

A Lightweight Software Design Process for Web Services Workflows

M. Brian Blake
Georgetown University
Washington, DC, 20057
blakeb@cs.georgetown.edu

Abstract

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.

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) [15][16].

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

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 Section 3, we discuss the design of the supporting simulation software. In Section 4, we detail and evaluate an experimental case. In the final sections, we discuss the merits of this work with regards to related work and future investigations.

2. 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 Sections 2.2 and 2.3. The software engineer models these views consisting of the best set of known information while incorporating feedback from the stakeholders.

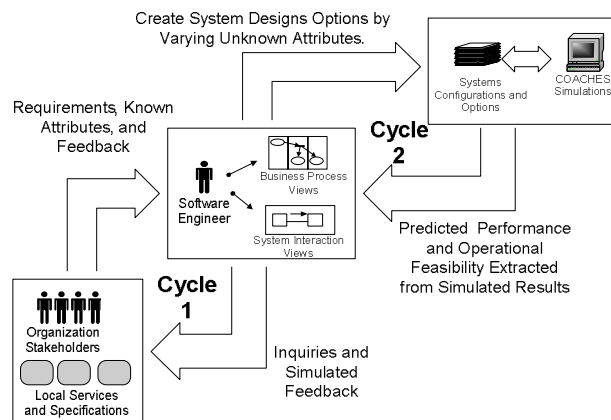


Figure 1. Iterative SOC design process.

In the second cycle, as the software engineer feels comfortable that the views are developed well enough with respect to 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 [3], 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.

2.1. 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, system communication, process management, and service processing. System load/traffic involves the rate at which the middleware receives requests. Considering the presence of a middleware workflow application that constructs processes from job requests (i.e. process management), we also model workload effects on this module. In most cases, the middleware 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). Figure 2 illustrates these categories in the context of a centralized middleware component. Job requests are received and handled by the middleware modules, such that requests are made to different web services as needed and the results are coordinated.

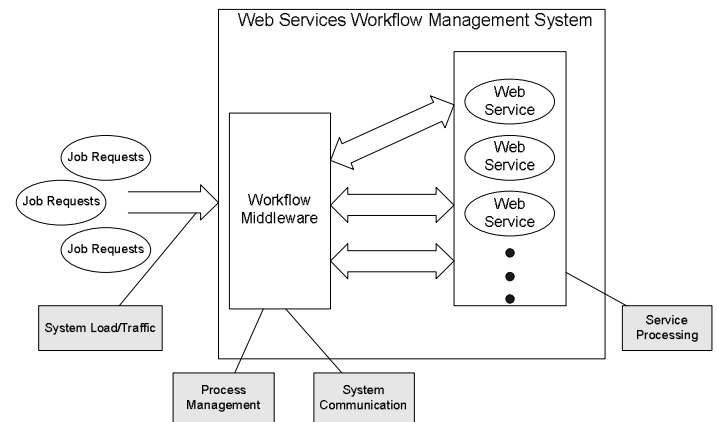


Figure 2. Selected Evaluative Criteria in the SOC Domain.

2.2. Two-level 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) [5], the previously mentioned evaluative attributes are coupled with software modeling approaches for capturing the web services business process.

When modeling distributed web services-based processes consisting of many systems including their own, 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 system 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 and others may exist. However, our modeling framework is easily extendable and 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*.

