1-1-2007

Performance Analysis of Security Aspects in UML Models

Dorin Bogdan Petriu
Carleton University

Dorina C. Petriu
Carleton University

C Murray Woodside
Carleton University

Jing Xiu
Carleton University

Toqeer A. Israr
Eastern Illinois University, taisrar@eiu.edu

See next page for additional authors

Performance Analysis of Security Aspects in UML Models

D.C. Petriu, C.M. Woodside, D.B. Petriu, J. Xu, T. Israr
Carleton University
Systems & Computer Eng. Dept.
Ottawa, ON, Canada, K1S 5B6
{petriu | cmw | dorin | xujing} @sce.carleton.ca

Geri Georg, Robert France, James M. Bieman
Colorado State University
Department of Computer Science
Fort Collins, CO 80523, USA
{georg | france | bieman} @cs.colostate.edu

Siv Hilde Houmb
Norwegian Univ. of Science & Tech.
sivhoumb@idi.ntnu.no

Jan Jürjens
TU Munich, Dept. of Informatics
juerjens@in.tum.de

ABSTRACT

The focus of the paper is on the analysis of performance effects of different security solutions modeled as aspects in UML. Aspect oriented modeling (AOM) allows software designers to isolate and separately address solutions for crosscutting concerns, which are defined as distinct UML aspect models, then are composed with the primary UML model of the system under development. For performance analysis we use techniques developed previously in the PUMA project, which take as input UML models annotated with the standard UML Profile for Schedulability, Performance and Time (SPT), and transform them first into Core Scenario Model (CSM) and then into different performance models. The contribution of this paper is in performing the composition of the aspects with the primary model at the CSM level. The input is represented by the primary model and a number of aspect models in UML+SPT, which are processed as follows: a) converted separately to CSM; b) composed into a single CSM model; c) transformed into a Layered Queueing Networks (LQN) model and d) analyzed. The proposed approach is illustrated with a case study based on two standards, TPC-W and SSL.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: *modeling techniques, performance attributes*. D.2.4 Software/Program Verification: *model checking*

General Terms

Performance, Security, Design.

Keywords

Software Performance Engineering, Aspect-Oriented Modeling, Security, Model transformations, UML.

1. INTRODUCTION

Complex distributed dependable systems, such as web-based applications that contain sensitive data and have many users, 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. Tradeoffs between competing requirements often come late in the development cycle, and changes can be expensive if they are fundamental, such as architectural changes. A better approach, enabled by OMG's Model Driven Architecture (MDA), is to start analyzing different non-functional requirements from the early stages of the development process, based on design models.

An approach for the analysis of various non-functional properties of a UML design model is currently emerging; similar steps are followed, regardless of the non-functional property considered: a) annotate the UML model with extra-information specific to the non-functional property (by using special-purpose UML Profiles); b) transform the annotated model in a specific analysis model (such as first predicate logic, queuing networks, Petri nets, etc.); c) analyze the model with existing tools; d) give feedback to designers from the analysis results.

The authors of the paper are involved in a larger research effort to integrate methodologies and tools that support the analysis of non-functional requirements, such as security and performance, from the early system development phases, based on UML models.

A UML-based approach for verifying whether a design meets security properties is presented in [9]. A UML model annotated with a specialized profile named UMLsec is converted into a first-order logic model, which is input to a theorem prover, along with a representation of an adversary. The prover determines if the system model is secure.

The Risk-Driven Development (RDD) profile supports the Aspect-Oriented Risk Driven Development (AORDD) framework [7],[8]. The goal of the framework is to assist developers in designing cost-effective systems with the desired level of security. It is a model-based approach, driven by asset risk management.

Aspect-oriented modeling (AOM) is used to specify and integrate security risks and solution designs into system models [4],[17]. In general, AOM allows software designers to isolate and separately address solutions for any crosscutting concerns, which are defined as separate UML aspect models, then are composed with (or "weaved in") the primary UML model of the system. Such concerns range from high-level, user-visible requirements (such as reliability, security, new functional features) to low-level implementation issues (such as synchronization and buffering). There are many notable research efforts in the area of aspect composition, both for structure and behaviour. Different UML means for representing behaviour have been used for aspect composition; for instance, interaction diagrams are used in [4],[20], statecharts in [6], [10], and activity diagrams in [2], [18].

