Studying the Impact of Design Patterns on the Performance Analysis of Service Oriented Architecture

Nariman Mani, Dorina C. Petriu, Murray Woodside
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
{nmani | petriu | cmw}@sce.carleton.ca

Abstract— **Design patterns for Service Oriented Architecture (SOA) suggest solutions for architectural, design and implementation problems, but these changes also affect performance and other non-functional properties. A performance model can be generated from a SOA software model (plus some implementation and deployment advice) and used to study these impacts. The paper proposes to propagate the change in the software model due to applying a given pattern into the corresponding change in the performance model by an incremental transformation. The software model and the patterns are described using UML extended with the profiles SoaML (for service-oriented system design) and MARTE (for performance annotations). The application of a pattern is described by application rules specified by the user. Systematic (but not, at this point, automated) incremental transformations are explored and evaluated for effectiveness on case study examples.**

Keywords— software performance analysis; service oriented architecture; Model Driven Engineering; layered queuing network.

I. INTRODUCTION

Service-Oriented Architecture (SOA) is an innovative architectural approach for developing and deploying software applications as a set of reusable composable services [1]. SOA provides many architectural benefits to the design of a distributed system including reusability, adaptability, and maintainability. Quality of SOA-based systems may be improved by applying SOA design patterns, which provide generic solutions for different architectural, design and implementation problems. However, changes due to design patterns may affect performance and other non-functional properties positively or negatively [1][2]. In this paper we propose to use performance evaluation to screen candidate pattern applications, after an initial SOA design has been modeled and evaluated for the first time.

In Model Driven Engineering (MDE), model transformations are used to generate the performance model of a system from its software design model extended with performance annotations [3] [4]; the goal is to evaluate the system performance in the early development phases and to help designers in their decisions. In this paper we propose an approach for propagating the change due to the application of a design pattern from the software model (SModel) to the corresponding performance model (PModel) by incremental model transformation. The process starts when a designer selects a design pattern to solve a performance or non-performance problem and decides where to apply it by binding the roles defined in the pattern to specific SModel elements. Using SModel to PModel mapping information, the affected PModel elements are identified and the change is applied. The benefit is that the construction of the PModel from scratch can be avoided every time a small subset of elements is changed. The faster incremental change propagation can be integrated more easily in an evaluation and state space exploration process, where many small changes are made and the PModel is solved multiple times. In general, there is an intrinsic interest in incremental model transformations, and we feel that the application of design patterns is an appropriate application of such transformations. A pattern generates a neighbourhood in the PModel space that we wish to characterize. Also, parameterized patterns particularly make it necessary to do some parameter exploration in the PModel space and the traceability back to SModel parameters is better in the proposed approach, due to the mapping table between the two models.

The paper is organized as follows: section II presents the overview of the proposed approach; section III discusses related work; section IV presents a SModel example and the proposed mapping between SModel and PModel; section V gives two examples of design patterns and their performance effects and section VI presents the conclusions and future work.

II. OVERVIEW

Figure 1 provides a high-level overview of the approach proposed in this paper. The shaded area at the top includes the activities involved in the transformation of a SModel expressed in UML annotated with SoaML and MARTE into a LQN performance model (PModel). We are using the PUMA transformation that has been developed in our research group

[3] .The PModel is solved with existing solvers and the results are used to evaluate the performance of the system. In order to facilitate further incremental model transformations, we propose to create a mapping table between SModel and PModel elements, as explained later in the paper. The performance analysis results as well as results of other non-functional analyses of the SModel may be used by developers to identify design problem that can be solved by applying design patterns. The logic behind the selection of the appropriate design pattern is not addressed in the paper.

The second shaded area includes activites for identifying the SModel elements affected by the pattern and the propagation of change to the PModel. First, a set of applications rules is manually extracted from the pattern definition, which help detect the affected SModel elements and parameters. Using the mapping table previously created, the PModel elements and parameters affected by the pattern are identified and the changes are applied to the PModel. The mapping table is also updated.

.

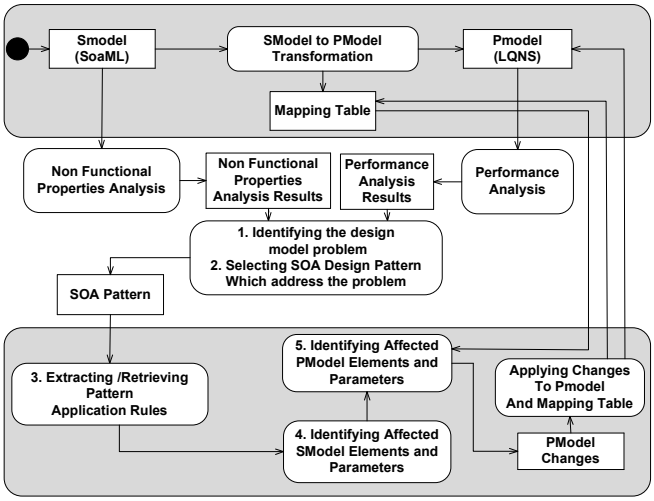

Figure 1 : Proposed Approach Overview

III. RELATED WORK

There are a significant number of papers on the derivation of a PModel of a system from the design model (SModel). The OMG standards for UML performance annotations, SPT and MARTE, have enabled research to transform UML design specifications into many kinds of performance models, based for example on Queuing Networks [15], Layered Queuing Networks [3][4][7] Stochastic Petri nets, etc..

