Approach for generating performance models from UML models of SOA systems

Mohammad Alhaj, Dorina C. Petriu

Department of Systems and Computer Engineering
Carleton University, Ottawa ON Canada, K1S 5B6
{malhaj|petriu}@sce.carleton.ca

Abstract

Model-Driven SOA is an emerging approach for developing service-oriented systems using models at different levels of abstractions and applying model transformations to generate either code or other models for the analysis of non-functional properties, such as performance. The paper proposes an approach for deriving layered queueing performance models for the evaluation of the runtime performance characteristics of such systems in the early development phases, before the entire system is built and can be deployed and measured. Early performance evaluation helps to choose an appropriate architecture, design and configuration alternatives, so that the final system meets its performance requirements. The starting point for derivation is a platform independent UML model of a SOA system representing the workflows, architecture of the underlying components offering services, and behavior of the corresponding runtime scenarios. A platform dependent model, obtained by weaving platform services into the platform-independent model through aspect-oriented modeling techniques, represents the source model for the transformation into a performance model. The deployment of the software on hardware resources is also part of the source model. The UML model is annotated with performance information by using the standard UML profile MARTE. The proposed approach is illustrated with a healthcare application.

Copyright © 2010 M. Alhaj, D.C. Petriu. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

1 Introduction

Service-Oriented Architecture (SOA) is an innovative architectural approach that aims to develop and deploy software applications as a set of reusable composable services [1]. SOA is applied to enterprise systems, web-based applications, multimedia, health-care, e-government, etc. Model-Driven SOA is an emerging approach for the development of service-oriented systems using models at different levels of abstractions and applying model transformations to generate either code or models for analyzing non-functional properties of the system, such as performance.

The top level model of a SOA system usually describes the business process through workflow models, which may be represented by different notations, either textual (e.g., BPEL) or graphical, such as the Business Process Modeling Notation (BPMN) or by UML activity diagrams [1]. In our work we chose UML activity diagrams, due to three reasons: a) it is desirable to use a single modeling language for all SOA levels - and UML is a general-purpose modeling language that fits the best our needs; b) UML includes a standard extension mechanism allowing to add new semantics to the models; and c) there is a standard UML profile MARTE [3] for performance annotations. Another OMG standard relevant to this study is SoaML, the UML Profile and Metamodel for Services meant to facilitate service modeling [2].

Software Performance Engineering (SPE) [4] is a methodology aiming to insure that software products are built to meet their performance requirements. SPE uses quantitative performance models such as queueing networks, Petri nets, simulation models, etc., to evaluate the temporal
responsiveness of the system (such as response times, delays and throughputs). The models can be used to study the performance impact of different architecture, design and/or configuration alternatives under different workloads, leading to advice for improvements. SPE begins early in the software process, before serious barriers to performance are frozen into architecture and design, and continues throughout the software lifecycle.

In this paper we propose an approach for generating automatically a Layered Queueing Network (LQN) performance model from a UML model of a SOA system with MARTE performance annotations. The proposed approach extends previous work of the authors [5][10] as explained in Section 7.B. The approach will be illustrated with an example from the healthcare domain.

Another goal of this paper is to separate the performance annotations for the overheads introduced by the service-oriented platforms from the annotations for the service-based applications themselves (workflows and actual services). This allows us to reuse the performance characteristics of different platforms, making the building of the performance model more efficient. To address the problem of composing the platform independent model of a SOA system with the platform model we apply the Aspect Oriented Modeling (AOM) approach [14][15]. An Aspect model in its Generic form is initially developed to describe a crosscutting concern – in this case the platform operations for a service invocation. The generic aspect model is then bound to the context of the application, becoming a context-specific model) and it is composed with the primary model at defined joint points (i.e., in every place where a service invocation take place).

