Avestia Publishing
International Journal of Electrical and Computer Systems
Volume 1, Issue 1, Year 2012
ISSN: 1929-2716
Article ID: 005, DOI: 10.11159/ijecs.2012.005

Using Aspects for Platform-Independent to Platform-Dependent Model Transformations

Mohammad Alhaj, Dorina Petriu
Department of Systems and Computer Engineering, Carleton University
1125 Colonel By Drive, Ottawa ON Canada, K1S 5B6
malhaj@sce.carleton.ca; petriu@sce.carleton.ca

Abstract– *This paper presents an aspect-based approach for realizing a transformation from platform-independent to platform-dependent models in the context of a model transformation chain that generates queueing-based performance models from UML design models of service-oriented applications. The purpose of generating such performance models is to evaluate the performance of the system under development in the early software lifecycle phases, in order to insure that it will meet the performance requirements. The paper presents the model transformation chain PUMA4SOA, which transforms automatically a UML model of a service-oriented architecture (SOA) system extended with MARTE performance annotations into an intermediate model, Core Scenario Model (CSM), which in turn is used to generate a Layered Queueing Network (LQN) performance model. Aspect-oriented modeling is used to represent different services offered by the underlying SOA platform to the SOA application. The paper discusses and compares different alternatives for composing the platform aspect models with the platform-independent model (PIM) of the application throughout the model transformation chain.*

Keywords: SOA, Software Performance Engineering, Aspect-oriented modeling, Model transformation, Performance Analysis

© Copyright 2012 Authors - This is an Open Access article published under the Creative Commons Attribution License terms (http://creativecommons.org/licenses/by/2.0). Unrestricted use, distribution, and reproduction in any medium are permitted, provided the original work is properly cited.

1. Introduction

Service-Oriented Architecture (SOA) is a software architectural approach that seeks to develop and deploy business applications as a set of reusable services (Earl, 2005). The developers of SOA applications would benefit from the ability to estimate the performance of the proposed design (in terms of response time, throughput and utilizations) in the early development phases, as promoted by the Software Performance Engineering methodology (Smith, 1990). This requires deriving (by hand or automatically) a performance model from the software design model and deployment information. In the case of UML designs, performance annotations are added to the software model with the help of the MARTE profile, standardized by OMG (OMG, 2009). An example of a model transformation chain which takes as input a UML software design model with performance annotations and generates automatically various types of performance models (such as queueing networks, Petri nets, simulation, etc.) is called PUMA - Performance for Unified Modeling Analysis (Woodside et al., 2005). The authors of this paper, who were also contributors to PUMA, have proposed an extension called PUMA4SOA (Alhaj and Petriu, 2010), which adds new capabilities for evaluating the performance of SOA systems. The main differences between PUMA and PUMA4SOA stem from the kind of design models taken as input and the separation between the Platform-Independent Model (PIM) of the SOA application, and the Platform-Specific Model (PSM) which incorporates platform details needed for performance evaluation. We account for platform effects similar to (Selic, 2008), considering that the underlying platform offers a set of services or operations to the applications running on top of it. This allows for the rapid composition of a given PIM with multiple platform models, which allows us to evaluate its performance for different platforms.

This paper focuses on modeling as aspects the platform operations details required for performance analysis and considers alternate ways for composing the aspects with the application model. This paper extends a previous conference paper (Alhaj and Petriu, 2012) with a performance-completion feature model for representing the variability in the service platform, an approach for composing multiple

aspects, a description of aspect composition at CSM level and a section on performance analysis using the LQN model.

The paper considers three alternative approaches for composing the aspects model representing platform operation with the application PIM, as shown in Figures 1, 9 and 12 and discussed later in the paper. In Figure 1, the starting point of PUMA4SOA is the UML design model with MARTE annotations, which includes three parts: 1) *application PIM*, 2) *deployment diagram*, and 3) *platform aspect models*. The application PIM represents the software design using three views: a) *workflow model* describing the business process, b) *service architecture model* representing the hierarchy of underlying services, and c) *service behaviour models* representing the execution steps of the invoked services. The deployment diagram describes the allocation of software artifacts to processors, needed for the performance model derivation.

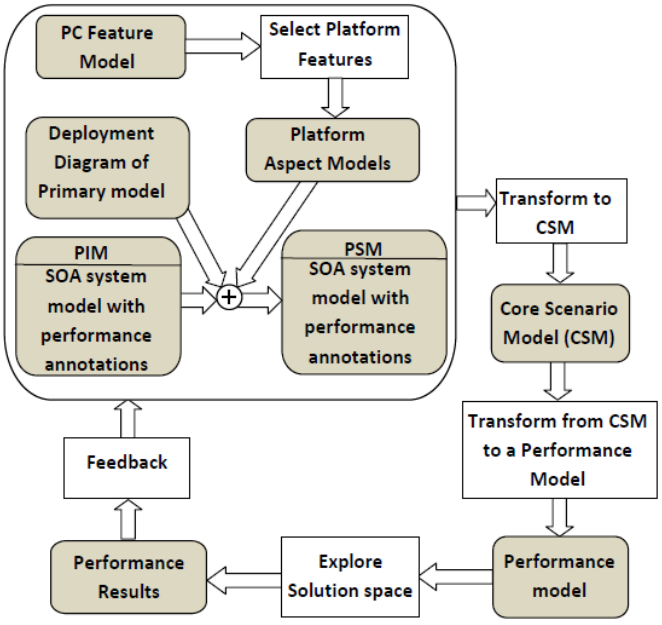


Fig. 1. PUMA4SOA approach: Aspect composition at UML level

A feature model called Performance Completion (PC) feature model represents the choices in platform characteristics, as described in section 2. Platform aspect models represent the structure and behaviour of different service platform (such as service invocation and service discovery) in a generic form. A recent OMG standard profile SOA Modeling Language (SoaML) (OMG, 2012) is also used to express the service architecture model. SoaML extends UML with the ability to define the service structure, behaviour, dependencies, and capabilities.

Aspect oriented modeling (AOM) is used to generate the platform specific model (PSM) by weaving platform aspect models into different locations in PIM. In principle, AOM uses two types of models: the *primary model*, which is the core design model, and a set of *aspect models* describing concerns that crosscut the primary model. Aspect *composition* (a.k.a. *weaving*) in AOM is performed in a number of steps: 1) defining the *point cut* rules (a set of conditions applied to the primary model to identify the join points), 2) identifying the *join points* (i.e., location in the primary model where an aspect composition occurs), 3) instantiating a *context-specific aspect model*, by binding the parametric values of the *generic aspect* to concrete values related to the context of the join point in the primary model, and 4) performing the *aspect composition* at the join points (France et al., 2004), (Woodside et al, 2009).

After obtaining the PSM, a model transformation is performed to generate the *Core Scenario Model (CSM)*. CSM is an intermediate language which aims to bridge the semantic gap between the software and performance domains during the model transformations (Petriu and Woodside, 2007). CSM reduces the complexity of mapping different views from design model to the performance model and helps checking the consistency of the generated models. The CSM is then transformed to one of different types of performance models, such as Layered Queuing Network (LQN) (Woodside et al., 2005), which is solved to produce the performance results of the system.

PUMA4SOA provides the ability to transform PIM to PSM based on the AOM approach at three modeling levels: UML, CSM and LQN. Although the platform aspect models are originally defined in UML, they can be transformed separately and composed into the primary model at different levels, i.e. UML (Fig. 1), CSM (Fig. 9), and LQN (Fig.12) as discussed in the paper.

The paper is organized as follows: Section 2 introduces the concept of performance completion feature model; Section 3 describes the UML input models, illustrated with a Purchase Order system example; Section 4 presents and compares the aspect composition at the UML, CSM and LQN levels; Section 5 describes the approach for performing multiple aspect compositions; Section 6 discusses the performance results of the case study; Section 7 presents related work and Section 8 the conclusion and directions for future work.