We have already mentioned the PUMA transformation developed in our research group, which takes as input different SModels and transforms them into different performance models [3][4]. PUMA uses a pivot language, Core Scenario Model (CSM), to extract and audit performance information from different kinds of design models (e.g., different UML versions and types of diagrams) and to support the generation of different kinds of performance models (e.g., QN, LQN, Petri nets, simulation). In [7], a graph-grammar based algorithm was proposed to divide the activity diagram into activity subgraphs, which are further mapped to LQN phases or activities. In this we propose to create a mapping table during the PUMA model transformation to help us in tracing future incremental modifications of the design model and its impact on the associated performance model.

Another category of related works studies the impact of design patterns on software performance. Beside the patterns which are recognized as best practices for software development, Smith and Williams [8] introduced general performance anti-patterns, which exclusively focus on performance concerns. Anti-patterns are defined as common design mistakes that consistently occur, causing undesirable results.

There are only a few works on studying the impact of the patterns/anti-patterns on the performance of the software applications. Cortellessa et al. [9] present an approach based on anti-patterns for identifying performance problems and removing them. The identification of an anti-pattern suggests the architectural alternatives that can remove that specific problem.

Menascé et al [10] presents a framework called SASSY whose goal is to allow designers to specify the system requirements using a visual activity-based language and to automatically generate a base architecture that corresponds to the requirements. The architecture is optimized with respect to quality of service requirements (i.e. as measured by several performance metrics such as execution time and throughput) through the selection of the most suitable service providers and application of quality of service architectural patterns.

Parsons and Murphy [11] introduce an approach for the automatic detection of performance anti-patterns by extracting the run-time system design from data collected during monitoring by applying a number of advanced analysis techniques.

Xu [12] applied rules to performance model results to identify performance problems and to propose solutions for fixing them, which resulted in changes at the PModel level.

In this paper, we propose a technique for studying the performance impact design pattern applied to the SModel of a system, by incremental change propagation to the PModel.

To the best of our knowledge, none of the works in literature addresses SOA design patterns incremental change propagation to the PModel as their main focus of study.

IV. SOA PERFORMANCE ANALYSIS

In this section we present an example of SModel and discuss the proposed mapping table between SModel and PModel.

A. SModel in UML extended with SoaML

The SoaML specification is an OMG standard for the design of services within a SOA system [5]. It is a UML profile that provides a standard way to architect and model SOA solutions. In this section we provide a brief description of how to model a SOA system according to the SoaML specification [13]. The first step is to identify services by analyzing the business goals and objectives of the system, as well as the businesses processes to be implemented for meeting these objectives. Using these processes it is possible to indentify other business-relevant services. Next, the specification of each service is

modeled by defining the service interfaces between consumers and providers of the service. These service interfaces include the provided and required interfaces, the roles that the interfaces play in the service specification, and the rules or protocol for how the roles interact. The third step, Service Realization, starts with identifying services that each participant provides and/or uses. This process has an important effect on service availability, distribution, security, transaction scopes, and coupling. The third step actually models how each service functional capability is implemented and how the required services are actually used. The fourth step, called Service Composition, is about assembling and connecting the service participants' modes and then choreographing their interactions to provide a complete solution to the business requirements. Finally the last step is Service Implementation.

In SoaML, UML interaction and/or activity diagrams are used to express the behavior of a service and its interactions within a SOA [13] (i.e. in service identification from requirements, service specification, etc.). We also use this kind of behavioral diagrams for SOA performance analysis. Figure 2 shows the nested structure of a Shopping &Browsing service, the offered service interfaces and the behavior model of one of its processes, the Shopping process.

In order to be able to derive a PModel from a SModel, the latter must contain a specification of the concurrent runtime component instances and their deployment to processors. Figure 3 also shows a deployment diagram which is a part of the SModel example given in this paper. In the absence of specific allocation information, one may assume there is one host node for each service.

B. Performance annotations with MARTE

One of the challenges in the SOA performance analysis is to bridge the gap between MDE techniques and existing NFP (i.e. performance) analysis formalisms and tools, such as Layered Queuing Network (LQN). According to [4], this challenge can be tackled by adding annotations describing the respective NFP to the software model and defining a model transformation from the annotated software model to the formalism used for NFP analysis. In the case of UML-based software development (e.g. SoaML), the extensions required for NFP-specific annotations are defined as UML profiles. Two standard UML profiles provide the ability to define performance annotations: the UML Profile for Schedulability, Performance and Time (SPT) defined for UML 1.X versions and the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) defined for UML 2.X versions [6]. Since this paper uses UML 2.X, we focus on MARTE for performance annotation.

Such annotations help us not only in transforming SModels into PModels, but also in tracing future modifications due to the application of design patterns.