The paper is organized as follows: Section 2 presents related work; Section 3 describes the proposed approach, Performance from Unified Model Analysis for SOA (PUMA4SOA), for transforming annotated UML models for SOA systems into performance models; Section 4 describes the platform independent UML model of the SOA system annotated with performance information; Section 5 presents the Aspect model of the service invocation, and the AOM approach of binding and composing the aspect model with the primary SOA model; Section 6 presents the transformation principles from the designed UML models to Core Scenario Model (CSM); Section 7 describes the LQN target model and some performance results; and Section 8 presents the conclusions and directions for future work.

2 Related Work

In the past decade, OMG has standardized two UML profiles which are extending UML for the real-time domain: first the “UML Profile for Schedulability, Performance and Time (SPT)” defined for UML 1.X versions, and second the “UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)” [3] defined for UML 2.X versions. Both SPT and MARTE include their own performance subprofiles, which allow annotating UML models of real time systems with quantitative resource demands and other information required for performance analysis. The standardization efforts on the one hand and the emergence of Model-Driven Architecture on the other hand, have triggered a lot of research on the automatic derivation of different kinds of performance models from annotated UML software models. Our research group has brought significant contributions to this field, not only by participation in the OMG working groups that have defined SPT and MARTE, but also through work on model transformations for automatic derivation of performance models. An example of such work is the PUMA transformation [5], which is extended for SOA by the approach presented in this paper.

One of the best known technologies for realizing SOA systems are Web Services, so it is not surprising that there are many efforts to model and analyze the performance of such systems. As Web Services use the SOAP protocol, which is XML-based and introduces performance overhead to parse and serialize XML documents, there was a lot of interest in studying the performance impact of different SOAP implementations, as for instance in [7]. Other research on building performance models for web services takes a two-layered user/provider approach in [6] and [8]: the user is a set of workflows and the provider a set of services deployed on a physical system. Performance information about service capabilities and invocation mechanisms is given by the means of P-WSDL (Performance-enabled WSDL). In [6] a LQN model is generated for the system, and in [8] the queueing network formalism is used to derive performance bounds.
3 PUMA4SOA

Performance from Unified Model Analysis for SOA (PUMA4SOA) is a modeling approach proposed in this paper, that is an extension of the PUMA approach developed in our research group [6][11]. PUMA reduces the complexity of model transformation by providing a unified interface between different kinds of software design models and performance models. Fig.1 shows the steps of PUMA4SOA. The difference between the original PUMA and the extended one for SOA stems from the kinds of design models accepted as input. In the original PUMA, the design model includes normally three views: software architecture, scenarios and deployment. In PUMA4SOA, the design model is decomposed in a Platform Independent Model (PIM), a middleware platform model and a deployment model, which maps the software to the hardware platform. In turn PIM contains three views: workflow model, service architecture model, and scenario model. The middleware platform model is represented as an aspect model, as shown in Fig. 1. These models are described in details in the next section.

Fig. 1 illustrates the steps of the PUMA4SOA transformation. First, the Aspect model of the middleware (which in our case represents the operations performed for a Service Invocation) is composed (woven) in the primary PIM model at different join points (i.e., for each service invocation in the SOA system). The result of the aspect composition plus the Deployment diagram constitute a Platform-Specific Model (PSM) of the system. The PSM design model is then transformed into a Core Scenario Model (CSM), similar to the original PUMA [6]. CSM is an intermediate scenario-based model described by a light-weight metamodel [18]. CSM is used to capture the performance characteristics from different design models. The CSM model is transformed into performance models of different types: queueing networks, Layered Queueing Networks (LQN), timed Petri nets or simulation models [6]. Once the performance model generated, performance analysis is done using existing solvers. The performance results are analyzed and used to produce feedback to the designers for improving the design model.

4 Platform-Independent Model of a SOA System

The starting point for generating a performance model for a SOA system is a UML software model with three levels of abstraction. The MARTE profile [4], defines stereotypes which are used to indicate the performance characteristics of model elements. Each level represents a part of the performance details that will be used together with the other parts to build the performance model. The three abstractions levels are as follows:

1. **Workflow Model** represents a set of business processes. Each workflow contains a sequence of activities and actions controlled by conditions, iterations, and concurrency. Although, there are several modeling languages for workflows, we are using UML Activity Diagrams because the performance profile used for performance analysis, MARTE, is a UML profile.

