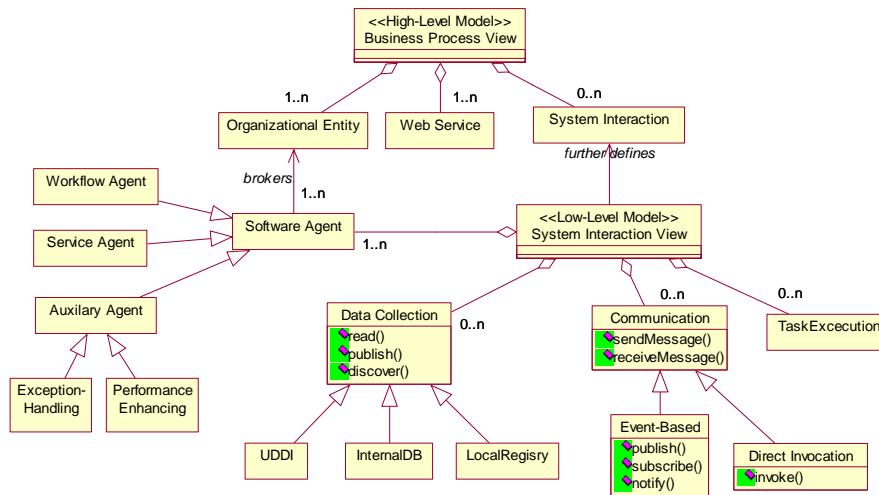


Fig. 3 SOC Software Design Meta-model.

#### 4.2. Sample Business Process and System Interaction Views

When modeling a new capability consisting of inter-organizational services, a software engineer must first model the high-level design as a business process view where interactions are common workflow routines. Using workflow patterns [19][20] is an effective way to describe complex process interactions. Furthermore many workflow patterns are also implemented in common web service interaction standards (e.g. BPEL4WS, WSFL, WSCI) [9][23]. Secondly, the software engineer would further define each system interaction (workflow pattern) with the specific implementation of the underlying agents using the system interaction view. Similar to earlier work in process modeling [11], business process views are modeled as UML activity diagrams. Organization entities are the roles that describe UML swimlanes, and web services are modeled using UML activities. System interactions are annotated in the UML activity diagram as associated with UML control flow notations, such as fork/join relations and initial/end state notations. Models are shown in Figure 4. Each system interaction in the business process view can be modeled in an independent UML interaction diagram (system interaction view). Although in this work, we use UML collaboration diagrams for the system interaction view, sequence diagrams are equally as effective. These views are consistent with UML 1.x semantics; however, in future work, these models will be enhanced with UML 2.0 notations (i.e. communication diagrams). By isolating system interactions in their own models, these models can represent patterns that can be reused in other business process views for other modeling scenarios. In Figure 4, there is an example of a business process view and associated system interaction view for a typical travel agency scenario.

This business process shows the sequence of services that perform a simple travel reservation scenario for purchasing an airline ticket, reserving a hotel room, reserving a rental car, and receiving an email-delivered itinerary. In this simple scenario, five workflow patterns can be annotated, which are illustrated by the stereotypes. In the first stereotype, the instance creation workflow pattern represents the execution of a new job request. Other stereotypes designated by <<SYNCHRONIZATION>>, <<PARALLEL SPLIT>> and <<SIMPLE-MERGE>> represent the precedent requirements (synchronization), the concurrency requirements (parallel split), and the merging requirements workflow patterns needed for the travel reservation scenario.

The system interaction view describes the step-by-step lower-level actions involved in the workflow patterns. In Figure 4, the lower-level actions define the instance creation workflow pattern. In this view, the workflow agent is notified of a new job creation event, the agent queries its internal data repository to find the schema of the business process. The workflow agent then creates a unique identification for the instance (createInstance). Once the instance is created, the agent subscribes for future (nonfunctional) events, such as error events or completion events. Finally, the workflow agent initiates the workflow by publishing an action event to the appropriate service agent. This system interaction view is just one of numerous approaches to instance creation. Our focus is not the specific view, but rather our flexible approach that allows the software designer to create multiple interactions and evaluate them with respect to the business process.