In Figure 3, the processing nodes are stereotyped as *«GaExecHost»* and the communication network nodes as *«GaCommHost»*. In Figure 2 showing the interface and behavioral diagram, MARTE performance annotations are used in the scenario model to indicate the scenario steps, the workload and the concurrent runtime instances corresponding to lifeline roles. *«PaStep»* represents the execution of an activity or the operations invoked by a message. Examples of execution step attributes are *hostDemand* giving the value and unit for the required execution time and *prob* giving the probability for the optional steps. The first step of the scenario has the scenario workload *«GaWorloadEvent»* attached to it, which defines a closed workload with a population given by the variable *$Nusers* and a think time for each user given by the variable *$ThinkTime*. In Figure 2 each swim line role is related to a runtime concurrent component instance, as indicated by *«PaRunTInstance»*.

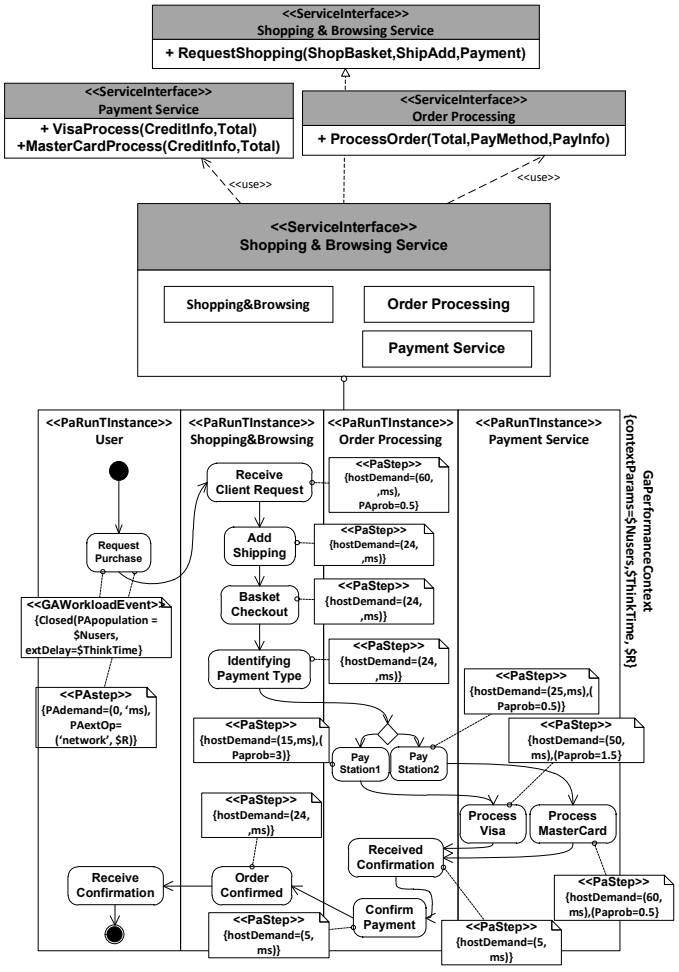

Figure 2 : Shopping Process Activities

C. Mapping from SModel to LQN Performance Model

In this section, we apply existing techniques [3][4][7] to convert the UML+SoaML+MARTE SModel to a LQN model. We also propose to modify the process by creating a mapping table which will be used for incrementally propagating the changes to the PModel (see Figure 1).

Figure 4 shows a LQN model corresponding to the SoaML model of Figure 2. For each service there is a task, shown as a bold rectangle, and for each of its operations there is an entry, shown as an attached rectangle inside the task. The task has a parameter for its multiplicity (e.g. {10} for Shopping&

Browsing task in Figure 4 ; {1} is default) and the entry has a parameter for its host demand, equal to the *hostDemand* of the operation in SoaML (e.g. [0.5 ms]). Calls from one operation to another are indicated by arrows between entries (a solid arrowhead indicates a synchronous call for which the reply is implicit, while an open arrowhead indicates an asynchronous call). The arrow is annotated by the number of calls per invocation of the sender (e.g. (3)). For deployment, the host node is indicated by an oval symbol attached to each task. Therefore, a very straightforward mapping can be established as follows (more details are discussed in [3][4][7]):

- Mapping «GaExecHost» to LQN hosts (processors)
- Mapping «PaRunTInstance» to LQN tasks
- Mapping «PaStep» to LQN entries