2. Performance Completion Feature Model

PUMA4SOA uses a so-called *Performance Completion (PC) feature model* to represent the variability in service platform which may affect the system's performance. PC feature model provides the choice to select between multiple platform characteristics (each represented by aspects) based on the business requirements. The concept of "performance completions" was introduced by (Woodside et al., 2002) to close the gap between abstract design models and external platform factors. It was also used in (Happe et. al, 2010) and (Tawhid and Petriu, 2011) to define the variability in platform choices, execution environments, types of platform realizations and other external factors which have impact on

the system's performance. A feature model is normally represented using feature modeling notations and grammars, such as the one introduced in (Batory, 2005). However, in order to maintain the model consistency with the UML input design models (PIM, Deployment diagram and Aspect models), we use a UML class diagram extended with stereotypes to represent the PC feature model.

The term "feature" was introduced in (Kang et al., 1990) and defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a software system". In PUMA4SOA, the PC feature model represents a classification of different SOA platform characteristics and their relationship in a hierarchical format. The developer can select from the PC feature model those platform features needed for the application, according to the system requirements. (Please note that in our work, we build and aspect model for each feature, so the selection of a certain feature implies the selection of an aspect).

The root class, which represents the service platform is decomposed into a set of feature groups, some mandatory and others optional. A feature group holds in turn a set of features, which may have sub-features. The relationships between a feature and its sub-features can be categorized as:

1) *Mandatory*: represented as a solid circle, used when a feature group or a feature is required;

2) *Optional:* represented as non-solid circle, used when a feature group or a feature is optional.

3) *Or:* represented as a solid arc, used when at least one of the feature groups or features must be selected.

4) *Alternative (XOR):* represented as a non-solid arc, used when one of the feature groups or features must be selected.

In addition to the relationships between parents and child nodes, cross-tree feature constraints are allowed. For instance, "A requires B" means that the selection of A implies the selection of B.

Fig. 2 shows the PC feature model used for the Purchase Order system example presented in the next sections. There are three mandatory feature groups which are required: *Operation, Message Protocol*, and *Realization*, and two optional feature groups: *Communication* and *Data Compression*. The relationships between the feature groups and their features are all alternative (XOR) with exactly-one-of feature selected. Although the dependencies between the sub-features are not shown in the model, some features, such as *Operation, Message Protocol* and *Realization* are dependent. For example selecting one of the *Operation* sub-features, such as *Invocation*, requires also selecting one of the message protocols (Http or SOAP) and one of the realizations (WebService, REST, etc.).

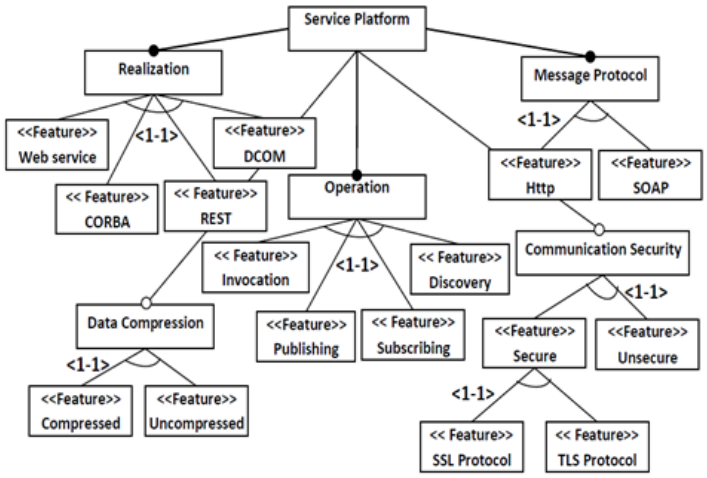


Fig. 2. Platform Completion (PC) Feature Model

3. Example of PUMA4SOA Input Model: Purchase Order System

PUMA4SOA begins with developing the Platform independent model (PIM). The top level model is the business workflow model represented as UML activity diagram(s). Fig. 3 illustrates the workflow of a simple purchase order system. The workflow starts when the sales server receives the purchase order (PO), followed by the parallel processing of invoice and scheduling activities, and then by the *ShipProduct* activity. An AD partition marked with the MARTE stereotype «*PaRTimeInstance*» indicates the runtime process/component responsible for the execution of the respective activities – in this case, the workflow *POProcess*. The stereotype «*GaWorkloadEvent*» indicates that a closed workload is associated to this business process, with a number of simultaneous users (i.e., population) expressed by the MARTE variable *Nusers*; each user has a think time of 3000 ms, given by the attribute *extDelay*.

Fig. 4 illustrates the service architecture model expressed with the help of SoaML stereotypes to define the participants, their binding rules and their capabilities. Components stereotyped as «*Participant*» indicate parties that provide or consume services. Ports are stereotyped with «*Request*» to indicate the consumption of a service, or with «*Service*» to indicate the offered service.

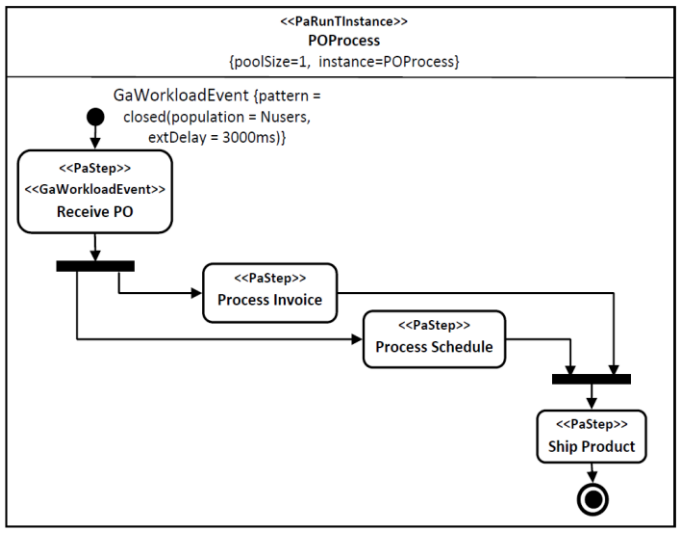


Fig. 3. PIM: Workflow model.

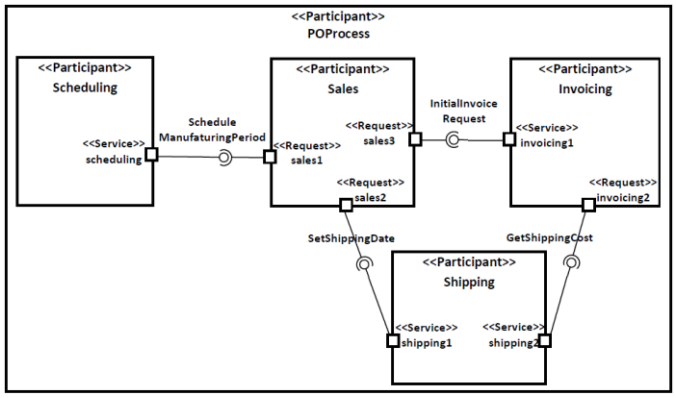


Fig. 4. PIM: Service Architecture model.

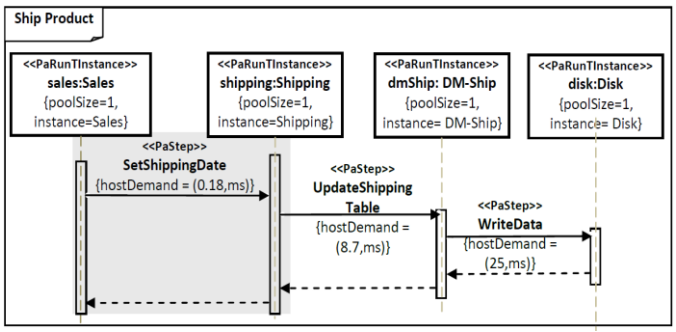


Fig. 5. PIM: Service Behavior model.

The details of each activity in the workflow model from Fig. 3 are described in a refined service behaviour model. However due to the space limitations, only *Ship Product* activity is presented in Fig. 5. The shadowed area indicates the invocation by *Sales* of the *SetShippingDate* service offered by the component *Shipping,* which uses a data management instance to store the data on a disk server. From a performance point of view, a UML behaviour model drawn as an activity or sequence diagram can be considered that it represents a Scenario composed of Steps related by predecessor-successor relationships (sequence, branch, merge, fork, join, etc.). MARTE has two kinds of step stereotypes: «*PaStep*» representing the execution of an activity or an operation invoked by a message and «*PaCommStep*» for the communication costs involved with passing a message between components. Examples of «*PaStep*» attributes are *hostDemand* giving the value and unit for the step execution time and *prob* giving the probability for the optional steps. An example of «*PaCommStep*» attribute is *msgSize* giving the size and unit.