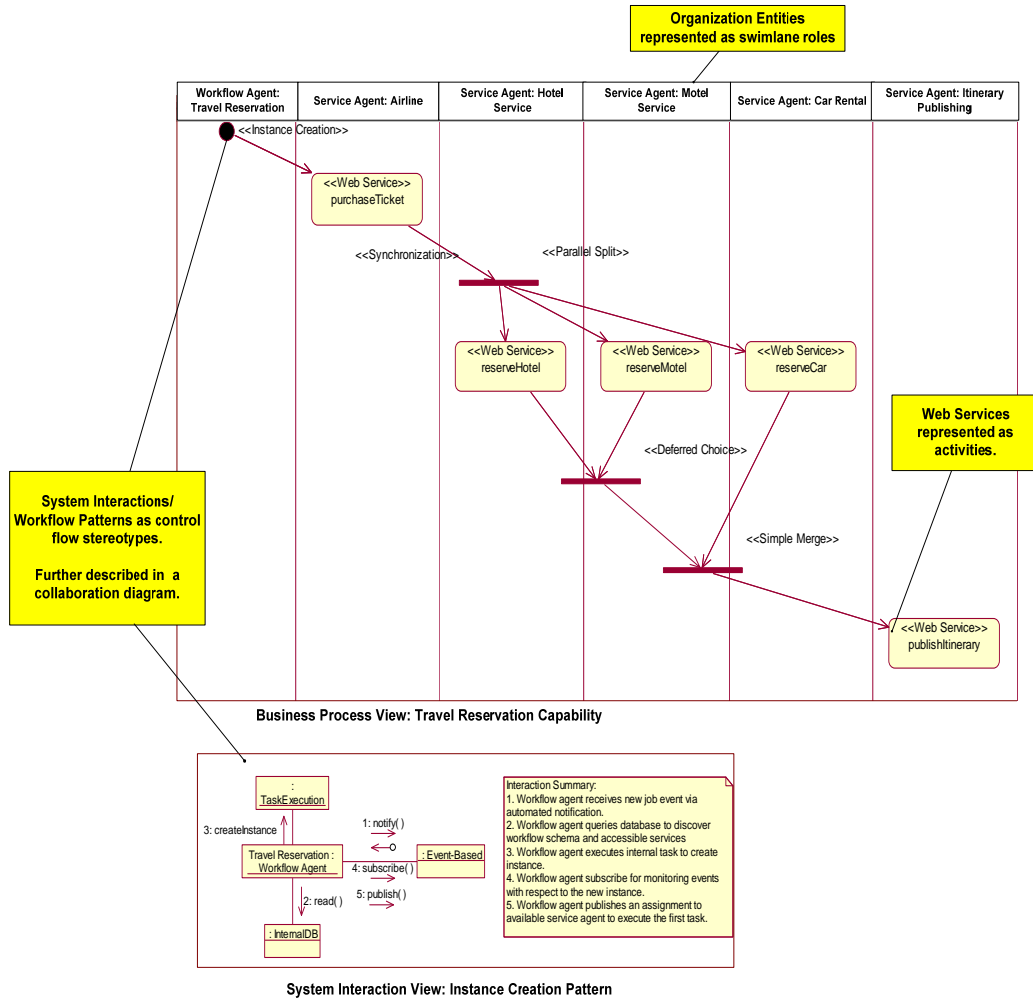


Fig. 4. UML Modeling for Workflow-Oriented SOC Software Design.

### 4.3. Visual Models with Evaluative Attributes

In the previous section, the business process view and the system interaction view were described with respect to functional modeling. In order to truly evaluate the efficiency of a proposed system, the software engineer must annotate those views with respect to nonfunctional concerns. In this paper, we concentrate on performance and operational information. We will now show how the annotated performance information are used for semi-automatic system analysis.

The intention of the modeling approaches in this paper is to leverage existing industry-standard approaches as opposed to creating new ones. As such, with slight customizations to their usage and the addition of a service-oriented middleware domain-

specific model, we leverage standard activity and collaboration diagrams for constructing the business process and the system interaction views. With respect to performance modeling, we also leverage existing approaches. The *UML Profile of Schedulability, Performance, and Time (UPSPT)* [18] is an existing framework for annotating scheduling, performance, and time information on functional models. For our approach, the performance modeling profile is most relevant. Figure 5 shows a re-creation of the UPSPT with extensions proposed in this work. Several extensions were added for completeness, but some extensions (i.e. reliability) have not been incorporated into our simulation at this time.

The UPSPT has close relations to the notion of agent-mediated service-oriented computing presented in this work. The **Workload** concept is similar to our notion of system load. A **Closed Workload** represents a closed population of actors to the system. In a **Closed Workload**, an actor exercises the system then experiences a delay before accessing the system again. Our approach is more closely related to an **Open Workload** where there is an open number of stakeholders with variable distributions of load. Three attributes relevant to our approach that we add to the model are the *number of services*, *number of agents*, *number of changes to service bindings*, and the reliability of the network and components in the system. The **PScenario** is similar to the concept of a workflow schema where each underlying **PStep** mirrors the realization of web services. We extend this model with the addition of the service-oriented attributes of *proximity*, *reliability*, and *connectionSpeed*. Each of the aforementioned attributes is a numerical measure describing a particular web service. Finally, the **PResource** further specified as **PProcessingResource** and **PPassiveResource** are used to realize agents. **PPassiveResources** represent system components that mainly execute in parallel to functional processing; however, their existence affects the system as a whole. In the future, we plan to model nonfunctional agents, such as performance-enhancing and error-handling agents (Figure 3). Currently, our modeling approach focuses on service agents as **PProcessingResources**. We extend **PProcessingResources** with the addition of the *agent-specific attributes of execution time, data management time, and communication time*.

Figure 6 shows an example of the instance creation pattern, as shown in the system interaction view of Figure 4, annotated with performance information. This example demonstrates how UPSPT can incorporate performance values. Performance values are captured with the UML note notation. The UML note is annotated with a stereotype that represents the associated object as defined in the UPSPT meta-model (shown in Figure 5). In Figure 6, the **PAOpenWorkload** object is associated with the flow of job requests to the Workflow Agent. This object further specifies that the occurrencePattern is represented using an estimated, mean distribution of Poisson traffic.