2. **Service Architecture Model** describes the service capabilities arranged in a hierarchy showing anticipated usage dependencies. The level of service granularity has a substantial effect on the system performance [1]. In SOA systems, where the environment is heterogeneous and distributed, the cost of marshalling/unmarshalling data increases the messaging overhead. So, it is preferred to use services with a large granularity. On the other side, large granularity produces unnecessary coupling between SOA systems, which makes them hard to implement, or to evolve and change over the time [3]. Service Archi-
tecture Modeling helps the modeler to manage the trade-off between granularity and performance.

3. Service Behavior Model refines the workflow behavior, giving more details about the services invoked. Each workflow activity may be refined by a sequence diagram which represents its detailed behavior, including the invocations of other services and the interaction between participants.

The three-level PIM model is illustrated with a healthcare case study, Eligibility Referral System, which was introduced in [10]. The top-level model from Fig. 2 describes a workflow for processing the transfer of a patient from one hospital to another. The system supports the process of transferring a patient from a certain hospital to another. Three organizations are involved, the sending and receiving hospitals and an insurance company. The workflow shown in Fig. 2 begins with the sending hospital determining the right replacement hospital for a patient that needs to be transferred. The next processes involve getting the physician approval and the necessary payment information. Once this is done, the sending hospital waits for an acknowledgement from the receiving hospital that it will accept the patient. Finally the sending hospital schedules the transfer with the receiving hospital and completes the transfer.

The activities of different participants are shown in different swimlanes. A swimlane stereotyped <<PaRunTInstance>> indicates a concurrent participant that will be executed by a specified process instance; such a process may have a poolSize attribute giving its number of threads. Each workflow activity is stereotyped as <<PaStep>> to indicate a scenario step. The attributes of a step are: hostDemand for the required execution time, prob for its probability of execution, and rep for the number of repetitions. An activity can be either atomic or composite. The first activity, Process Eligibility Referral, stereotyped as <<GaWorkloadEvent>> defines the stream of events that produces the workload to the system. The workload in the example has a closed arrival pattern with a population attribute that defines the number of concurrent users, and extDelay defining the delay between the end of one cycle and the start of the next. A control flow arc between two swimlanes representing the communication between participants are stereotyped as <<PaCommStep>> to indicate the conveyance of a message, which has a msgSize attribute to indicate the amount of data transmitted by the sender. MsgSize can be used to calculate the overheads at the transmission, and receiving points using commRcvOvh, and commTxOvh attributes defined for the execution host of the respective participant. Parallel execution of Perform Physician Authorization and Perform Payor Authorization activities takes place; after they are done, they join back to the original path. Also, a decision node (branching) takes place after the Validating Transfer Request activity.

Fig. 3 shows the Service Architecture model which defines the participants, and their binding roles in each service contract and the capabilities of each participant. Components, the logical participants, are stereotyped as <<ScheduledResource>> to indicate a process or a thread pool. Service points are stereotyped with <<PaRequestedService>> to indicate services provided by subsystem. <<PaRequestedService>> is a subtype of <<PaStep>> and may be refined by a BehaviorScenario model element.

UML Sequence diagrams are used to model the behavior of each activity defined in the workflow model in Fig. 2. The behavior can be either a single message (atomic activity) or a scenario (composite). Fig. 4 shows as an example the behavior of Initial Patient Transfer activity. Lifelines are stereotyped with <<PaRunTInstance>>. Messages are stereotyped with <<PaStep>>. A workload event is not needed here, because it is given in the Workflow model in Fig. 2.