For performance modelling we need also a deployment diagram as in Fig. 6, which shows the allocation of active software components (indicated with the MARTE stereotype «*SchedulableResource*») to the hardware nodes («*GaExecHost*» and «*GaCommHost*»).

A platform aspect model describes the structure and behaviour of a platform operation in a generic format. In this paper, the SOA platform is in fact the middleware used for service invocation, discovery, publishing, etc. Each middleware operation is represented by a different aspect model which has multiple views. Fig. 7a illustrates the generic deployment diagram and Fig. 7b the sequence diagram of the service invocation aspect model.

The aspect models are defined independently of the SOA application with which they will be composed, so generic parameters are used. As a naming convention, the vertical bar '|' indicates a generic role name (France et al., 2004). For example, the |*client* component acts as service consumer, while the |*provider* component as service provider. The middleware on both sides defines two generic roles: the |*xmlParser* which helps to parse and marshal/unmarshal data and the |*SOAP* stub for sending/receiving messages using the SOAP protocol.

4. PIM to PSM Transformation

4.1 Aspect Composition at the UML Level

Fig. 1 shows the alternative where the aspect composition takes place in the UML design model. The models described in Fig. 3 to Fig. 6 represent the purchase order system in UML extended with MARTE and SoaML stereotypes. In our example, the primary model is the platform independent model (PIM) in Figs 3, 4 and 5, while the platform aspect model represents the generic structure and behaviour of the platform operation "service invocation" in Figs. 7a and 7b.

The first step in the AOM approach is to define the point cut rules. Since our example aspect (i.e., service invocation) applies only to SOA services, the point cut rule must be able to locate the service calls in the primary model. In the case of Service Invocation, we are looking for a UML element of type *Message* in a service behaviour model that calls a service stereotyped «*Service*» in the architecture model. (In our example, there are four services identified in Fig. 4).

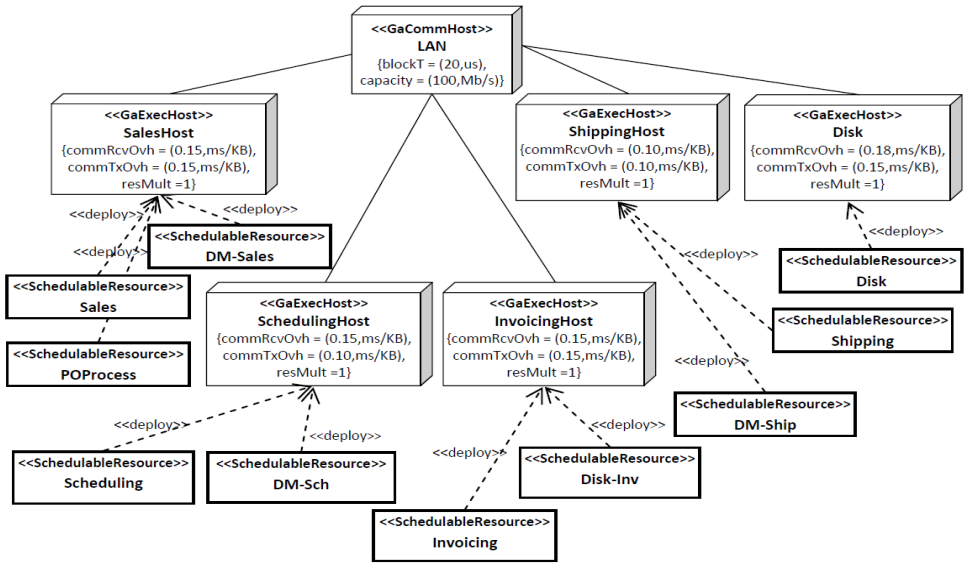
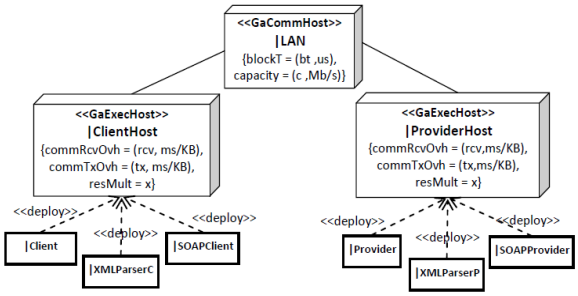
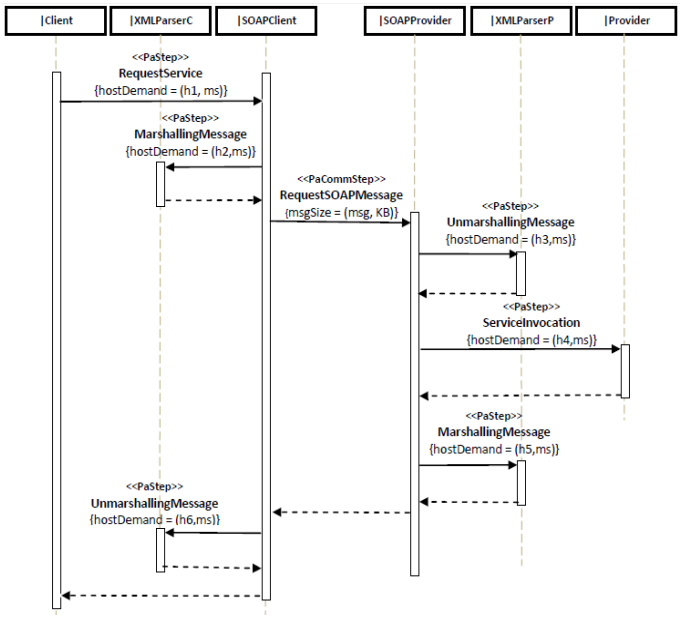


Fig. 6. Deployment diagram



a) Generic deployment diagram for Service invocation



b) Generic behaviour diagram for Service invocation

Fig. 7. Platform aspect model for "Service invocation"

The next step looks for the join points (i.e., the places in the model where to weave the aspects) by applying the point cut rules to PIM. The result of this step will identify all the messages in behaviour models that represent service calls. For instance, the highlighted message *SetShippingDate* in Fig. 5 is one of the join points.

Once the join points are identified, a context specific aspect model is generated by instantiating the generic aspect model for each join point. The instantiation requires binding the generic roles to concrete ones from the scope of the join point, and then assigning context specific performance values to «*PaStep*» attributes (processing demands, probabilities, etc.). Table 1 illustrates a sample of binding of generic roles to concrete values: some to existing PIM roles and some to newly created roles (for example, XML parsers and SOAP processing). Table 2 illustrate a sample of the binding of generic performance parameters represented by MARTE variables assigned to stereotype attributes. The context specific aspect model is not shown here due to the space limitations; however it will appear within the composed model in Fig. 8.

Table 1. Binding generic to concrete roles.

Generic Aspect	Context Specific Aspect
|Client	sales
|xmlParserC	XMLParserInv (new)
|soapClientC	SOAPInv (New)
|provider	shipping
|xmlParserP	XMLParserShip (new)
| soapClientP	SOAPShip (new)
RequestService	SetShippingDate
ServiceInvocation	SetShippingDateInvocation

Table 2. Binding generic performance parameters.

Generic Aspect attribute		Context Specific Aspect value
RequestService::hostDenamd	h1	0.18 ms
MarshallingMessage::hostDemand	h2	1.56 ms
RequestSOAPMessage::msgSize	msg	50 Kb
UnmarshallingMessage::hostDemand	h3	1.75 ms
ServiceInvocation::hostDemand	h4	0.1 ms
MarshallingMessage::hostDemand	h5	1.69 ms
UnmarshallingMessage::hostDemand	h6	1.82 ms

The final step in AOM is to weave (i.e., compose) the context specific aspect model within PIM at the join point location. Fig. 8 shows a composed model, where the *SetShippingDate* message (join point) has been replaced with the context specific aspect model (the gray area).

4.2. Aspect Composition at the CSM Level

CSM is focused on modeling scenarios, which are implicit in many software specifications. The CSM metamodel describes three main types of concepts: resources, scenarios, and workloads. A scenario is a graph of steps with precedence relationships. A step may represent a basic operation or be refined as a sub-scenario. There are four kinds of resources in CSM: *Processing Resource*, *ComponentResource*, *LogicalResource* and external resource.