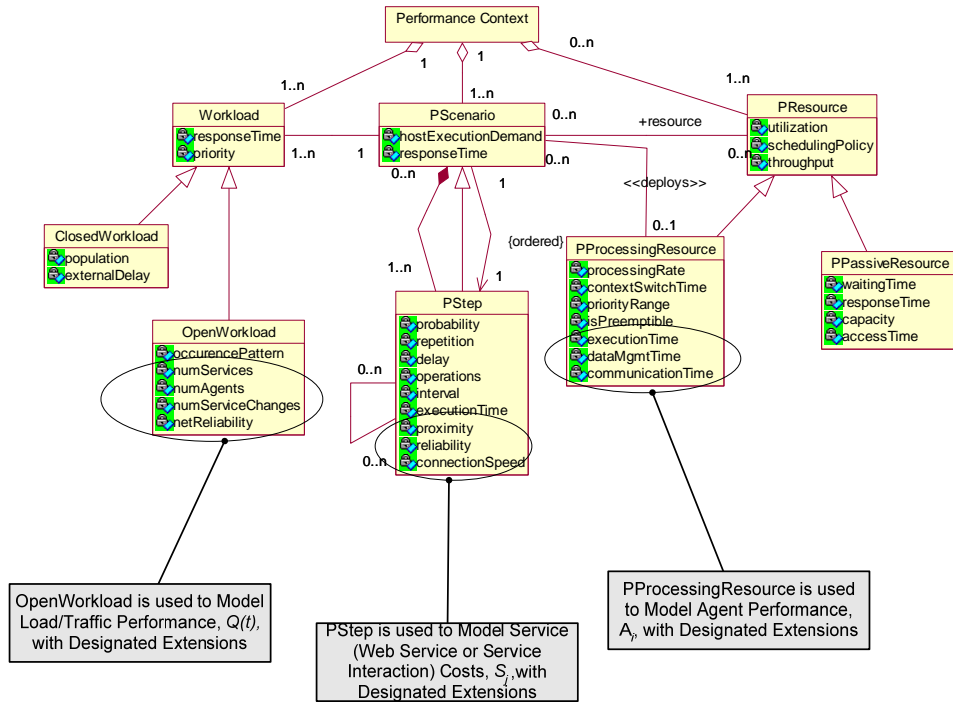


Fig 5. UML Profile of Schedulability, Performance, and Time with New Extensions for Service-Oriented Computing.

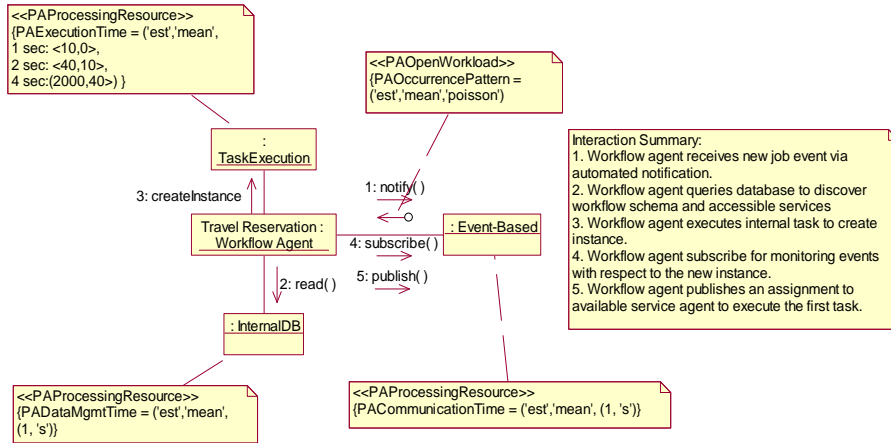


Fig 6. Adding Evaluative Attributes to the Instance Creation Interaction.

The Workflow Agent is also annotated with execution time specifications. Again using an estimated mean, the execution time of the agent varies depending on the range of the queue size, e.g. 1 second when the queue contains between 0 and 10 requests, 4 seconds

when the queue is between 40 and 2000 requests. The reader should note that all performance values are specified using 3 parameters, the source of the value (e.g. required, assumed, predicted, measured, and estimated), the type of value (e.g. average, sigma, kth-moment, and max) and the actual value with units. For further discussion on describing performance values using UPSPT the reader is directed to [18].

### 5. Simulation-Based Model Evaluation

This approach uses a systematic process of extracting information from views into equations that subsequently configure the simulation software. Business process views and system interaction views can be decomposed into simplified lists representing the underlying information in the three categories defined in Figure 5, agents,  $A_i$ , services,  $S_j$ , and system load/traffic,  $Q(t)$ . As an example, Table 2 shows how information is organized in the COACHES simulation after it is extracted from the system interaction view of Figure 6. The response and variability of the services,  $S_j$ , are not considered in this system interaction since the focus is on the interaction between workflow agents and service agents. The agent performance measures were extracted from an earlier agent infrastructure developed in previous work [5]. The magnitudes were kept consistent with the real measures, but for simplicity the measures were rounded to whole numbers. After running the corresponding simulation software, several estimations can be yielded (e.g. average overall delay of the business process, average overall delay of the business process per simulation cycle, average delay of workflow agents (per cycle), and average delay of service agents (per cycle)).