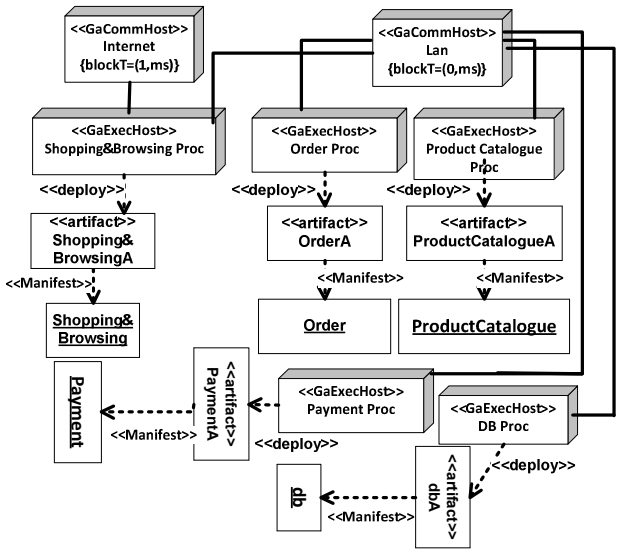

Figure 3 : SOA Deployment Diagram

D. SModel to PModel Mapping Table

We define the format of this mapping table as follow: each SModel Element (i.e., SElements) / Parameter (i.e. SParameter) is mapped to a set of PModel Elements (PElements) / PModel Parameters (PParameters). On the other hand, in the PModel, and each PElement/ PParameter is mapped to a set of SElements / SParametes. An example of mapping table is shown in Table 1. An alternative example of mapping representation is shown below:

PModel(PElements) ={(Serv2)} ↔ SModel(SElements) = {(Receive Client Request), (Add Selection Criteria), (Retrieve Catalogue)}

V. STUDYING THE IMPACT OF DESIGN PATTERN ON SOA PERFORMANCE MODEL

Similar to any other enterprise application, SOA systems can be improved by design changes. Some of these changes can be applied through design patterns chosen by software designer. In this section we provide a technique for tracing the impact of SOA design model changes on the performance model produced by model transformations, as discussed in Section IV. In other words, we incrementally propagate the changes from SModel to PModel. The first step is to identify the types of changes to a SOA design model suggested by a SOA design pattern. Therefore, in this section we first define a manual processes to extract the instructions of applying a pattern from its description.

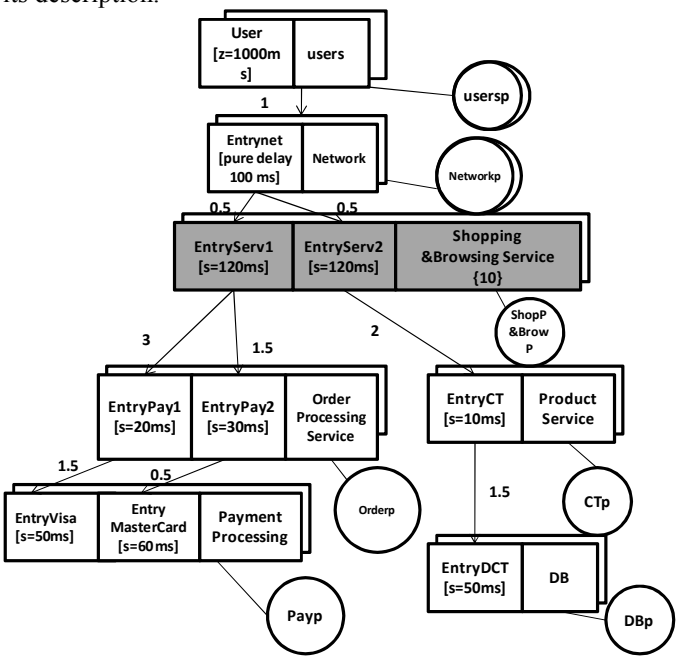

Figure 4 : LQNS Model for Shopping and Browsing SOA

Table 1: Mapping Table Example

SModel	Type (SElement /SParameter)	PModel	Type (PElement /PParameter)
Activities = {(Receive Client Request),(Add Shipping), (Check out Cart), (Identifying Payment Type)}	Activity, Element, «PaStep»	Serv1	Entry, Element
Activities Host Demands = {(Receive Client Request),(Add Shipping), (Check out Cart), (Identifying Payment Type)}	Parameter , Activity Host Demand	Serv1 Host Demand = sum of Activities Host Demands = {(Receive Client Request),(Add Shipping), (Check out Cart), (Identifying Payment Type)}	Parameter
Activities = {(Receive Client Request), (Add Selection Criteria), (Retrieve Catalogue)}	Activity, Element, «PaStep»	Serv2	Entry, Element
Deployment Artifact = Shopping &Browsing	Deployment Artifact ,Element , «Artifact»	Shopping &Browsing	Task, Element
Deployment Node = Order Proc	Deployment Node, Element , «GaExecHost»	Order	Host, Element
Deployment Node = Product Catalogue Proc	Deployment Node, Element, «GaExecHost»	Product Catalogue	Host, Element

We call these instructions *the applications rules*. Once the application rules are identified, the impact of the design pattern on design model elements and parameters can be realized.

We also propose to include a mapping table generation activity in the model transformation used to obtain the PModel from the SModel. Using the mapping table and the annotated performance elements from the design model, the affected performance elements and parameters in PModel can be easily identified. In the end, the changes are applied to the performance model. The whole process helps the system designer to avoid the reconstruction of the performance model from scratch for each design model modification.

A. Patterns Application Rules and Smodel Modification (ΔSmodel)

Each SOA design pattern comes with application rules explaining the procedure of applying the pattern to a design model. The applications rules contain the information about the modifications that should be made to the SModel for the specific purpose of the pattern. The first step toward tracing the change propagation to PModel is to extract these rules in a formatted style. Since the step needs human interpretation, we consider it as a manual process which can be done by the system designer. In general, we categorize the modifications to a UML model suggested by a SOA pattern into structural and non-structural:

- Structural modifications category includes adding, removing a new SModel element or altering an existing one.
- Non-structural modifications category includes changes which are made into the SModel parameters (e.g. number of calls or service time in an activity) such as increasing or decreasing the values.