Fig. 5 shows the UML Deployment diagram, which is important for performance analysis. The three subsystems, registration, insurance and admission from Fig. 3 are distributed over the Internet. <<GaCommHost>> identifies a physical communication link, which may have blockT attribute for the network latency, and capacity for the maximum throughput rate. <<GaExecHost>> identifies a processor resource hosting the components deployed to it, which may have commRcvOvh and commTxOvh attributes for transmitting and receiving communication overheads. <<ScheduledResource>> indicates a process with a thread pool. The deployment diagram show three different LANs, one for the hospital transferring the patient, one for the hospital receiving the transfer, and one for the insurance company.
5 Aspect Model for Service Invocation

The Aspect-Oriented Modeling (AOM) architecture consists of a primary model, which describes the core design decisions and a set of aspect models, each describing a concern that crosses the primary model [14]. In our case, the primary model is the PIM model of a SOA system described in the previous section, and the aspect

---

**Figure 2. Workflow model of Eligibility Referral System**

**Figure 3. Service Architecture model of Eligibility Referral System**
model is defined for a Web Service platform and represents the middleware operations performed for a service invocation. (We are planning to define aspects for other platform operations, as well). Initially, a generic aspect model is developed to describe the solution proposed by the aspect in a general way, unrelated to the primary model [15]. Later, a generic aspect model can be instantiated multiple times to produce multiple context-specific aspect models. For each instantiation, the generic parameters of the generic aspect are bound to concrete parameters defined for the primary model context. Once the context-specific aspect models are instantiated, the composed model will be generated by weaving the context-specific aspect model into the primary model. The weaving involves identifying the appropriate join points in the primary model and inserting the context-specific aspects at those join points. The join points are found based on user-defined or system-defined conditions called point cuts. The deployment diagram in Fig. 6 describes the generic Hosts and Artifacts involved in the Service Invocation aspect model. Note that we use a naming convention as in several other papers (e.g., [15]): an identifier beginning with a vertical bar ‘|’ indicates generic role names. On |ClientHost there are allocated: |client (the service consumer),
|XMLParserC (the parser of XML message at the consumer side), and |SOAPClientC (the XML serializer/deserializer at the client side). On the |ProviderHost, there are: |Provider (the service provider), |XMLParserP (the parser of XML message at the provider side), and |SOAPClientP (the
XML deserializer/serializer at the provider side). The aspect models of service invocation describe the behavior of a generic middleware which can be later instantiated to many specific contexts.

Fig. 7 describes a Service Request Invocation and Service Response. In the Service Request Invocation model, the |client| sends a request message to the |soapClientC|. The |soapClientC| requests the |xmlParserC| for message parsing, and then serializes the message. At the provider side the |soapClientP| deserializes the message, requests the |xmlParserP| for message parsing, and then sends the invocation request to |provider|. In its turn, |provider| performs the service operation and sends back the service response, as described in the Service Response model. The messages in both interactions are stereotyped either with <<PaStep>> or <<PaCommStep>> to indicate scenario steps or conveyance of messages. The attributes of |Steps| and |CommSteps| have variable values that will be bound later to concrete values. Generic aspect models are transformed into context-specific aspect models based on two phases: binding the resource roles to actual resources, and then assigning context-specific performance values to steps, processing demands, probabilities, etc. The actual resources can be either found in the primary model, or they can be newly defined resources. Using the sequence diagram Initial Patient Transfer Service shown in Fig. 4 as a primary model, we obtain the resource bindings for context-specific aspects as shown in Table 1.

Table 1. Binding generic to concrete resources

<table>
<thead>
<tr>
<th>Generic Aspect</th>
<th>Context-Specific Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Invocation</td>
<td>RecordTransferForm Invocation</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 describes the concrete values of the variable of the stereotypes defined in Generic models. The values of the service demands and message sizes are similar to those used in [17], [18].