**Table 2.** Performance Information from Modeling Views.

<b>Description</b>	<b>Performance Factor</b>	$A_i$ (Sec)	$S_j$	$Q(t)$
Incoming Requests	Number of Services	-----	--	2000
Incoming Distribution	Occurrence Pattern	-----	--	poisson
Receive Job	Communication	1	--	-----
Read Schema	Data Collection	1	--	-----
Create Instance	Task Execution	$1 + delay$	--	-----
Subscribe	Communication	1	--	-----
Publish Events	Agent-to-Agent Communication	1	--	-----

## 6. CASE STUDY 1: Model Evaluation of Centralized vs. P2P Control

An experiment was conducted to analyze the effect on the instance creation interactions with different distributions of incoming business process requests. The experiment varied two system interactions for instance creation, one interaction where a third party workflow agent instantiates the process and another interaction where the process initiation occurs among the service agents (peer-to-peer (P2P) among the agents where the first Service Agent or Lead Service Agents creates the instance). Figure 7 gives an overview of the two approaches and the interaction views.

In the P2P-style, there is a static delay for Lead Service Agent of 4 seconds while the queuing delay can vary between 2 and 5 additional seconds. In the 3rd party control-style, the workflow agent has a static delay of 5 seconds but the additional delay only varies between 1 and 4 additional seconds. Four types of traffic were chosen, each evenly spaced over 100 cycles. The four types of traffic are, 10 Job Requests every 10 cycles (Batch 10), 2 Job Requests every 2 Cycles (Batch 2), Starting from 1 increasing the amount of jobs by 1 every 7 cycles (Increasing), and 1 Job Request per Cycle (Continuous).

These experiments were executed on a 1.7 Gigahertz, 1 Gigabyte Dell Workstation. The experiment was based on relatively homogeneous distribution of requests (i.e. business processes containing the same type of underlying service). In these experiments, the third-party control interaction performs more favorably than the P2P interaction for instance creation. Since the queue size was predominantly higher than 20 outstanding tasks, the P2P interaction has the maximum delay of 9 seconds which is higher than the maximum delay of the third-party control interaction (6 seconds) at the same queue size. The queue size would have to consistently remain under 20 outstanding tasks for the two interactions to perform equivalently or the P2P interaction to perform better. The experiment also has led to other interesting results. Although the best choice of traffic (Batch10) is the same for both P2P and 3rd party control, the worst choice of traffic differs between the two interactions. This result supports the notion that variations in traffic can lead to unpredictable results. Performance variability and/or deterioration under various conditions are sometimes underestimated by business process modelers and software engineers. For both scenarios, the queue size varies as the distribution of the network traffic. Other operational concerns, such as evaluating system memory capacity and/or CPU utilization (not discussed here), can be a function of the queue size. The ability to predict average queue size by evaluating software modeling approaches can easily address these additional operational concerns and further support software engineers.