PUMA4SOA provides the flexibility to perform the aspect composition at the CSM level, as shown in Fig. 9. The platform independent model (PIM) and the platform aspect model are first transformed into CSM models separately. Fig 10 presents a subset of the scenarios that constitute the CSM model of the Purchase Order system PIM: the top-level scenario and the refinement of the composed step *Ship Product*. The UML workflow model is transformed into the CSM top scenario model, whose activities are in turn refined by CSM sub-scenarios obtained by the transformation of the service behaviour model interaction diagrams into CSM.

The procedure for the aspect composition in CSM form Fig. 9 is similar to the one in UML. However, the point cuts rules cannot be defined exclusively at the CSM level, because the details of the service architecture model at the UML level is not entirely transformed to CSM. CSM is scenario-based and its metamodel mainly captures the details of the behaviour models, while many of the details of the structure models are lost during the transformation. This is one of the major drawbacks of performing aspect composition at the CSM level. To solve this issue, we need to go back to the UML level in order to use the service architecture model for identifying the services of the PO system, then to identify the steps in the CSM model that correspond to service invocations.

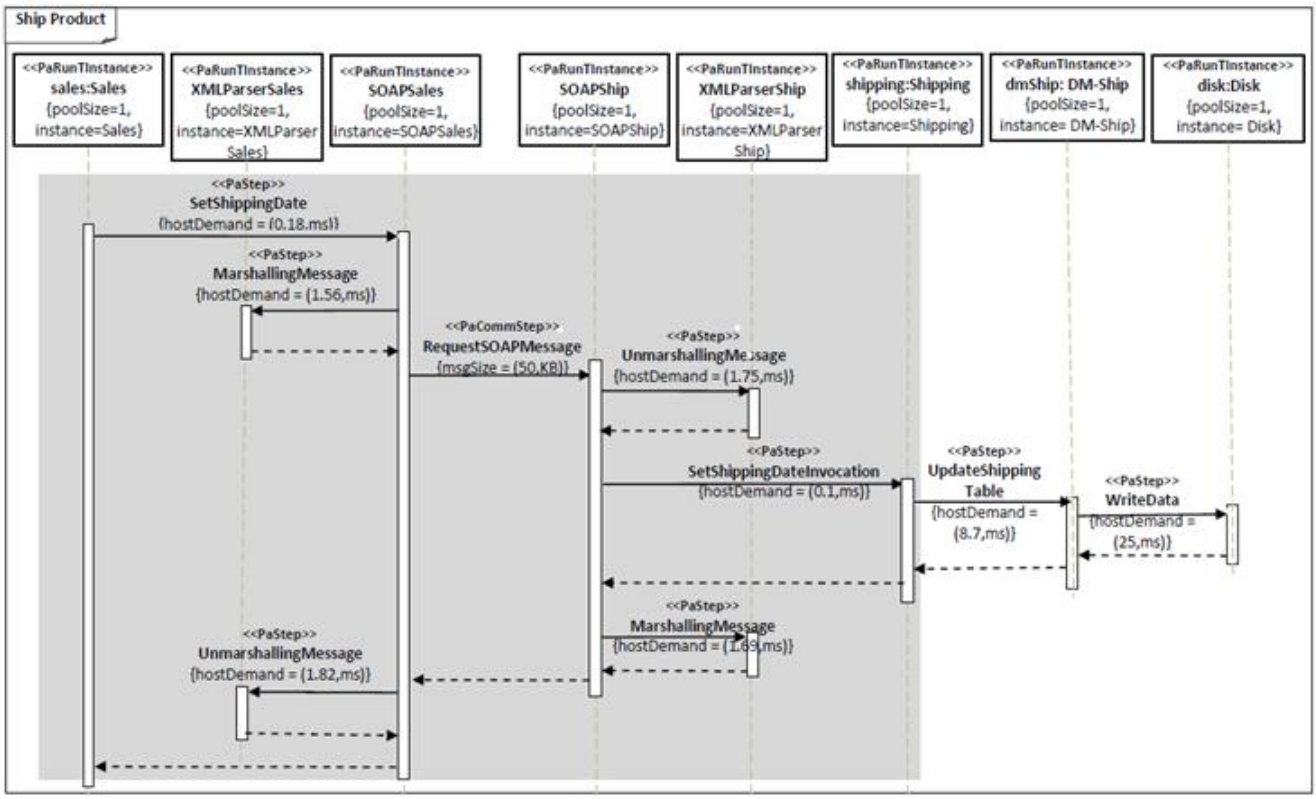


Fig. 8. Result of composition for *Ship Product* at UML level

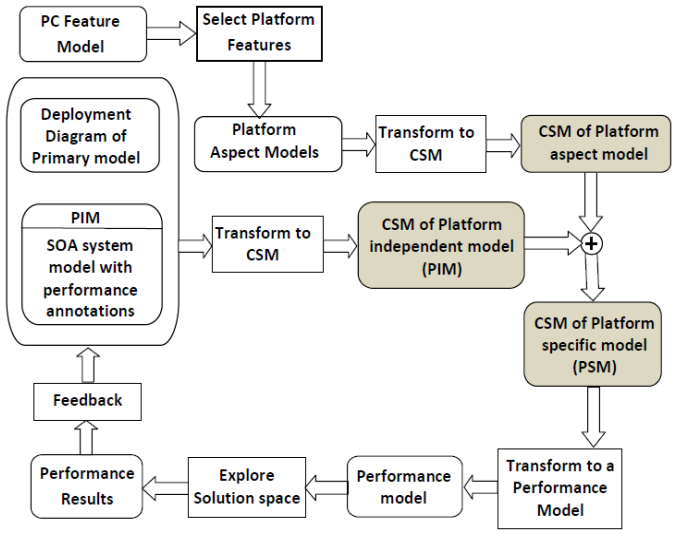


Fig. 9. PUMA4SOA approach: Aspect composition at CSM level.

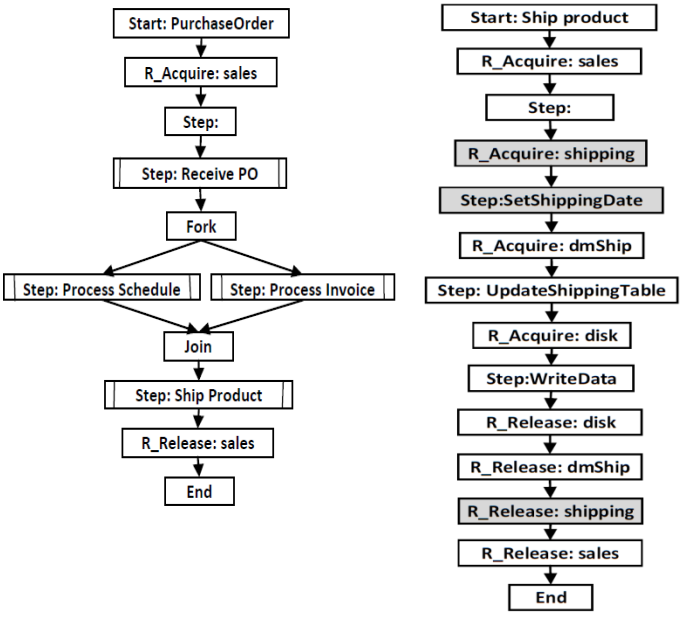


Fig. 10. Subset of the CSM model of the Purchase Order PIM.

After identifying the join points in CSM, we need to instantiate the context specific aspect model. The generic roles are bound to concrete ones, similar to the UML composition, and the performance variables are assigned concrete values. The generic roles in a CSM scenario are used in *ResourceAcquire* or *ResourceRelease* steps, which describe the acquisition and releasing of a generic resource. Table 1 and Table 2 describe the binding of the generic roles and parameters to the concrete ones for our example.