The next step is to generate the composed model by weaving the context-specific aspect models into the primary model. The model transformation for aspect weaving is not presented in this paper, but it is described in several papers from literature, such as [19]. In our example, there are two types of point cuts to specify the place for weaving the context-specific aspect models:

- Point cut for Service Request Invocation
- Point cut for Service Response.

Fig. 8 shows the composed model for our example, which contains |xmlParserNA|, |soapNA|, |xmlParserDM1|, and |soapDM1| as new resources. Aspect model composition must be applied to every service invocation in the PIM model.

6 From Annotated SOA System Models to CSM

The next step is to transform the annotated platform-specific model of the SOA system into CSM. The Core Scenario Model (CSM) is a main part of PUMA. It provides a metamodel for an intermediate model which allows for correlating multiple UML diagrams, extracting the behavior elements and their performance annotations and attaching important resource information obtained from the design model. CSM supports the creation of many different kinds of performance models [19], [20]. The CSM metamodel captures the performance concepts from the SPT profile, and recently was updated for MARTE as well.

The source model processed by the PUMA4SOA toolset is built with an UML tool (IBM Rational Software Architecture RSA7.0). CSM models are automatically generated from UML models annotated with MARTE profile from behavior diagrams (Sequence or Activity diagrams). Other performance entities related to the workload, resources, service granularity can be captured from workflow models, deployment diagram, and service architecture models. Fig. 9 illustrates two CSM scenarios generated from UML+MARTE design models. Fig 9a is the top level CSM scenario representing the workflow diagram. The scenario begins with Start, followed by ResourceAcquire, and finishes with ResourceRelease and End (which are generated
Figure 8. Model for Initial Patient Transfer after aspect composition

whenever there is an initial node or final node in an activity diagram). ResourceAcquire and ResourceRelease are generated whenever there is an incoming or outgoing edge crossing partitions. An atomic node is generated as a step with a single line box, while a composite node is generated as a refined step with a double-line box on both sides. Refined steps describe a sub-scenario, which can be viewed by double-clicking on the refined step. Fig.9b is an example of a sub-scenario. Fork-Join pathconnections are generated to indicate parallel paths, while Branch-Merge pathconnections are generated to indicate alternative paths. Fig.9b describes Initial Patient Transfer service generated from the composed model defined in the Fig 8, corresponding to a composite node in Fig. 9a.

Generating CSM from sequence diagrams is quite similar to generating it from activity diagrams. In a sequence diagram, the ResourceAcquire is generated whenever a lifeline first receives a synchronous or asynchronous call message or execution starts first. A ResourceRelease is generated whenever a lifeline sends an asynchronous call message or its execution finishes. A Step is generated whenever there is an incoming message to the lifeline. The CSM transformation viewer tool used to obtain the representation from Fig. 9 does not show all CSM model elements, just the graph of steps. The other elements (e.g., Resources, Hosts, and logicalResources), attributes (e.g., hostDemand, probability, repetition, and overheads), and values are generated and stored in an internal data structure representing the CSM model, which can be saved in an xml-based file.
7 From CSM to LQN

A LQN: The Target Model

The LQN model [11] is an extended queueing network model which is able to represent nested services (see Fig. 11). An LQN model is a set of objects called “tasks” that offer services called “entries”. The entries of a task may send requests to entries of other tasks. LQN is used to model several types of system behavior and inter-process communication style. Tasks represent both software and hardware devices, and allow for multi-threading and nested services. An entry may contain a graph of activities. Graphically, the software tasks are represented with thick rectangles and the entries with attached thin rectangles.