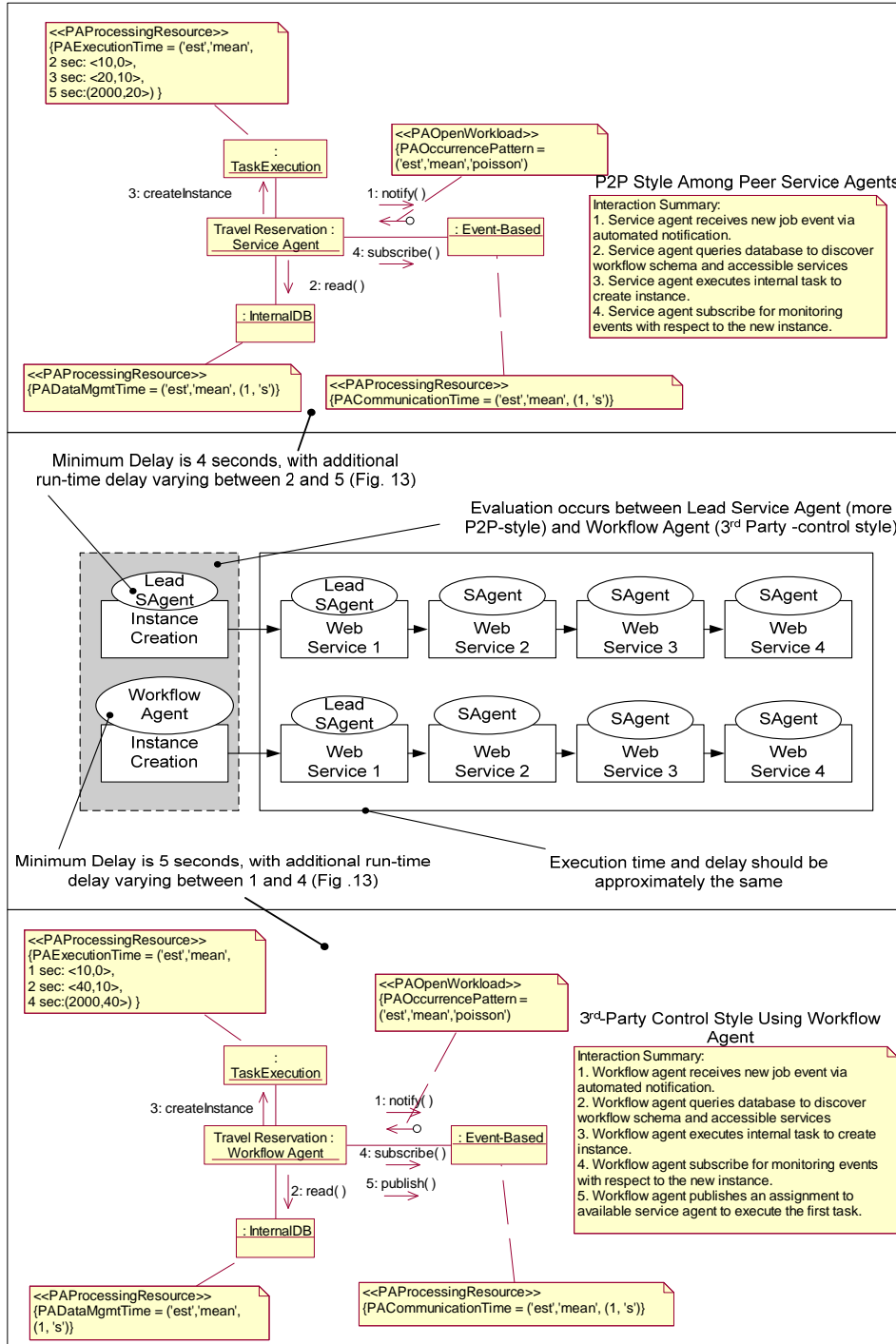
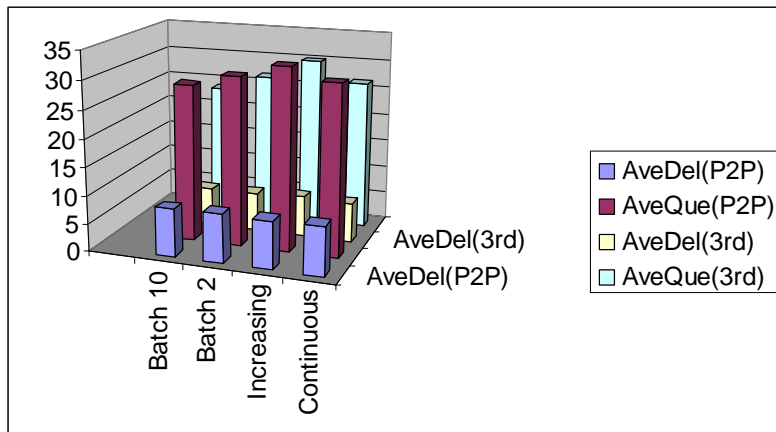


Fig 7. Two Instance Creation Implementations.

**Table 3.** Numerical Results for Model Evaluation.

Traffic Type	AveDel (P2P)	AveQue (P2P)	AveDel (3 <sup>rd</sup> Party)	AveQue (3 <sup>rd</sup> Party)
Batch 10	8.49495	28.0471	6.89899	23.7374
Batch 2	8.65333	30.1667	6.96333	26.6228
Increasing	8.51736	32.3576	7.32292	30.0799
Continuous	8.68122	30.2967	6.96467	26.6514

**Fig. 8.** Simulation Results of Model Evaluation.

## 7. CASE STUDY 2: Modeling Service Responsiveness Considering UDDI

In another experiment, that extends other previous work [7], business process interactions incorporating UDDI technologies were investigated using this software engineering approach. We analyzed several operational modes for using UDDI technologies to validate the responsiveness of pre-existing service bindings at run-time. When new job requests are received, an organization can reaffirm the responsiveness of all of the services prior to initiating the business process. Alternatively, the organization can choose to allow the connection to fail at run-time then re-bind to another service, as needed. We acknowledge that there are numerous variations to these operational modes, but since these two modes represent two general notions, they are the focus of this experimentation. Rebinding services is directly related to the *OpenWorkload* concept and *numServiceChanges* attribute as shown in our enhanced UML profile in Figure 5. Similarly, responsiveness is measured directly by the *executionTime* attributed in the *PProcessingResource*. As such this experiments considers variations in request traffic in

addition to the variations in the request of services that no longer exist and must be replaced.

There are two general operational modes for incorporating UDDI into service-oriented business process management systems with respect to this type of reliability. They are illustrated in Figure 9. The first general mode of operation is to configure the system to check all of the underlying services once a new job is received (*pre-process validation*). For the pre-process validation standpoint, the system follows the steps 1, 1a, and 2 in Figure 9. In step 1, a consumer requests a new job and the provider captures the job and starts a new business process. In step 1a, the provider confirms that all current services for that business process are still viable. One assumption made in this work is that a service listed and valid in the provider's UDDI at the beginning of the job remains valid until the service is executed. The purpose of this work is not toward the discovery of new services, but just the validation of pre-established services. Finally, step 2 is the enactment of the service based on the specific business process.

Instead of preemptively assuring the viability of all services, the second mode is triggered via internal failures. The operational mode for *connection-time validation* accesses the UDDI registry for services that have problems (i.e. after a connection error). This approach minimizes the need to search for the address of every service, but only services that have changed locations. However, the service time for an individual service is increased because the new time must include the time required to determine that a connection error has occurred. The process for connection-time validation is also illustrated in Figure 9, but the application steps are 1, 2, and 2a. In the consideration of space, the business process and system interaction views are not captured, instead results are explained.