Aspect composition is the last step, done by weaving the context specific aspect model in the PIM. At CSM level, the weaving requires replacing the join point steps, the preceding *ResourceAcquire* and the succeeding *ResourceRelease* with the

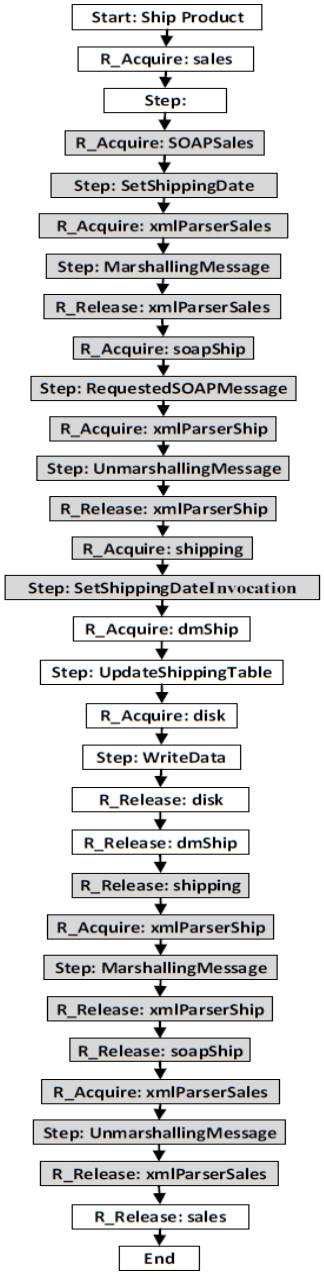


Fig. 11. Result of composition for *Ship Product* at CSM level.

context-specific aspect. Fig. 11 illustrates the composition result for *Ship Product*.

More details regarding the aspect composition in CSM can be found in (Woodside et al., 2009) and (Alhaj, 2008).

4.3. Aspect Composition at the LQN Level

The LQN model is an extended queueing network model which is able to represent nested services (Woodside et al., 1995). An LQN model is an acyclic graph of nodes called "tasks" that offer services called "entries" (see Fig.13). The

entries of a task may request services from the entries of other tasks (the requests are represented as directed arcs). LQN is used to model several types of system behaviour and inter-process communication styles.

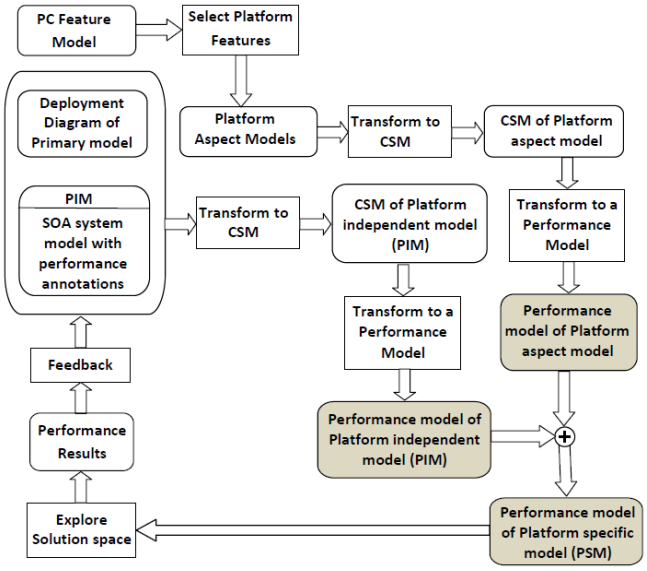


Fig. 12. PUMA4SOA approach: Aspect composition at LQN level.

Tasks represent both software and hardware devices, and allow for multi-threading and nested services. An entry may contain a graph of activities, as in Fig. 13. Graphically, the software tasks are represented with thick rectangles and the entries with attached thin rectangles. The hardware devices are represented as ellipses.

PUMA4SOA allows for performing the AOM aspect composition at the LQN level (see Fig. 12). The CSM of PIM and the generic CSM of the platform aspect model are first transformed into LQN models separately. Fig. 13 represents the LQN model of the platform-independent PO model. The workflow layer, which represents the CSM top scenario model, is transformed into a top level LQN activity graph associated with a task called *POProcess* allocated on *SalesHost*. The service layer, corresponding to CSM sub-scenario models that describe the services, is transformed into a set of tasks with their owned entries. Fig. 14 illustrates a simplified LQN of the generic platform aspect model, where the steps performed by the XML parser and the SOAP stubs in the middleware model have been aggregated. (More details are given in (Alhaj and Petriu, 2010). The |*Client* and |*Provider* tasks own a *client* and *service* entries, respectively. The middleware on the |*Client* side (|*MW1*) owns a *send* entry which sends the service request, and the |*Provider* side (|*MW2*) owns a *receive* entry which receives the service request. The network delay suffered by the message is modeled by the delay entry of the |*net* task.

The procedure for aspect composition in LQN is similar to the ones for UML and CSM levels. However, since the LQN metamodel is different, the point cut rules and join points are defined with different model elements. Note that the drawback of CSM regarding the loss of service details has been propagated to the LQN level. The LQN point cut rule for Service Invocation is checking for a request arc from a client which is requesting an entry that corresponds to a service. Note that, by construction, an entry providing a service has the same name as the service.

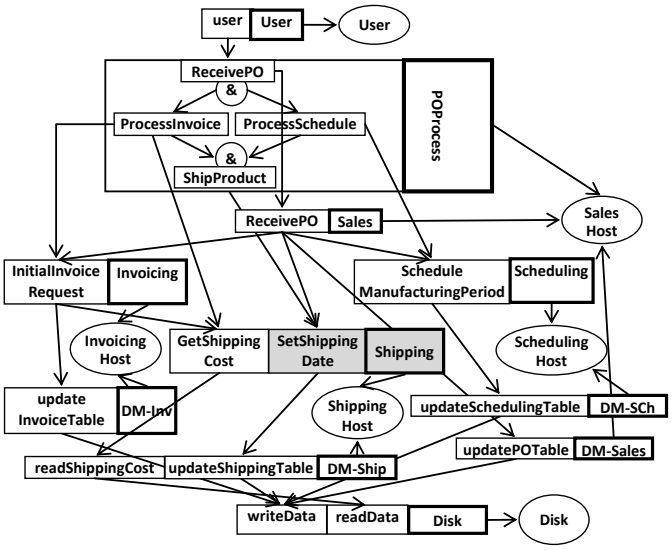


Fig. 13 The LQN model of PIM for Purchase Order.

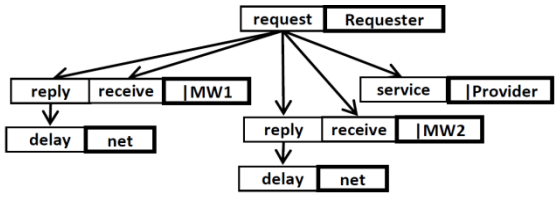


Fig. 14. The LQN generic platform aspect model for Service Invocation.

The next phase is instantiating the context-specific aspect model from the generic one, which is similar to the step performed at the UML level, where the generic roles defined in the LQN model in Fig. 14 are bound to concrete ones. The final phase is performing the aspect composition at the join points. In our Purchase Order example, we composed the context specific aspect model of the service invocation of *SetSchedulingDate* by adding new tasks with entries to the LQN, which model the middleware for all components and the network delay. The result of aspect composition in LQN is illustrated in Fig. 15 (the LQN tasks and entries that represent the platform are shaded in gray).