As mentioned before, a system analyst extracts the pattern application rules from the applying instruction given in the SOA pattern description. Once done, the application rules associated to the pattern can be stored for future reuse. We explain the process in this section in more details by providing an example. We use the Functional Decomposition design pattern description and discuss the process.

Functional Decomposition Pattern[1]

In general, this pattern discusses how to design a service solution for a large business problem without having to build a standalone body of solution logic.

Problem: To serve a large and complex business task, a corresponding amount of solution logic (service) needs to be created, resulting in a self-contained application with traditional governance and reusability constraints.

Solution: The large business task should be broken down into a set of smaller, related tasks, leading to a corresponding set of smaller, related services which satisfy those tasks.

Application Instruction: The large services in SOA which carry a very high load should be identified and be broken down into a set of smaller services, each satisfying a functionality of the larger service.

As seen in this pattern description, the heavy loaded services in the SOA should be identified and broken down into smaller services. For this purpose, a new service (possibly including its hardware infrastructure) is added into the SOA system and a subset of activities from another service is moved to the newly created service. Therefore the application rule for this design pattern is defined as:

- **Condition:** If there is any large service within the system
- **Actions:** Split the service in two and allocate the responsibility associated with one of the halves to a newly created runtime instance; allocate the runtime instance to the same node. Optionally, add a new deployment diagram node into deployment diagram and allocate the new component to it.

The target of change for this specific SOA pattern is the deployment diagram in SModel. We define the set of SModel modifications as follows:

$Modifications_{SModel} = \{SEM_1, ..., SEM_n\} U \{SPM_1, ..., SPM_n\}$
where SEM_n/SPM_n*:modification to a SModel element /parameter*

In the case of the Functional Decomposition pattern the modifications are defined as:

$Modifications_{SModel} = \{SEM1, SEM2, SEM_3\}$
SEM_1: *Creating a new «PaRunTInstance» and artifact in behavioral diagram*
SEM_2: *Moving the «PaStep»s associated to the service it supposed to be moved into the newly added «PaRunTInstance»*
SEM_3: *Creating a new «Artifact» in the deployment diagram (Attached to the same «GaExecHost») containing the newly added instance.*

B. Pmodel Change (ΔPmodel)

The mapping table suggested to be created during the model transformation (Section IV.D) is used in this section for identifying the impacts on the PModel. The SModel $Modifications_{SModel}$ contains the set of affected SModel elements and parameters, while the mapping table contains the mapping between SModel and PModel parameters and elements (an example of this table is shown in Table 1). Therefore using the mappings, the affected PModel elements and parameters are detected. We define the PModel modifications as follows:

$Modifications_{PModel} = \{PEM_1, ..., PEM_n\} U \{PPM_1, ..., PPM_n\}$
where PEM_n/PPM_n*: Modification to a PModel element/ parameter.*

The identification of $Modifications_{PModel}$ based on the mapping table and $Modifications_{SMode}$ proceeds as follows. SEM_3 indicates adding a new *«Artifact»* into the deployment diagram and splitting the existing *«Artifact»* attached to target *«GaExecHost»* into two by moving a part of it to newly added *«Artifact»*. Since in the mapping table (Table 1), the deployment diagram *«Artifact»* is mapped to the PModel tasks, we conclude that this change is mapped to creating a new LQN task on the same host. We name this PModel change as PEM_1 and PEM_2. As an update to our mapping table, a row indicating the new *«Artifact»* in deployment diagram and its mapped task in LQN is added to the table. SEM_1 and SEM_2 indicate the creation of a new

«PaRunTInstance» and artifact in behavioral diagram and moving the *«PaStep»* associated to the service supposed to be moved into the newly added *«PaRunTInstance»*. Assuming the Browsing service needs to be moved, it can be understood from Table 1 that the following activities stereotyped with «PaStep» are associated to the Browsing service: *{(Receive ClientRequest), (AddSelectionCriteria), (RetrieveCatalogue)}*. Then, the mapping table (Table 1) shows that these SModel elements are mapped to "*EntryServ2*" entry in PModel. So the next PModel change (PEM_3) is defined as moving the entry "*EntryServ2*" into the newly added task. Therefore $Modifications_{PModel}$ is formed as:

$Modifications_{PModel} = \{PEM_1, PEM_2, PEM_3\}$
PEM_1 = Adding *a new task associated to new SModel deployment node into PModel*
PEM_2 = Connecting *the newly added task to the host of the task supposed to be decomposed (Running on same CPU).*
PEM_3= Moving *the entry associated to the SModel artifact supposed to be moved ("EntryServ2" in this case) and all its connections to the newly added PModel task.*

The modified PModel due to SOA design pattern change is shown in Figure 5. The affected elements are shaded in gray.