In evaluating the aforementioned operational modes, each of modes were exercised by varying three different aspects, traffic composition, percentage of service changes, and connection failure time. In the experiments, we considered a finite number of job requests (i.e. 100 job requests) independent of the traffic composition. Considering this is a *notional* system, we chose a relatively small number of job requests to simplify the analysis; however the simulation can support any number of requests. The traffic composition was varied such that the first traffic scheme was consistently light and steady. In this scheme, 1 job request is transmitted for each time cycle. Another traffic scheme had a relatively high level of concurrency. Ten requests were transmitted at once at intervals that spanned the length of the experiment. The final traffic was delivered using a Poisson distribution.

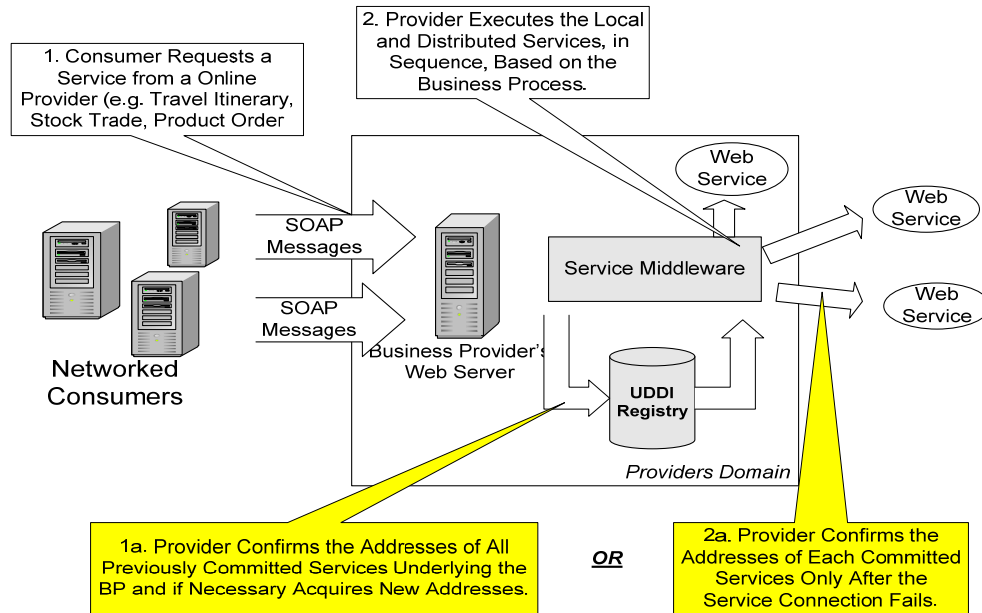


Fig 9. Two models of Operation Incorporating UDDI Approaches.

The percentage of service changes were varied independently. The overall percentages of services that change throughout the entire simulation were varied at 33%, 66%, and 100% (e.g. at 100%, all services change and must be re-bound each time). Considering the notion of service-oriented computing is relatively new, we are unaware of any study that surveys the frequency of services to change. This is perhaps the first framework that considers the potential dynamics of web services in this new environment. Finally, the connection failure time was also varied. The connection failure time was varied relative to the UDDI service time. The connection time was varied to be 25%, 50% and 100% of UDDI service time. Similar to the lack of studies in web services dynamism; there are no related studies that measure the ratio of connection time to service time.

Considering the variations, both operational modes were simulated in five sets of experiments. In this paper, we do not present all of the numerical findings since the actual numbers would vary depending on the specification of the machine. However, in Table 4, 5, and 6, we summarize the findings, generally comparing the merits of both operational modes. The cells in the table denoted as *CT* represent that the connection-time validation mode has the lowest average service time per business process. When *PP* is denoted the pre-process validation mode has the lowest average service time. *PP/CT* represents the fact that the service times have negligible difference. Figure 10 shows a graphical representation of how the results are used as a decision matrix for software engineers in this domain.

This experiment led to several conclusions. The factor that has the most impact on the choice of modes is the percentage of services changes. In all tests, when the percentage

of service changes was less than 59%, the connection-time validation was most efficient. Although the exact percentage varied, typically, when the service changes were between 59% and 70% of all services, the pre-process validation mode was most efficient in completing business processes. In all experimental cases, the pre-process validation mode was more efficient when the percentage of changes were greater than 70%. The experimentation also suggests that steady traffic is more beneficial to the connection-time mode than highly concurrent and Poisson traffic. With steady traffic, the connection-time mode has reduced concurrency across processes. Steady or concurrent traffic does not significantly improve the pre-process validation mode because, in this mode, the service requests are sent concurrently regardless of the traffic composition. Finally, when the connection failure time is high, the percentage of service changes required to make the pre-process validation mode more efficient than the connection-time validation mode is reduced.