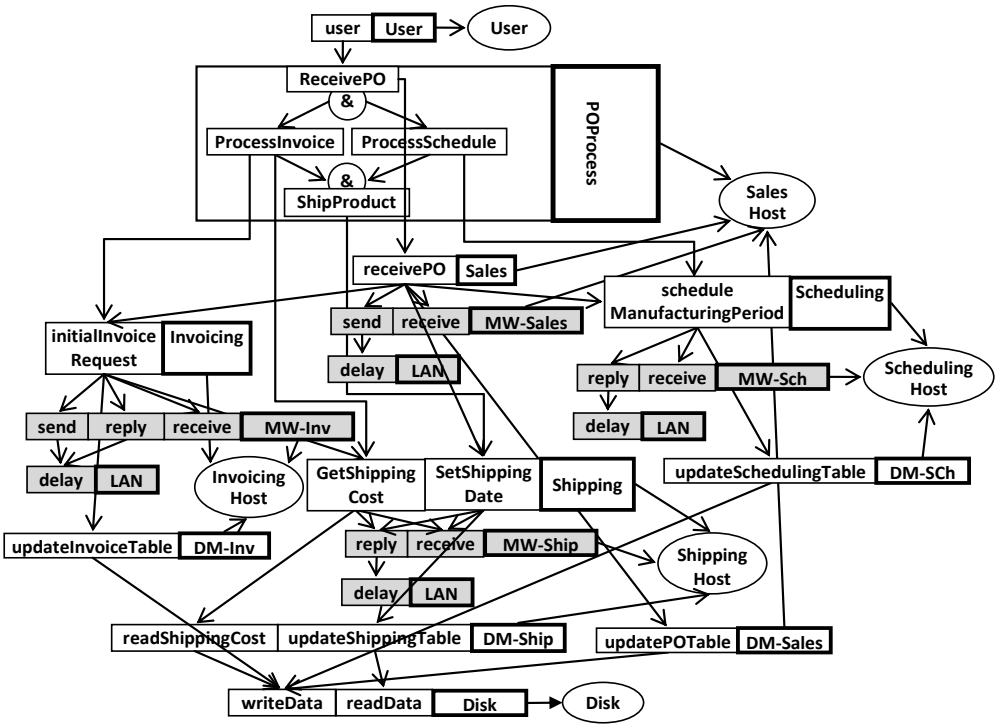


Figure 15. The LQN model of the Purchase Order system.

4.4. Comparison between Composition Alternatives

As already mentioned, PUMA4SOA provides the flexibility of transforming a platform independent model to a platform specific model based on aspect composition at three levels: UML, CSM and LQN. This allows the modelers to select between different 43odelling languages and to perform the AOM aspect composition based on their needs.

The main advantage of performing aspect composition in UML stems from the language features. UML is an OMG standard 43odelling language. It is popular, supported by several model-driven tools and represents the software model which is well understood by software developers.

We also noticed that we were able to define easily the point cut rules in UML, whereas it is more complicated in CSM and LQN. The reason is that some of the SOA-related information, which is present in the UML model, gets lost in the transformation to CSM and to LQN, which are more abstract models. Another advantage of performing aspect composition in UML is the ability to modify models with changing requirements, which cannot be done as easily in CSM or LQN directly; any model change must start from UML, in order to keep the consistency between software and performance model throughout the transformation chain.

However, the UML aspect models have usually multiple views (structural and behavioural) represented by different diagrams, so the composition should take place in all these views. This complicates not only the composition, but the effort of maintaining all the views of the composed system consistent. Besides, the composed models are more complex and may become harder to be understood by the developers.

On the other hand, CSM and LQN both have a lightweight metamodel compared with UML and a single view for each model. This advantage makes the implementation of the aspect composition simpler in CSM and LQN. It is significantly easier to insure the composition consistency and to reduce the execution time required for the model transformation in PUMA4SOA. Table 3 summarizes the points of comparison between the three levels of platform aspect composition.

Table 3. Comparison between aspect composition at UML, CSM and LQN levels.

Comparison Criteria	UML level	CSM level	LQN level
Language features (standard, popularity)	yes	no	no
Modeling language expressive power	high	low	low
Facility to define point cuts	easy	difficult	difficult
Metamodel complexity	high	low	low
Facility to insure composition consistency	difficult	easy	easy
Implementation complexity of aspect composition	difficult	easy	easy
Execution time of model transformations	long	short	short

5. Multiple Aspect Composition in PUMA4SOA

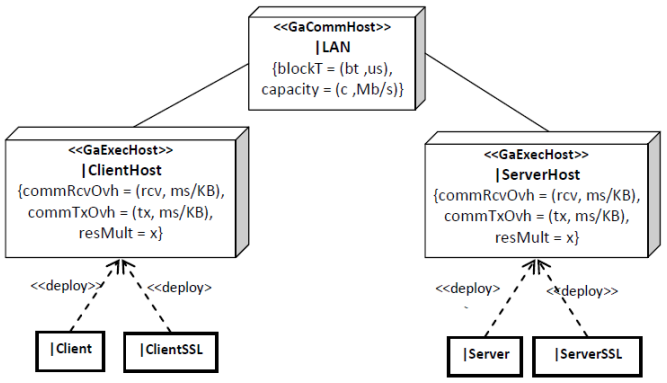
As described by the PC feature models in Section 2, a service platform is characterized by many features. In our work, each platform feature corresponds to an aspect model. PUMA4SOA provides the ability to perform multiple service platform aspect compositions. From the hierarchy of service platform aspects defined in the PC feature model, the developers are able to select as many aspects as needed for the application according to the requirements. The generic UML models of the selected aspects are extracted from a predefined library. This library contains structural and behavioural representations for every service platform aspect in a generic form, as described in section 4.1.

In the multiple aspect composition process, a chain of individual aspect compositions is performed, where each composition may depend on others within the chain. The cause of the dependency comes from the fact that the selected aspects are woven into the same primary model, which means that they share the same context model elements, i.e. same concrete roles and parameters. Moreover, in some cases *nested compositions* are required, where the result of weaving a preceding aspect may act as a primary model for a succeeding one. Due to the dependencies between the aspect compositions, ordering the chain of individual aspect composition is critical.

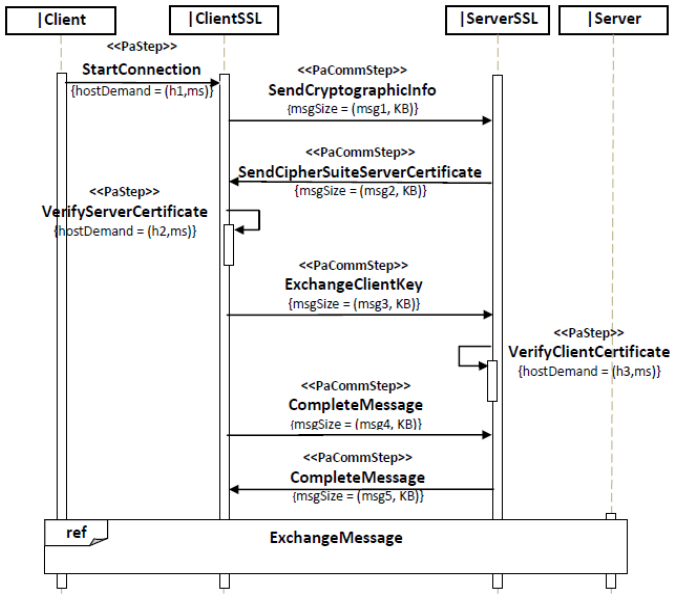
In the PO system example, the goal of aspect composition is to generate the platform specific model by weaving the service invocation aspect into the PIM. Assume that a new aspect is selected from the PC feature model, such as message security, which aims to secure all messages which are sent through public networks. To secure the message exchange that corresponds to service invocations in the PO system, the Secure Socket Layer (SSL) protocol can be used (Freier et al, 2011). SSL is normally used for web-based secure transactions; it handles mutual or one one-way authentication and preserves the integrity of the data exchange between the client and the server. Fig. 16a illustrates the generic deployment diagram and Fig. 16b the sequence diagram of a simplified SSL protocol aspect model (Freier et al, 2011).

The behaviour of the SSL protocol has two phases: 1) the handshake phase, where both the client and the server authenticate each other, and 2) the transfer phase, where the message exchange takes place, with the encryption of the data before the transfer and the decryption of the data on the other side. This is a case of nested composition, where the invocation aspect is woven first and the security aspect second.

Aspect composition is performed by following the same AOM procedure described in the previous sections. For simplicity, we will perform aspect composition at the UML level. In defining the point cut rule, we are looking for all UML elements of type *Message* in the service behaviour model stereotyped with «*PaCommStep*», whose sender and receiver are allocated on different nodes. The second step is



a) Generic deployment diagram for SSL protocol



b) Generic behaviour diagram for SSL protocol

Fig. 16. Platform aspect model for "SSL protocol".

identifying the join points by applying the point cut condition to the primary model. The result will identify all the join points; one of them is *RequestSOAPMessage* in the *Ship Product* sequence diagram shown in Fig 8. The context-specific aspect models are then generated by instantiating the generic aspect model of the SSL protocol by binding the generic roles and formal parameters to concrete values. Table 4 illustrates the binding of generic roles in this case.

Table 4. Binding Generic to Concrete roles.