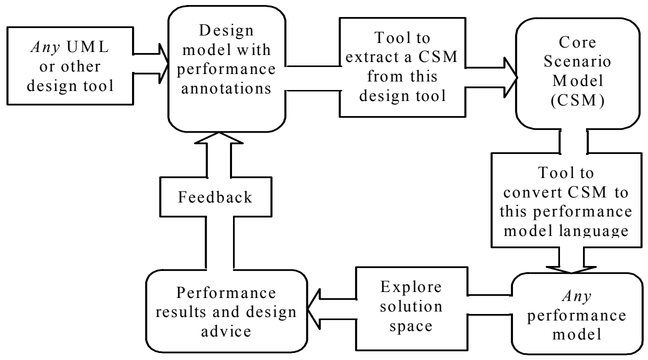
The first attempt to analyze the overall performance effects of weaving an aspect model (which represents a security mechanism) is found in [18]. The aspect model is composed with the primary model at the UML level; the composed model is then transformed into a LQN model according to the PUMA approach [22]. Bringing performance concerns to aspect-oriented modeling is adding new issues that have to be taken care of during aspect composition, such as paying special attention to resource usage and taking care of the composition of SPT performance annotations. For instance, it is necessary to add concurrency and deployment to the list of model views affected by aspect weaving, as these are important for performance analysis (whereas in non-performance related aspect composition, these views are usually ignored).

The present paper approaches the problem from [18] in a different way. It takes as input a primary model and one or more security aspect models in UML+SPT, and processes them as follows: i) separately transforms the primary and the aspect models to CSM; ii) composes them at the CSM level; iii) transforms the CSM composed model into LQN, and iv) analyzes the LQN model. The main advantages of this approach are: a) the aspect composition algorithm is simpler, as the CSM metamodel is much simpler than the UML metamodel and processing performance annotations is an intrinsic part of CSM; b) a "single" composition algorithm must be devised instead of several algorithms for all the possible kinds of UML behavioral diagrams; and c) it is possible to mix different behavioural descriptions (e.g., compose aspects defined originally as activity diagrams with a primary model using sequence diagrams, and vice-versa). An obvious disadvantage of the proposed approach is that aspect composition at the CSM level is not able to solve non-performance related problems that are dealt with at the UML level (such as consistency and correctness checks at model level, model transformations, etc.)

The paper is organized as follows: section 2 briefly describes the PUMA approach for transforming UML+SPT models into performance models; section 3 discusses aspect-oriented modeling of security mechanisms and describes the case study system based on TPC-W and SSL; section 4 introduces our new approach of composing aspects at the CSM level; section 5 presents the LQN model and some performance results and section 6 gives the conclusions and future work.

2. PERFORMANCE ANALYSIS OF UML MODELS

To support early performance analysis of software specifications, the UML has been extended by the standard SPT Profile ([12], currently being revised [13]), which defines annotations for performance parameters, resource usage, and workloads. SPT focuses on specified *scenarios* based on important use cases and is designed to work with a tool chain like the PUMA tool architecture shown in Figure 1.



Any UML or other design tool
Design model with performance annotations
Tool to extract a CSM from this design tool
Core Scenario Model (CSM)
Tool to convert CSM to this performance model language
Feedback
Performance results and design advice
Explore solution space
Any performance model

Figure 1. Tool interactions and information flows in the PUMA architecture [22].

2.1 The PUMA methodology

A wide variety of approaches can be applied to performance modeling of software [1]. PUMA (Performance by Unified Model Analysis) [22] unifies the creation of models by an intermediate performance metamodel called Core Scenario Model (CSM) [14]. As shown in Figure 1, CSM provides a single target for transforming from a design model created by any UML tool, and from different UML views of system behaviour (e.g. interaction diagrams, activity diagrams), and a single source for creating different kinds of performance models (e.g. queueing models, Petri net models).

CSM defines operations (called *Steps*) with precedence relationships and resource requirements and demands. Precedence patterns include sequence, loop, fork/join, branch/merge, with no requirement that a forked or branched path should rejoin into a single flow. Resources include processors, other devices, software components, processes and logical resources of all kinds, and demands describe how many requests are made by a step, or how much CPU processing is demanded. Each scenario has a *workload* which defines the arrivals of requests to execute the scenario. Examples are given in Section 4 below.

The process of model derivation from a CSM is described in [22] for queueing networks, layered queueing networks (which are used in this paper), and stochastic Petri net models. In this paper we show that system transformations for the insertion of security aspects into the primary model of a system can also be executed at the CSM level, to explore alternative approaches.

3. UML ASPECT-ORIENTED MODELING OF SECURITY MECHANISMS

Aspect-Oriented Modeling (AOM) techniques allow software designers to separately conceptualize, describe and communicate solutions for crosscutting concerns (such as security, reliability, new functional features, etc.) An aspect-oriented architecture model produced by AOM consists of a base architecture model called the *primary model*, which reflects core design decisions, and a set of *aspect models*, each reflecting a concern that crosscuts the primary model [4]. In order to build the complete solution for a system, different aspect models are composed with the primary model. Figures 2, 3 and 4 show the UML primary model for the TPC-W example used in this paper, while Figure 5 shows the SSL aspect model.