The hardware devices are represented as ellipses. The LQN solver is used to produce task service times (including waiting for nested services), queueing delays, processor utilization, response time, and throughput. The results are used to identify the performance hotspots in the system. Although the aspect composition takes place in

![Diagram of CSM for Eligibility Referral example](image)

Figure 9. CSM for Eligibility Referral example
the software model, the composition effects are reflected also in the LQN model, as illustrated in Figure 10. Fig. 10.a shows an LQN model corresponding to a simple PIM model containing two tasks: a Requester, and a Service. The Requester has a request entry, while the Service has a provide entry. Fig. 10.b shows the LQN model corresponding to the PSM obtained by composing the simple PIM with the service invocation aspect, which is adding middleware functionality to the system; the PSM model takes also into account the hardware resources from the deployment. The middleware is modeled in LQN with three tasks: soap (which has send, receive, and reply entries), xmlParser (which has a parse entry), and net (which accounts for the network delay). The net task is invoked in the LQN model to account for the network delay suffered by both the service request and the reply. The LQN model is using synchronous messaging (the dark solid arrows), and forwarding messaging (dotted arrow messaging). Fig 10.c and 10.d are models equivalent with the one from Fig. 10.b, obtained by replacing the forwarding with synchronous messages (as done by the LQN solver) and by concatenating the middleware tasks running on the same hosts [12].

**B Model Transformation Approach**

As already mentioned, the transformation approach proposed in this paper extends a model transformation technique called PUMA, previously developed in our group (see [6][11]). The main difference is due to the fact that a SOA model has additional levels of abstraction: platform-independent and platform-specific. In turn, PIM has a two-level behavior model. The PUMA transformation takes as input an annotated UML model composed of a structural view (the high-level software architecture showing concurrent components and their deployment on hardware...
resources) and a behavioral view represented by one or more scenarios (modeled either as sequence or as activity diagrams). In the case of SOA systems, the scenarios are represented by two levels of abstraction: the workflow model and the service behavior model. Other differences between PUMA and PUMA4SOA are due to a higher emphasis on service contracts and the fact that we use aspect-oriented models to capture the performance overheads of the service-based middleware.

For simplicity, the mapping concepts from software to performance models are explained here as if the transformation would be direct from annotated UML to LQN. In fact, both PUMA and our extended transformation (being currently implemented) are using the intermediate model CSM described in the previous section. In a direct or indirect mapping, the structure of the LQN model is generated from resources identified in the high-level service architecture and deployment. In principle, active software component instances and hardware devices (which are all resources) are mapped to LQN tasks. In some cases, LQN tasks are also generated from passive instances representing logical resources shared by active instances.

Task entries correspond to service access points. The request arcs between task entries are obtained from the two-level behavioral description (workflow level and service level). LQN activities correspond to workflow activities executed in the same swimlane. The resource demands (such as execution times) are obtained from the attributes of different steps.

C Performance Results

In this section we analyze the performance of the Eligibility Referral System for two design alternative: a) finer service granularity corresponding to the service architecture from Fig. 3, and b) coarser service granularity, where the invocations of low-level services for accessing the database DM1 are replaced with regular calls. The second solution practically integrates the functionality of the lower level services in the higher-level services provided by the component NursingAccount. The LQN results will show the response time reduction obtained by making fewer expensive service invocations using SOAP and XML in the same scenario.

Fig. 11 shows the LQN model generated for the Eligibility Referral System for the first alternative. The main purpose of our performance analysis is to find the performance bottleneck (i.e., software or hardware components that saturate first and throttle the system). After identifying the bottleneck, we apply a series of hardware and/or soft-
ware modifications to mitigate the bottleneck and to improve the response time and the throughput of the overall system.

The results of the performance analysis, shown in Fig. 12 give the response time and throughput of the system versus the number of users ranging from 1 to 100. The results for the system with finer service granularity are given in Fig. 12.a and 12.b.

We analyzed several system configurations:
- A: the base case, where the multiplicity of all tasks and hardware devices is 1, except for the number of users. The Transferring processor is the system bottleneck, because it is the first resource to saturate. To remove the bottleneck we need to increase its processing power either by replacing it with a faster processor or by increasing its multiplicity.

Figure 12. LQN results for finer and coarser service granularity
B: The previous bottleneck is resolved by increasing the multiplicity of the bottleneck processor node to 4 processors. The response time is reduced only slightly because the next bottleneck kicks in immediately. The bottleneck is a software task this time, the middleware MW_NA, which is single-threaded and does not provide enough concurrency, hence it throttles the system.