Generic Aspect	Context Specific Aspect
|Client	SOAPSales
|ClinetSSL	SOAPSalesSSL (new)
|Server	SOAPShip
|ServerSSL	SOAPShipSSL (new)

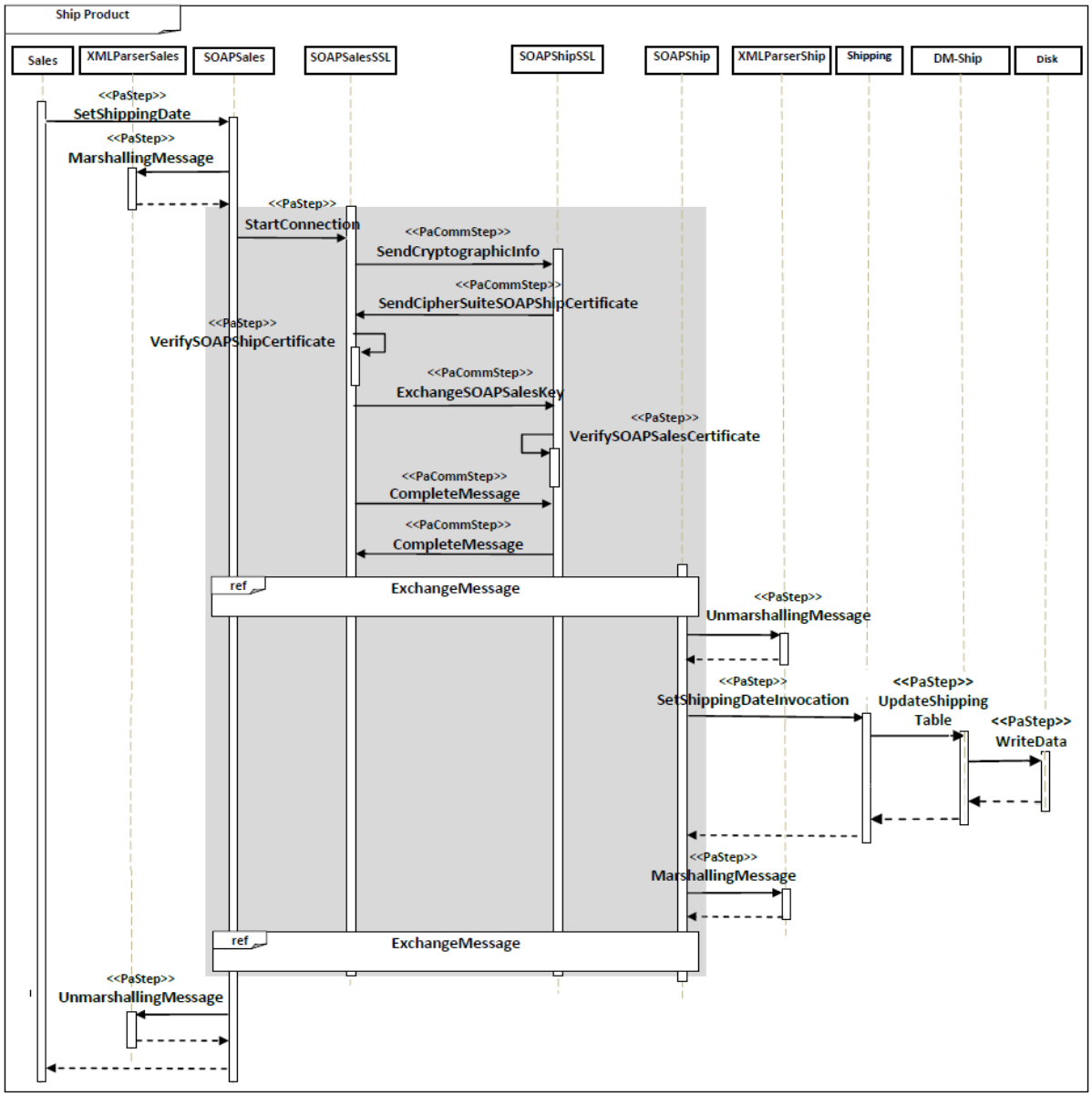


Fig. 17. Multiple aspect composition for *Ship Product* at UML level.

The final step in AOM is to weave the context specific aspect model within the primary model at the join point location. Fig. 17 shows a composed model, where the *RequestSOAPMessage* message (join point) has been replaced with the context specific aspect model (the gray area).

6. Performance Analysis

In this section, we give a brief example of performance analysis for the Purchase Order system. The main purpose of the 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 software modifications to mitigate the bottleneck and to improve the response time and the throughput of the overall system.

Fig. 15 represents the LQN model generated for the PO system (the aspect composition of SSL protocol is not included). An existing LQN solver is used to solve the model and to produce task service times (including waiting for nested services), queueing delays, processor utilizations, response times and throughputs. The results are used to identify the performance hotspots in the system.

The results of the performance analysis, shown in Fig. 18 represent the response time and throughput of the system in function of the number of users ranging from 1 to 100. We analyzed several configurations:

- Initial: the base case where the multiplicity of all tasks and hosts is 1. The Sales task is the software bottleneck. To mitigate the bottleneck, we increase the multiplicity of Sales task.
- A: The bottleneck is resolved by increasing the multithreading of the Sales task to 15, so that multiple requests can be served concurrently. The response time is reduced by 55% (with respect to the initial), and moves the bottleneck to the Invoicing task.
- B: The bottleneck is resolved by increasing the multithreading of the Invoicing to 5. The response time is reduced by 49% (with respect to A), and moves the bottleneck to Disk host, and disk server.
- C: the bottleneck is resolved by increasing the multiplicity of the disk host to 4, and increase the multithreading of the disk server to 3.

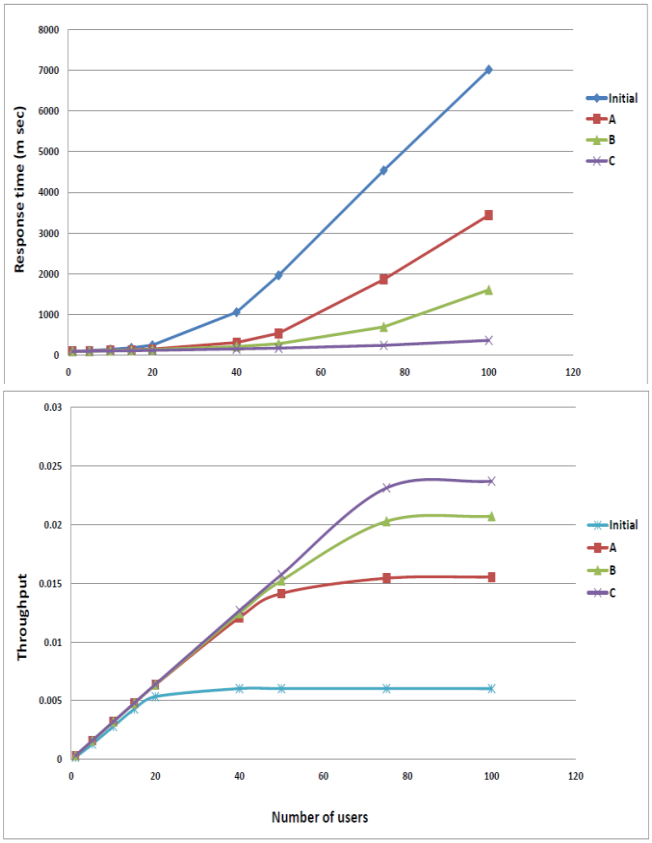


Fig. 18. The LQN results of Purchase Order system: Response time and Throughput.

The performance analysis results show that the response time has been reduced by 90%, and throughput improved by 60% compared to the base configuration (initial).

Another example of performance analysis for a SOA system is given in (Alhaj, 2011), where the trade-off between service granularity and system performance is investigated.

7. Related Work

This section presents a brief overview of related works. In the past decade, OMG has standardized two UML profiles that extend UML for the real-time domain: the "UML Profile for Schedulability, Performance and Time (SPT)" defined for UML 1.X, and the "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)" (OMG, 2009) defined for UML 2.X. 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 software models, as surveyed in (Cortellessa et al, 2011).