**Table 4.** Connection Failure Time = 25% of UDDI Search Time.

	<i>Steady</i>	<i>Heavily Concurrent</i>	<i>Poisson</i>
<i>SC 33%</i>	CT	CT	CT
<i>SC 66%</i>	CT	CT	PP
<i>SC 100%</i>	PP	PP	PP

**Table 5.** Connection Failure Time = 50% of UDDI Search Time.

	<i>Steady</i>	<i>Heavily Concurrent</i>	<i>Poisson</i>
<i>SC 33%</i>	CT	CT	CT
<i>SC 66%</i>	CT	PP/CT	PP
<i>SC 100%</i>	PP	PP	PP

**Table 6.** Connection Failure Time =100% of UDDI Search Time.

	<i>Steady</i>	<i>Heavily Concurrent</i>	<i>Poisson</i>
<i>SC 33%</i>	CT	CT	CT
<i>SC 66%</i>	PP/CT	PP	PP
<i>SC 100%</i>	PP	PP	PP

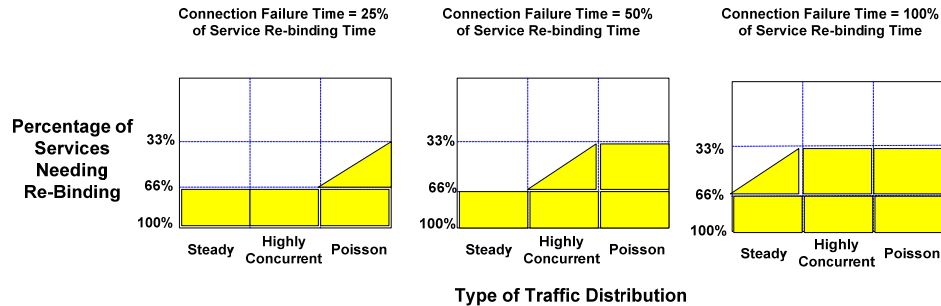


Fig 10. Decision Matrix Generated from Simulation Results (Shaded quadrants represent areas where pre-process validation should be instituted)

## 8. Related Work

Although the simulation approaches in this paper are similar to related work considering QoS for distributed workflows, the major innovation in our work is the combination of standard software design methods with formal modeling techniques and QoS simulation. Exemplary projects that focus mainly on the QoS in this domain can be found in the work of [3][24] who also investigates centralized control versus P2P. Cardoso [10] considers QoS in workflow by experimenting with factors, such as time, cost, reliability, and fidelity. Liu [15] et. al. also focus on web service composition, but not on execution and costs. Both Grundy [13] and Zeng [25] investigate QoS with regards to service composition, however unlike our approach, there do not emphasize the benefits of an iterative process. Zeng introduces state chart diagrams to model performance and other nonfunctional attributes. An innovation in our work is the separation of concerns, whereas there is a modeling hierarchy where business models can be separated from system models. Alternatively, Liu attempts to minimize the aggregate data-communication cost among services. There are recent projects that use software modeling approaches for service composition. Orriëns et. al. [16] and Skogan et.al. [17] use UML to model web service composition. Our approach extends these approaches by adopting a customization of UML that embeds performance information. In addition, our approach uses a stand-alone simulation application to evaluate the UML models. This simulation application can be configured to evaluate a large number of service composition environments by altering the software models. In addition, the application fits seamlessly within a software design and modeling framework based on industry-standard software lifecycle approaches. Moreover, another major enhancement is that our work considers the dynamics surrounding the nature of the Internet.

## 9. Discussion

The main contribution of this work is two-fold. First, we introduce a principled software engineering approach for evaluating systems that execute processes consisting of web services. This software engineering approach enables the incorporation of bottom-up knowledge (i.e. existing web services) with top-down knowledge (i.e. required business

capability). In addition this approach allows constraints stemming from the dynamism of the Internet (i.e. service changes, responsiveness, network conditions) to be factored into the analysis. Second, within the framework of this approach, we develop a performance model that analyzes responsiveness and execution costs.

A formal model for web services-based workflow processing was introduced. This model incorporates several factors critical to analyzing systems in this domain. In experimentation, the results demonstrate the complexity of decision-making when varying these system factors. These experiments have shown that the designated factors have significant importance in designing a system composed of distributed web services. In one case, slight changes to the traffic distribution have had measurable impacts on the choice of validation schemes.

Further, the new formal model was integrated with standard UML-based design techniques (i.e. UPSPT) and a new simulation tool for model evaluation. Experimentation shows that this new software process and supporting tools can support software system analysis at the high-level business level as well as the lower-level implementation level.

This work is related to the larger area of agent-oriented software engineering [1][12] and has significant implications if web services are to be converted into intelligent agents as several researchers predict. The current state-of-the-art in agent-oriented software engineering focuses on developing software engineering techniques to build agent-based systems. In this work, the premise is slightly different. The goal is to incorporate agent development into existing software development environments and paradigms. Although, many researchers in the areas of web service composition, process management, and workflow investigate techniques to realize service-oriented computing from the consumer-planning standpoint, the innovation of this work is in the evaluation of such systems once they are in place. In addition, this approach addresses the impact that external conditions related to the Internet may have on success of service-oriented computing.

In future work, a real operational web-services environment in the electronic commerce domain will be evaluated with software engineering approaches presented here. These real world experiments will facilitate further validation of the simulation software and modeling techniques within a complete operational environment as opposed to the various testing environments. Furthermore, the notion of service dynamics is defined as *service changes* in this work. In the future, a richer model of service dynamics must be investigated and analyzed. As such, we plan to enhance the model and then run several new case studies that evaluates our approach at each step of the lifecycle.