C: The software bottleneck is resolved by multi-threading MW_NA. The maximum throughput increases by 16.5% with respect to case B. The bottleneck moves to Disk1.

D: Increasing to 2 the number of disks units for Disk1 and adding additional threads to the next software bottleneck tasks, dm1 and MW_DM1, the maximum throughput goes up by 24% with respect to case C. The bottleneck moves to DM1 processor.

E: Increasing the number of DM1 processors to 2 has a considerable effect; the throughput increases by 32% with respect to case D.

The results for the case with coarser service granularity are shown in Fig. 12.c, and 12.d. The analysis revealed the following cases:

A: This is the base case, where the multiplicity of all tasks and hardware devices is 1, except for the top users. The software task dm1 is the initial bottleneck. In order to improve the performance, we need to mitigate the bottleneck by increasing the concurrency level of the bottleneck task.

B: The software bottleneck is resolved by multi-threading dm1. The response time is reduced slightly and the bottleneck moves to Disk1.

C: Increasing to 2 the number of disks units for Disk1 has a considerable effect. The maximum throughput goes up by 60% with respect to case B. The bottleneck moves to the Transferring processor.

D: Increasing the multiplicity of the Transferring processor to 2 processors, and adding additional threads to the next software bottleneck task MW-NA; the throughput is increased by 11% with respect to case C.

Fig. 12.e and 12.f illustrate the difference between the cases D of the system with finer and coarser granularity. The compared configurations are similar in the number of processors, disks and threads, except that the latter performs fewer service invocations through the web service middleware. The improvement is considerable (about 40% for a large number of users). This is easy to explain in our example, since 8 service invocations are executed by the MW-dm1 in the case with finer service granularity. These invocations were replaced with regular messages in the case with coarser service granularity.

The effect of service granularity on system performance should be evaluated at an early design stage, when the service boundaries are defined. The proposed analysis helps the modeler to decide on the right granularity level, making a trade-off between system performance and the level of coupling between the system services.

8 Conclusions

The paper proposes an approach for deriving automatically LQN performance models from UML software models of SOA systems with MARTE performance annotations. We are currently in the process of implementing the proposed model transformation in Java as an Eclipse plugin, which takes as input the layered SOA model exported from a standard UML editor (such as IBM Rational Software Architect). We are using the Eclipse UML2 plugin that implements the UML 2.X metamodel in order to navigate and manipulate UML models.

As future work, we are planning to make full use of the service modeling concepts introduced in SoaML in order to investigate the performance effects of allocating service capabilities to services and participants in different ways. We will also define aspects for other middleware services. We intend to apply this performance evaluation approach to SOA systems in the healthcare domain.

Acknowledgements

This work was partially supported by the Centre of Excellence for Research in Adaptive Systems (CERAS) and by the Natural Sciences and Engineering Research Council (NSERC) and industrial and government partners, through the Healthcare Support through Information Technology Enhancements (hSITE) Strategic Research Network.
About the Authors

Mohammad Alhaj is a PhD student in the Department of Systems and Computer Engineering at Carleton University, Ottawa ON Canada. He has completed his M.Appl.Sc. in Electrical and Computer Engineering at Carleton University in 2008. His research interests are in model-driven development of service based systems, aspect-oriented modeling, models transformations and performance analysis.

Dorina C. Petriu is a professor in the Department of Systems and Computer Engineering at Carleton University, Ottawa, ON, Canada. Her main research interests are in the areas of performance modeling and model-driven development, with emphasis on integrating performance engineering into the software development process. She was a contributor to two OMG standards, SPT and MARTE, which extend UML for real-time system modeling and analysis. Dr. Petriu is a Fellow of the Engineering Institute of Canada and of the Canadian Academy of Engineering, a Senior Member of IEEE and a member of ACM.

References