A popular realization of SOA systems is based on Web Services; therefore there is interest in modeling and analyzing the performance of such systems. As Web Services use the SOAP protocol, which is XML-based and introduces performance overheads, the performance impact of different SOAP implementations was investigated in (Gomez-Martýnez and Merseguer, 2006). Other research on building performance models for web services takes a two-layered user/provider approach in (Marzolla and Mirandola, 2007): the user is a set of workflows and the provider a set of services deployed on a physical system; the queueing network formalism is used to derive performance bounds. In (D'Ambrogio and Bocciarelli, 2007), performance information about service capabilities and invocation mechanisms is given by the means of P-WSDL (Performance-enabled WSDL).

An approach to model platform operations, called *performance completions* is presented in (Happe et al., 2010). Such completions abstract from platform-specific details and can be instantiated for different target environments using measurements of a test-driver designed especially for a parametric performance completions. Performance completions close the gap between available high-level models and required low-level ones.

Reusable Aspect Models (RAM) is a scalable approach to multi-view aspect-oriented modeling presented in (Kienzle et.al., 2010). RAM allows the modeler to define stand-alone reusable aspect models that support the modeling of structure (using UML class diagrams) and behavior (using UML state and sequence diagrams). RAM supports aspect dependency chains, which allows an aspect providing complex functionality to reuse the functionality provided by other aspects. The paper (Xu et al., 2007) presents an aspect-based modeling approach for web service composition using UML. The approach was motivated by weaknesses of the current composite specification, such as BPEL. In (Abu-Eid, 2007) it is presented an aspect oriented approach which aims to make the process of applying features to web services more flexible and less resource consuming. The approach introduced an aspect oriented extension module which

modularizes the logic of applied features. In (Feng et al., 2009), a requirement-driven aspect oriented approach for web service composition (WSC) is proposed. In this approach, the level of abstraction of evolution modeling for web services composition has been elevated from business process level to strategic goal level. However, none of the AOM approaches targets platform modeling like in our case.

8. Conclusions

The paper focuses on the ability of PUMA4SOA to transform the Platform Independent Model (PIM) of SOA systems into Platform-Specific Model (PSM) by using AOM. PUMA4SOA is a model transformation framework which aims to generate performance models from different design models of SOA systems. It provides the flexibility to perform aspect composition at three levels: UML, CSM and LQN. Although the aspect composition follows similar steps, the degree of composition complexity differs between the three levels, especially between UML and the other two. The complexity depends on the size of the respective language metamodel. A comparison has been done to present the pros and cons of performing aspect composition at the three levels. The main points of comparison are related to language features complexity of the metamodels, implementation complexity of aspect composition, etc.

The paper also discussed the ability of PUMA4SOA to perform multiple aspect compositions by selecting multiple aspects from the PC feature model and then applying a chain of aspect compositions. The dependency between aspect compositions was a major concern, since the order of aspect composition is importance. The dependencies between service platform aspects can be defined by using the relationships defined in the PC feature model, and by using a constraint language, such as the OCL language, to define which aspects can be used together and in which order. The relationship and dependencies between the service platform aspects will be part of our future work,

We have implemented aspect composition in CSM (Alhaj, 2008) but not in UML or LQN. We are integrating CSM composition in the PUMA4SOA. We are planning to define other aspects for SOA platform services (such as discovery, publishing, subscription, and message security) and to study their influence on the performance of SOA systems.

Acknowledgements

This research was partially supported by the Natural Sciences and Engineering Research Council (NSERC) and industrial and government partners, through the hSITE Strategic Research Network.

References

Abu-Eid, V. (2007), “An Aspect Oriented Approach for Applying Features to Web Services”, IEEE International Conference on Web Services (ICWS07).

Alhaj, M. (2008), "Aspect Composition in Core Scenario Models", Master Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa.

Alhaj, M., Petriu, D. C. (2010). “Approach for generating performance models from UML models of SOA systems”, Proceedings of CASCON 2010, Toronto, ON, Canada, pp. 268-282.

Alhaj, M., Petriu, D. C. (2012). “Aspect-oriented Modeling of Platforms in Software and Performance Models”, Proc. of the International Conference on Electrical and Computer Engineering ICECS’ 2012, Ottawa, Canada.

Alhaj, M. (2011). “Automatic generation of performance models for SOA systems”, Proceedings of the 16th international workshop on Component-oriented programming WCOP '11, pp. 33-40.

Batory, D. (2005) "Feature models, grammars, and propositional formulas", SPLC'05 Proceedings of the 9th International Conference on Software Product Lines, pp 7-20.

Cortellessa, V., Di Marco, A., P Inverardi, P. (2011). Model-based software performance analysis, Springer.

D’Ambrogio, A., Bocciarelli, P. (2007) “A Model-driven Approach to Describe and Predict the Performance of Composite Services”, Proceedings of WOSP’07, Buenos Aires, Argentina.

Earl, T. (2005), Service-Oriented Architecture: Concepts, Technology, and Design, Pearson Education.

Feng, Z., He, K., Ma, Y., Peng, R., Gong, P. (2009), “A Requirements-Driven and Aspect-Oriented Approach for Evolution of Web Services Composition”, Proceedings of WMWA '09 the Second Pacific-Asia Conference on Web Mining and Web-based Application.

France, R., Ray, I., Georg, G., Ghosh, S. (2004), “An Aspect-Oriented Approach to Early Design Modeling,” IEE Proceedings - Software, Special Issue on Early Aspects: 151(4):173-185.

Gomez-Martýnez, E., Merseguer, J. (2006) “Impact of SOAP Implementations in the Performance of a Web Service-Based Application”, ISPA 2006 Ws, LNCS 4331, pp. 884–896.

Happe, J., Becker, S., Rathfelder, C., Friedrich, H., Reussner, R. H. (2010), "Parametric Performance Completions for Model-Driven Performance Prediction”, Performance Evaluation, Vol. 67 (8): 694-716.

Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S., (1990) “Feature-oriented domain analysis (FODA) feasibility study”, Technical Report CMU/SEI-90TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Kienzle, J., Al Abed, W, Fleurey, F., Jézéquel, J.M., Klein, J., (2010) “Aspect-Oriented Design with Reusable Aspect

Models", Transactions on Aspect-Oriented Software Development, vol. 7, pp. 279 – 327.

Marzolla, M., Mirandola, R. (2007) "Performance Prediction of Web Service Workflows," Proceedings. of QoSA 2007, LNCS 4880, pp. 127–144.

OMG (2009b), UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems), Version 1.0, formal/2009-11-02.

OMG (2012), Service oriented architecture Modeling Language (SoaML), formal/2012-03-01.

Petriu, D. B., Woodside, C. M. (2007), "An Intermediate metamodel with scenarios and resources for generating performance models from UML designs", Software and Systems Modeling, 6 (2): 163-184.

Selic, B. (2008) "Accounting for platform effects in the design of real-time software using model-based methods". IBM Systems Journal 47(2): 309-320.

Smith, C.U. (1990), Performance Engineering of Software Systems, Addison Wesley.

Tawhid, R., Petriu, D.C., (2011) "Automatic Derivation of a Product Performance Model from a Software Product Line Model", Proc. of the 15th International Conference on Software Product Line (SPLC'11), Munich, Germany.

Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S. (1995) ''The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software,'' IEEE Transactions on Computers, 44 (1): 20-34.

Woodside, C.M., Petriu, D.B., Siddiqui, K.H., (2002) "Performance-related Completions for Software Specifications". Proc. of the 22rd Int Conference on Software Engineering, ICSE 2002, pp. 22-32, Orlando, Florida, USA.

Woodside, C. M., Petriu, D. C., Petriu, D. B., Shen, H., Israr, T., Merseguer, J. (2005), " Performance by Unified Model Analysis (PUMA)" Proc. 5th Int. Workshop on Software and Performance WOSP'05, pp. 1-12.

Woodside, C. M., Petriu, D. C., Petriu, D. B., Xu, J., Israr, T., Georg, G., France, R., Bieman, J., Houmb, S. H., Jürjens, J., (2009), "Performance Analysis of Security Aspects by Weaving Scenarios from UML Models", Journal of Systems and Software, 82 (1): 56–74.

Xu, Y., Tang, S., Tang, Z. (2007), "Towards Aspect Oriented Web Service Composition with UML", Proc. of 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS07).