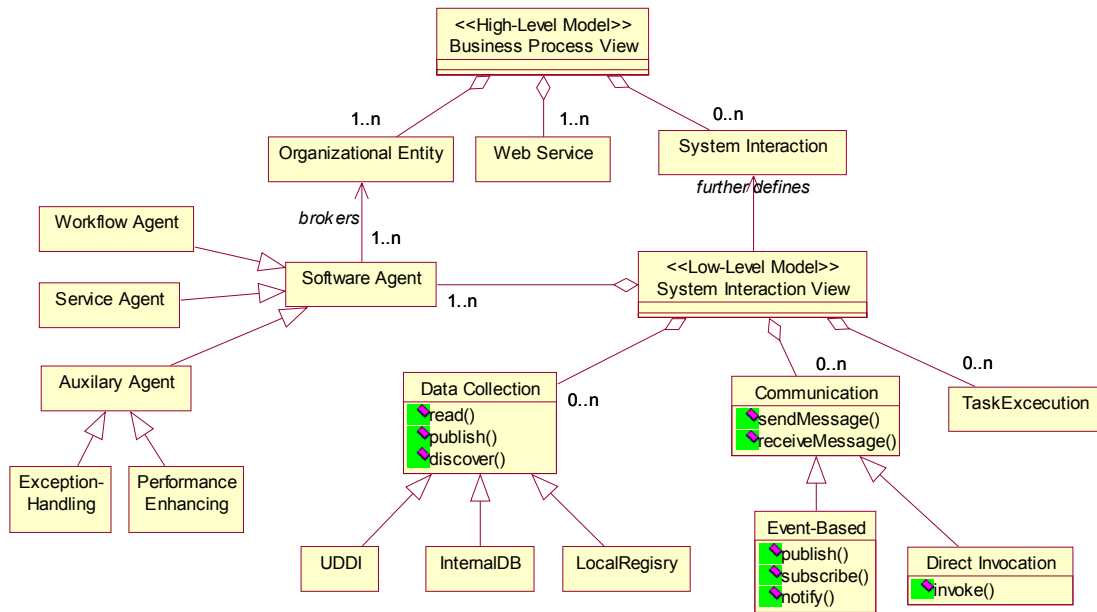


Figure 3. SOC Software Design Meta-model.

2.3. 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 system interactions are common workflow routines. Using workflow patterns [13][14] 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 [6][17]). 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 [8], 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.

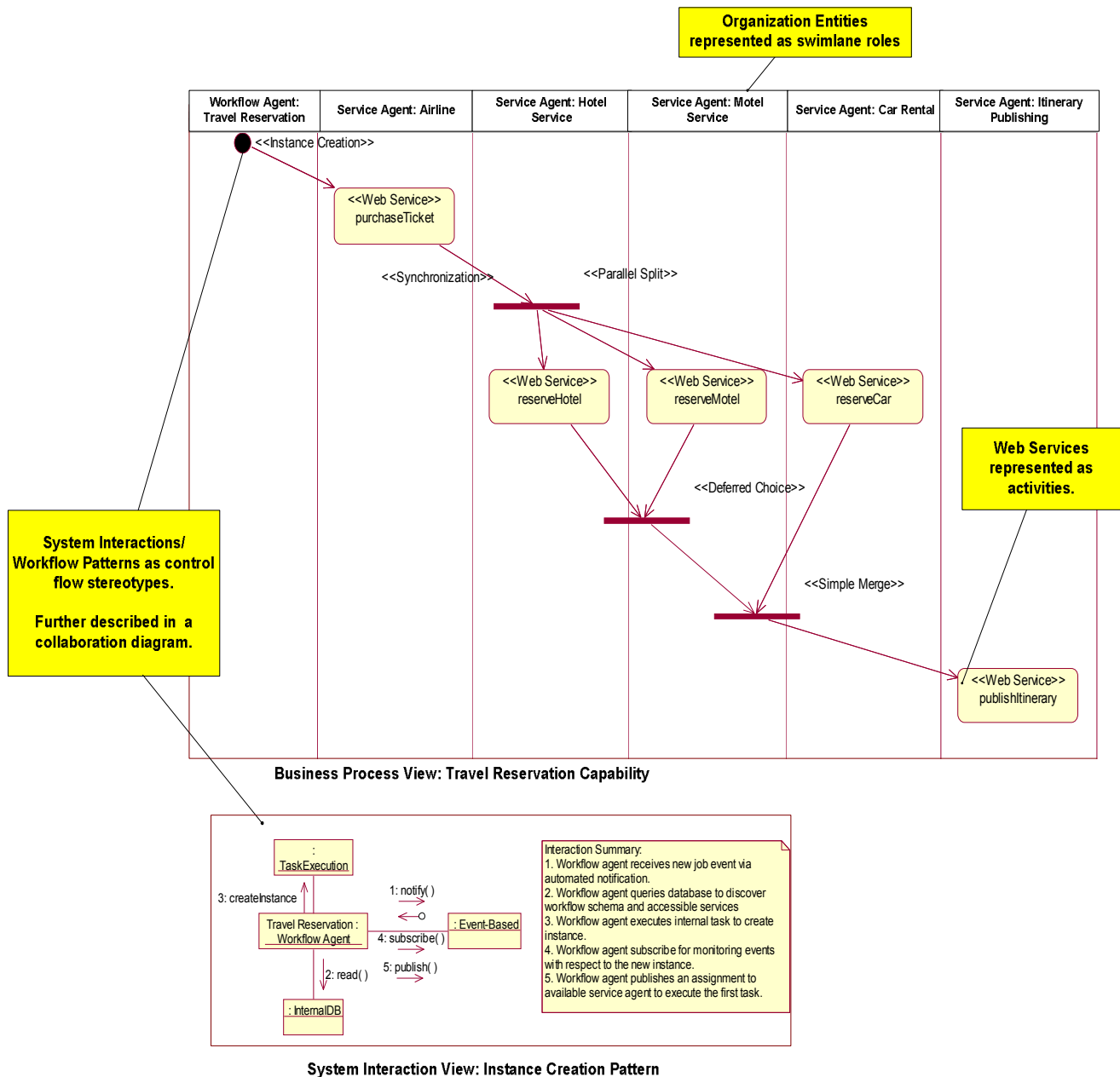


Figure 4. UML Modeling for Workflow-Oriented SOC Software Design.