### **Acknowledgments**

Amy L. Sliva provided development support to the analysis performed in the second case study. The second case study was benefited by conversations with Michael zur Muehlen and Jeffrey V. Nickerson of Stevens Institute of Technology.

## References

- [1] Agent-Oriented Software Engineering (AOSE) (2003): <http://www.csc.liv.ac.uk/~mjw/aose/>
- [2] Benatallah, B., Dumas, M., and Sheng, O.Z. (2005) Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Databases* 15(1):5-37, January 2005. Kluwer Academic Publishers.
- [3] Benatallah, B., Dumas, M., Sheng, Q., and Ngu, A. (2002) Declarative composition and peer-to-peer provisioning of dynamic web services. In 18th International Conference on Data Engineering., February 2002
- [4] Blake, M.B. (2003) "Coordinating Multiple Agents for Workflow-Oriented Process Orchestration", *Information Systems and E-Business Management*, Eds. J. Becker and M. Shaw, Vol. 1, No. 2, pp 387-405, December 2003, Springer-Verlag
- [5] Blake, M.B. (2003) "Evaluating Agent-to-Agent Workflow Interactions for Service Composition: Third-Party Control or P2P?" Georgetown University, Tech Report CSTR-20030420-5, 2003
- [6] Blake, M.B. (2006) A Lightweight Software Design Process for Web Services Workflows, *Proceedings of the 2006 International Conference on Web Services*, Chicago, IL
- [7] Blake, M.B., Sliva, A.L., zur Muehlen, M, and Nickerson, J. (2007), Binding Now or Binding Later: The Performance of UDDI Registries", *Proceedings of the IEEE Hawaii International Conference of System Sciences (HICSS-2007)*, Track on Technology and Strategies for Realizing Service-oriented Architectures with Web services
- [8] Booch, G. Rumbaugh, J, and Jacobson, I. (1999) "The Unified Modeling Language User Guide", Addison Wesley, Reading, MA 1999
- [9] BPEL4WS (2006): <http://www.ebpm.org/bpel4ws.htm>
- [10] Cardoso, J. (2002) "Quality of Service and Semantic Composition of Workflows," PhD thesis, Univ. of Georgia, 2002
- [11] Dumas, M. and ter Hofstede, A.H.M. (2001) "UML Activity Diagrams as Workflow Specification Language", *The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, 4th International Conference, Toronto, Canada, October 1-5, 2001, *Proceedings. Lecture Notes in Computer Science* 2185 Springer 2001, ISBN 3-540-42667-1
- [12] Gianchiglia, F., Mylopoulos, J. and Perini, A. (2002) "The Tropos Software Development Methodology: Process, Models and Diagrams" In *Proceedings of the 3<sup>rd</sup> International Workshop on Agent-Oriented Software Engineering (AOSE2002)* in conjunction with AAMAS2002, Italy
- [13] Grundy, J.C., Hosking, J.G., Li, L. And Liu, N. Performance engineering of service compositions, *ICSE 2006 Workshop on Service-oriented Software Engineering*, Shanghai, May 2006, ACM Press
- [14] Huhns, M.N. *Software Agents: The Future of Web Services* (2003). In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, *Lecture Notes of Computer Science*, 2592 Springer, pp. 1-18
- [15] Liu, D., Peng, J., Law, K.H., and Wiederhold, G. (2004) "Efficient Integration of Web Services with Distributed Data Flow and Active Mediation" *International Conference on Electronic Commerce*, Delft, Netherlands, 2004
- [16] Orriens, B., Yang, J., Papazoglou, M.P. (2003) Model driven service composition. *Proceedings of the First International Conference on Service Oriented Computing*, Trento, Italy, December 15-18, 2003
- [17] Skogan, D., Grønmo, R., and Solheim, I. (2004) "Web Service Composition in UML", In *Proceedings of Enterprise Distributed Object Computing Conference*, Eighth IEEE International (EDOC'04), September 20 - 24, 2004 Monterey, California

- [18] UML® Profile for Schedulability, Performance, and Time (UPSPT), version 1.0 (2007): <http://www.omg.org/technology/documents/formal/schedulability.htm>
- [19] van der Aalst, W.M.P. (2003) “Don’t go with the flow: Web Services composition standards exposed”, *IEEE Intelligent*, February 2003
- [20] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P. *Workflow Patterns*. QUT Technical report. FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002)
- [21] *Web Services* (2007) <http://www.w3.org/2002/ws/desc/>
- [22] *WSDL* (2007) <http://www.w3.org/TR/wsdl>
- [23] *WSFL* (2007): <http://www.ebpml.org/wsfl.htm>
- [24] Zeng, L., Ngu, A. Benatallah, B., O’Dell, M. (2001) An agent-based approach for supporting cross-enterprise workflows. In *Proceedings of the 12th Australasian Conference on Database Technologies*, 123-130, Queensland, Australia 2001
- [25] Zeng, L., Benatallah, B., Ngu, A. H.H., Dumas, M., Kalagnanam, J., and Chang, H.. QoS-aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering* 30(5):311-327, May 2004, IEEE Computer Society