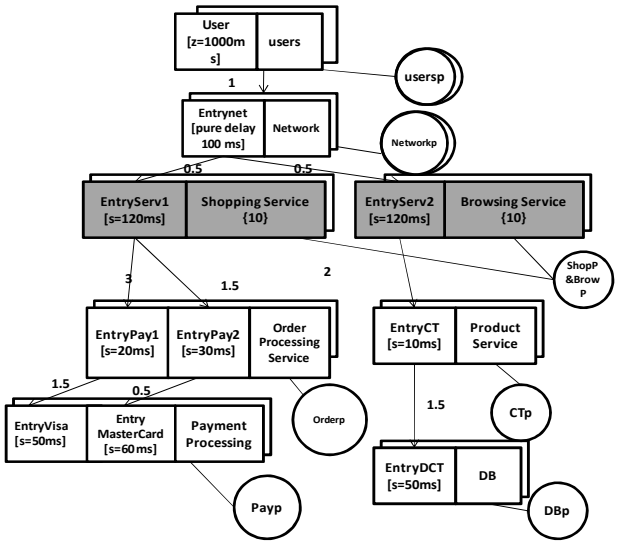

Figure 5 : Modified LQN after applying Function Decomposition Design Pattern

C. Impact of Security Design Patterns

Distributed systems such as SOA systems have many users and have to meet different and sometimes conflicting non-functional requirements, such as security and performance. For example, a highly secure system may pay a performance price compared to an unsecure system, due to the extra security checks it must do. System designers need to make choices between competing design solutions in order to satisfactorily balance system requirements. The technique in this paper can help them to evaluate the performance effect of a security design pattern applied to SOA. In this section, we apply a security pattern to our case study and show the application of our technique. Secure Sockets Layer (SSL) is a cryptographic protocol that provides communications security over the Internet. The requirements for applying such kind of capability (application rules) as a design pattern can be expressed as follows [1] [15]:

Problem: Communication between client (browser) and the service provider over the internet needs to be secure
Solution: A protocol allowing for encryption/decryption of client requests and server responses needs to be used on both client and server sides.
Application Instruction: Activities are defined for both client and server to encrypt the requests before transmission through Internet and decrypt the received responses.

The impact to the SModel due to this pattern include both elements and parameters modifications. Based on the applications rules, the SModel changes are defined as follows:
$Modifications_{SModel} = \{SEM_1, SEM_2\} \cup \{SPM_1, SPM_2\}$
SEM_1: Creating *two new activities (with «PaStep») in the user «PaRunTInstance»: one to encrypt the client request before transmitting over Internet and the other one to decrypt the received responses coming over Internet.*
SEM_2: Creating *two new activities (with «PaStep») in the Shopping&Browsing «PaRunTInstance»: one to decrypt the client request transmitted over Internet and the other one to encrypt the responses before it was transmitted over Internet.*
SPM_1: Annotating *the newly added activities to the user «PaRunTInstance» with the host demand parameter of each.*
SPM_2: Annotating *the newly added activities to the Shopping&Browsing «PaRunTInstance»with the host demand parameter of each.*

The modified SModel with SSL is shown in Figure 6.

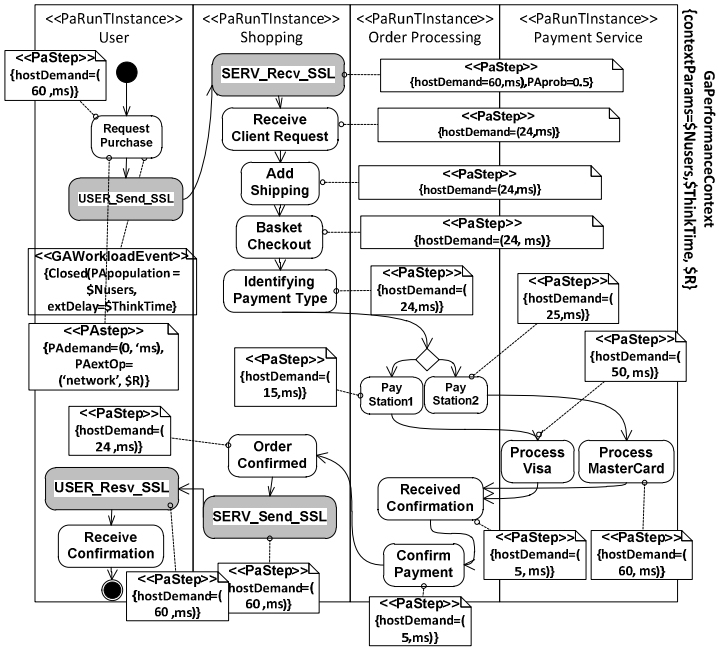

Figure 6 : Shopping Service Behavioral model secured with SSL

Based on the $Modifications_{SModel}$, the following new activities are added into the user swimlane stereotyped *«PaRunTInstance»*: USER_Send_SSL and USER_Resv_SSL. Also two activities are added to the Shopping swimlane: SERV_Recv_SSL and SERV_Send_SSL. Therefore, the row in the mapping table (Table 1) associated to:

Activities = {(ReceiveClientRequest), (AddShipping), (CheckoutCart), (IdentifyingPaymentType)}

is modified as follows (modifications shown in bold):

{**(SERV_Recv_SSL),**
(ReceiveClientRequest),(AddShipping), (CheckoutCart), (Identifying Payment Type), **(SERV_Send_SSL)**}**.**

These activities are mapped to EnteryServ1 in Table 1. Also, as shown in another row of Table 1, the *hostDemand* Parameter of *EnteryServ1* is calculated by adding the host demand of the activities together. The same scenario is repeated for the *user «PaRunTInstance»* by updating its browser processing time (set as zero in the initial model) by the host demand of the newly added activities .The impact on PModel due to this pattern is only on host demand parameters of Shopping&Browsing task there is element change. Therefore, the $Modifications_{PModel}$ is formed as below:

$Modifications_{PModel} = \{PPM_1, PPM_2\}$

PPM_1 = Updating the host demand parameter associated with the "EntryServ1" with a new value calculated by adding the host demand of the newly added activities to the old value.
PPM_2 = Updating the host demand parameter associated with the "EntryUser" with a new value by adding the host demand of the newly added activities to the old value.

The modified LQN model elements due these changes are shown in Figure 7. The added SSL times to "EntryUser" and "Entry Serv1" are underlined.

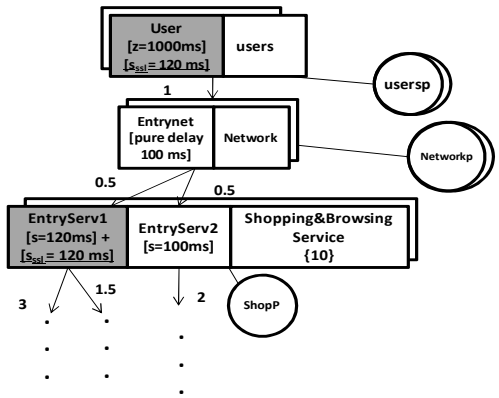

Figure 7 : LQN model after applying SSL

D. Impact on Performance Results

Once the PModel is modified based on the SOA design pattern, it can be solved to obtain the new performance results. Table 2 and shows the results of the performance analysis before and after applying the Functional Decomposition and SSL security design patterns for 50 users. As the results are compared in Figure 8 and Figure 9 (for users varies from 1 to 120), a very slight improvement is seen in the system throughput and response time after applying the Functional Decomposition pattern. More specifically, the initial performance model has been solved considering that the Shopping&Browsing task was multithreaded, with a maximum of 10 threads in the pool. All the tasks below had an unlimited thread pool, with the exception of PaymentProcessing, which accepted a maximum of 10 concurrent requests. When applying the Functional Decomposition pattern, we split the Shopping&Browsing task in two tasks, each with its own thread pool of maximum 10 threads. Both tasks were allocated to the same processor, so no new hardware resources were added. A deeper analysis shows that the performance improvement is due to the increased concurrency level (the two tasks were running together 20 threads compared to 10 before). If, however, the number of threads of each new task was limited to 5, no performance improvement was observed compared to the initial system. Furthermore, adding a new processor for the new task does not improve the performance at all, as the processor for the Shopping&Browsing task is underutilized. This shows that the performance improvement of design patterns depends very much to the performance context in which the pattern is applied. Without the ability of doing performance analysis, it would be impossible to estimate the performance effect of a design pattern.

Table 2: Performance Analysis Results for 50 users before and after applying Function Decomposition Design Pattern and SSL

Initial Model Performance Results			
Task	**Throughput (req/sec)**	**Utilization (%)**	**Response Time (sec)**
Users	0.007458	42.54	6.703
Shopping & Browsing	0.007458	9.937	
After applying Function Decomposition Design Pattern			
	Throughput (req/sec)	**Utilization (%)**	**Response Time (sec)**
Users	0.007548	42.45	6.624
Shopping	0.003774	2.17	
Browsing	0.003774	0.993	
After applying SSL to the Initial Model			
	Throughput (req/sec)	**Utilization (%)**	**Response Time (sec)**
Users	0.006152	43.11	8.127
Shopping & Browsing	0.006152	9.94	

As expected, the SSL security pattern has a negative impact on both the system throughput and response time of the initial model (before applying the Functional Decomposition). The effect depends on the ratio between the SSL overhead and the actual work of the entry to which it is added, as well as on whether the respective task is the bottleneck or not. Adding an overhead to the bottleneck task enhances the loss in throughput and response time more than in the case when the task is lightly loaded. Again, without a performance model would be impossible to assess the actual effect of the security

overhead on the system performance. By using the technique proposed in this paper (incremental change propagation), it is faster to apply a various range of SOA design patterns and to assess their performance impact in a time and cost effective manner.