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.

2.4. Annotating 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 as defined in Section 2.1. In the following section, we will show how the annotated performance information can be 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) [12] 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 recreation of the UPSPT with extensions proposed by the authors. We now summarize the approach as it relates to our formal performance model in Section 3 and briefly describe our extensions to the profile as it relates to the service-oriented computing domain.

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 Workflow, 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. 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, max, etc.) and the actual value with units. For further discussion on describing performance values using UPSPT the reader is directed to [12].

3. 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, $Q(t)$.

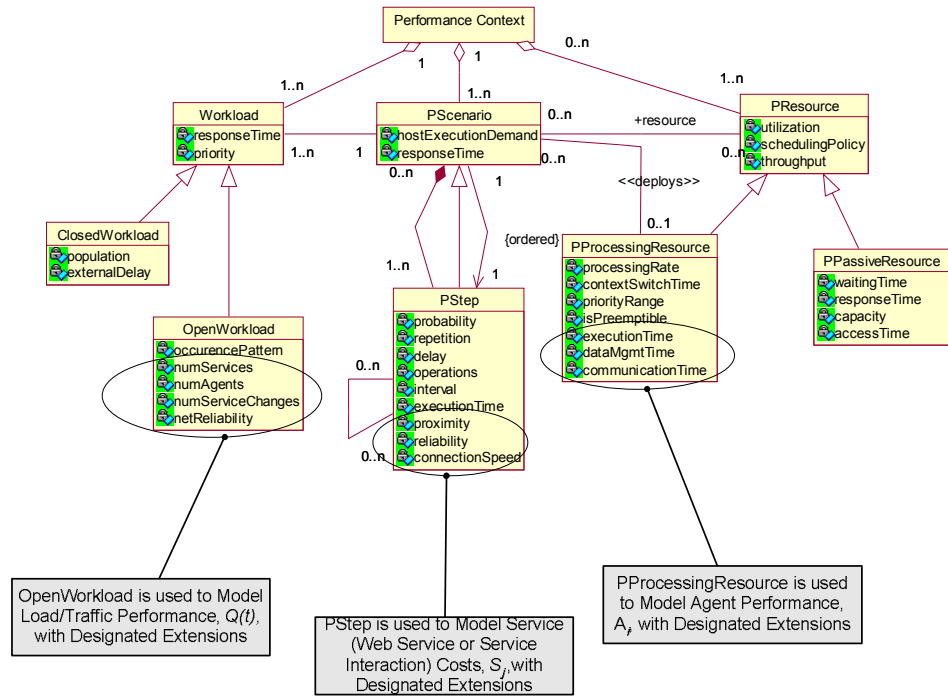


Figure 5. UML Profile of Schedulability, Performance, and Time with New Extensions for Service-Oriented Computing.

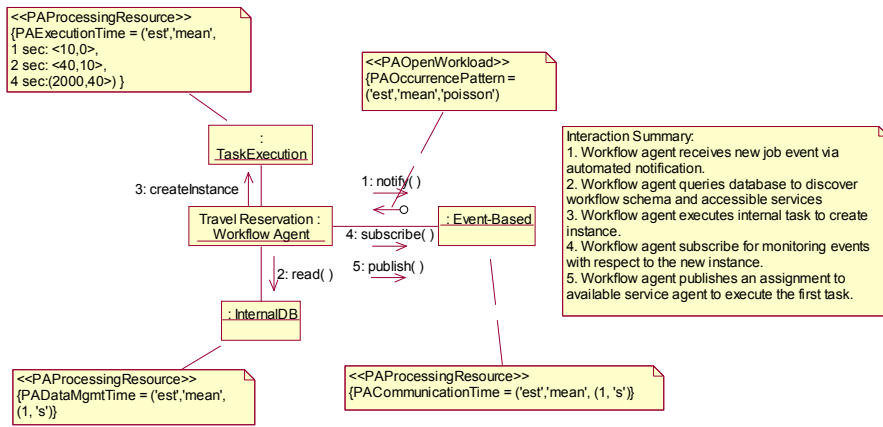


Figure 6. Adding Evaluative Attributes to the Instance Creation Interaction.

Table 1. Performance Information from Modeling Views

Description	Performance Factor	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	--	-----

As an example, Table 1 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 by the authors [3]. 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))

4. Case Study: Centralized Control vs. P2P

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). In consideration of space constraints, we do not show both system interaction views, however Figure 7 gives an overview of the two approaches and the full interaction views can be viewed [4]. 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 request (i.e. business processes containing the same type of underlying service).