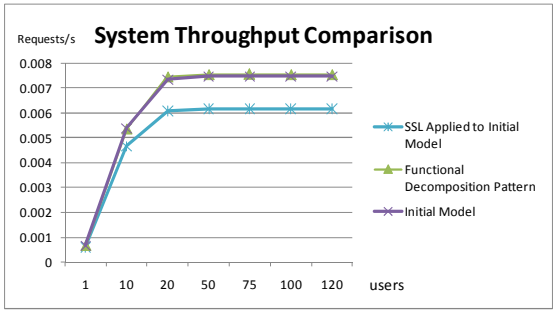

Figure 8 : System Throughput Comparison

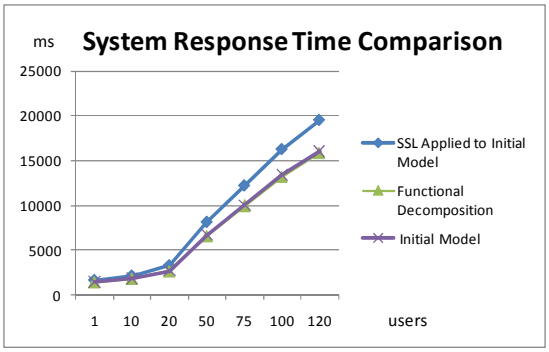

Figure 9 : Response Time Comparison

VI. Conclusions and Future Work

This paper proposes an approach to study the performance impact of SOA design patterns. Design patterns suggest solutions for architectural, design and implementation problems, but these changes also affect performance and other non-functional properties. The main idea is to propagate the changes due to the application of SOA design patterns from the SModel to the corresponding PModel by incremental model transformation. Since the change introduced by a design pattern is well understood and limited to a small subset of elements, we expect that the incremental change propagation will speed up the process and will allow us to evaluate a large range of patterns in a time and cost effective manner.

Future work will fully develop and implement a general approach for incremental change propagation, based on the concept of mapping table from the SModel to Pmodel. We are planning to apply the proposed technique to many SOA patterns from literature in order to find the kind of changes that need to be propagated. We also plan to automate the incremental change propagation from SModel to PModel by using traceability links from PUMA. Last but not least, our goal is to be able to screen automatically for improvements.

Acknowledgment

This research was partially supported by Discovery grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Centre of Excellence for Research in Adaptive Systems (CERAS).

References

[1] T. Erl, *SOA Design Patterns*, Boston, MA, Prentice Hall PearsonPTR , 2009

[2] A. Rotem-Gal-Oz, E.Bruno, and U. Dahan, *SOA Patterns (Early Access Edition),* Manning Publications, June 2007

[3] C.M.Woodside, D. C. Petriu, D.B. Petriu. H. Shen, T. Israr, J. Merseguer, "Performance by Unified Model Analysis (PUMA)", *Proc. ACM Workshop on Software and Performance WOSP'05,* Palma, Illes Balears, Spain , 2005, pp. 1-12

[4] D.C. Petriu, "Software Model based Performance Analysis", in Model Driven Engineering for distributed Real-Time Systems: MARTE modelling, model transformations and their usages (J.P Babau, M. Blay-Fornarino, J. Champeau, S. Robert, A.Sabetta, Eds.) , ISTE Ltd and John Wiley & Sons Inc., 2010..

[5] Object Management Group, "Service oriented architecture Modeling Language (SoaML)", Version 1.0, formal/2009-11-02, Dec. 2009.

[6] Object Management Group, "A UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems)", Version 1.0, formal/2009-11-02, Dec. 2009.

[7] D.C.Petriu, H.Shen, "Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications", in Computer Performance Evaluation - Modelling Techniques and Tools, (Tony Fields, Peter Harrison, Jeremy Bradley, Uli Harder, Eds.) *LNCS* 2324, pp.159-177, Springer, 2002.

[8] C. U. Smith and L.G. Williams, *Performance Solutions*. Addison Wesley, 2002.

[9] V. Cortellessa, A.D. Marco, R. Eramo, A. Pierantonio, C. Trubiani, "Digging into UML models to remove performance antipatterns", *Proc. of 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems*, Cape Town, South Africa, pp.9-16, 2010.

[10] D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malex , J. P. Sousa , "A framework for utility-based service oriented design in SASSY", *Proc 1st WOSP/SIPEW Int Conf on Performance Engineering*, San Jose, CA, pp. 27-36 , 2010.

[11] T. Parsons, J.Murphy, "Detecting Performance Antipatterns in Component Based Enterprise Systems", *Journal of Object Technology*, Vol. 7(3), 2008.

[12] J. Xu , "Rule-based automatic software performance diagnosis and improvement*", Proc 7th Intl Workshop on Software and Performance*, Princeton, NJ, pp. 1-12, 2008.

[13] IBM , "Modeling with SoaML, the Service-Oriented Architecture Modeling Language" , IBM , January 7th , 2010 [Last time accessed March 15th ,2011]

[14] Cortellessa, V., Mirandola, R.: Deriving a Queueing Network based Performance Model from UML Diagrams. In: Proc. of 2nd ACM Workshop on Software and Performance, Ottawa, Canada (2000) 58–70.

[15] C.M. Woodside, D.C. Petriu, D.B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J.M. Bieman, S.H. Houmb, J.Jürjens, "Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML Models", The Journal of Systems and Software Special Issue WOSP'2007, Vol.82, pp.56–74, 2009, DOI:10.1016/j.jss.2008.03.067