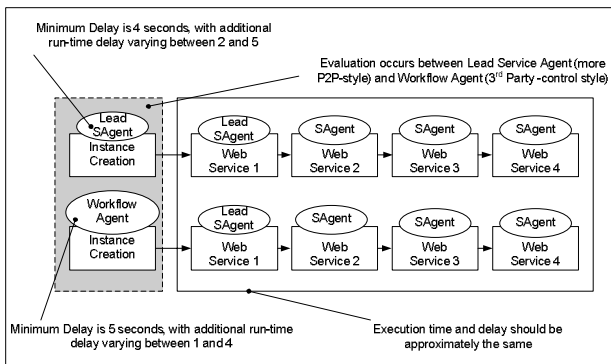


Figure 7. Two Instance Creation Implementations.

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 more 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.

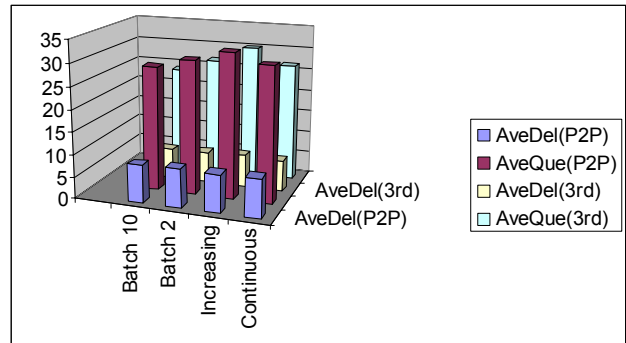


Figure 8. Simulation Results of Model Evaluation.

5. Discussion

Although the simulation approaches in this paper are similar to related work considering QoS for distributed workflows consisting of web service processes, the major innovation in our work is the combination of formal software process 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 Benatallah et. al. [2][3][41] who also investigates centralized control versus P2P. Cardoso [12] considers QoS in workflow by experimenting with factors, such as time, cost, reliability, and fidelity. Liu et. al. [25] also focus on web service composition, but not on execution and costs. 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. [27] and Skogan et.al. [30] 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.

6. Conclusion

The main contributions of this work are 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.

In future work, we intend to apply this approach to a real operational web-services environment in the electronic commerce domain. Although useful presently for side-by-side analysis, in future studies, the simulation software and modeling techniques can be furthered validated in a complete operational environment.

References

- [1] Benatallah, B., Dumas, M., and Sheng, O.Z. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Databases* 15(1):5-37, January 2005. Kluwer Academic Publishers.
- [2] Benatallah, B., Dumas, M., Sheng, Q., and Ngu, A. Declarative composition and peer-to-peer provisioning of dynamic web services. In 18th International Conference on Data Engineering., February 2002
- [3] Blake, M.B. "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
- [4] Blake, M.B. "Evaluating Agent-to-Agent Workflow Interactions for Service Composition: Third-Party Control or P2P?" Georgetown University, Tech Report CSTR-20030420-5, 2003
- [5] Booch, G. Rumbaugh, J, and Jacobson, I. "The Unified Modeling Language User Guide", Addison Wesley, Reading, MA 1999
- [6] BPEL4WS (2006): <http://www.ebpmi.org/bpel4ws.htm>
- [7] Cardoso, J. "Quality of Service and Semantic Composition of Workflows," PhD thesis, Univ. of Georgia, 2002
- [8] Dumas, M. and ter Hofstede, A.H.M., "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](http://www.springer.com/978-3-540-42667-1) 2185 Springer 2001, ISBN 3-540-42667-1
- [9] Liu, D., Peng, J., Law, K.H., and Wiederhold, G. "Efficient Integration of Web Services with Distributed Data Flow and Active Mediation" International Conference on Electronic Commerce, Delft, Netherlands, 2004
- [10] Orriëns, B., Yang, J., Papazoglou, M.P., Model driven service composition. Proceedings of the First International Conference on Service Oriented Computing, Trento, Italy, December 15-18, 2003
- [11] Skogan, D., Grønmo, R., and Solheim, I. "Web Service Composition in UML", In Proceedings of Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04), September 20 - 24, 2004 Monterey, California
- [12] UML® Profile for Schedulability, Performance, and Time, version 1.0 (2005): http://www.omg.org/technology/documents/formal/sc_hedulability.htm
- [13] van der Aalst, W.M.P. "Don't go with the flow: Web Services composition standards exposed", IEEE Intelligent, February 2003
- [14] 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)
- [15] Web Services (2006) <http://www.w3.org/2002/ws/desc/>
- [16] WSDL (2006) <http://www.w3.org/TR/wsdl>
- [17] WSFL (2006): <http://www.ebpmi.org/wsfl.htm>
- [18] Zeng, L., Ngu, A. Benatallah, B., O'Dell, M. An agent-based approach for supporting cross-enterprise workflows. In Proceedings of the 12th Australasian Conference on Database Technologies, 123-130, Queensland, Australia 2001