3.1 Primary model

The case study for this paper is based on TPC-W, a transactional web benchmark of the Transaction Processing Performance Council [21], which models the workload of an on-line bookstore. The primary model represents the basic functionality without any security mechanisms. SSL secure communication is later added to the primary model through aspect composition.

The components of TPC-W are logically divided into three tiers: a) set of emulated web browsers (EB), b) web tier including Web Servers, Image Servers and Web Caches and c) persistent storage. TPC-W simulates customers browsing and buying products from a website.

The TPC-W specification describes in detail 14 different web pages that correspond to typical operations performed by a customer of an e-commerce website. The first page to be visited by a user is the "Home" page; it includes the company logo, promotional items and navigation options to the top best selling books, a list of new books, search pages, shopping cart, and order status pages. At every page, the user is offered a selection of pages that can be visited next; the user will make a random choice. The user may browse pages containing product information, perform searches with different keys and put items in the cart, or may decide to order books by entering secure order pages, protected by SSL. In order to make an order, a new customer has to fill out a customer registration page; for returning customers, the personal information is retrieved from the database and filled in automatically. Before ordering, the user may update the shopping cart content. When deciding to buy, the user enters the credit card information and submits the order. The system will obtain credit card authorization from a Payment Gateway Emulator, PGE, and present the user with an order confirmation page. At a later date the user can view the status of the last order. Two additional web pages are provided for the system administrator.

The navigation options provided on every page, lead to an access distribution referred to as the "Web Interaction Mix". Thus 80% of the web page accesses are to the Home, New Products, Best Sellers and Search pages while the remaining 20% of the accesses are to the Shopping Cart, Order, Buy and Admin web pages. Of the 20% ordering web pages, 5% of the accesses are to secure web pages requiring SSL encryption.

As the purpose of this paper is to illustrate how security aspects can be composed with the primary model of a system at CSM level, we have not considered the entire functionality of TPC-W for the UML primary model. We have selected instead only two scenarios that are accessing secure pages: one is a light-weight scenario that returns the customer registration page (shown in Figure3) and the other a heavy-weight scenario that allows the user to buy a product (shown in Figure4).

The primary UML model contains a structural and a behavioural view necessary for performance evaluation [15].

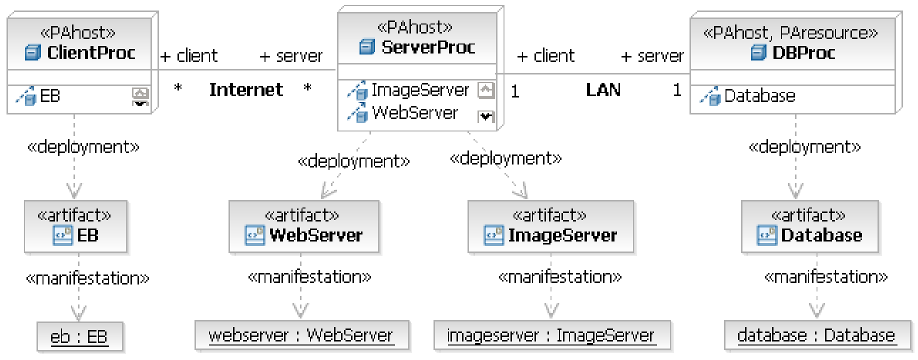


Figure 2. UML deployment diagram for TPC-W

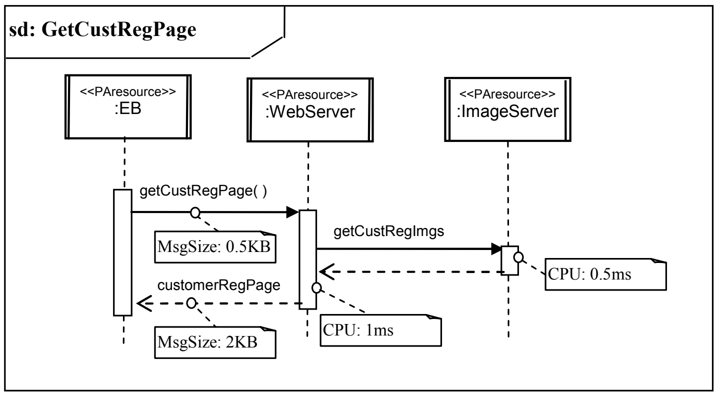


Figure 3. TPC-W UML primary model: scenario GetCustRegPage

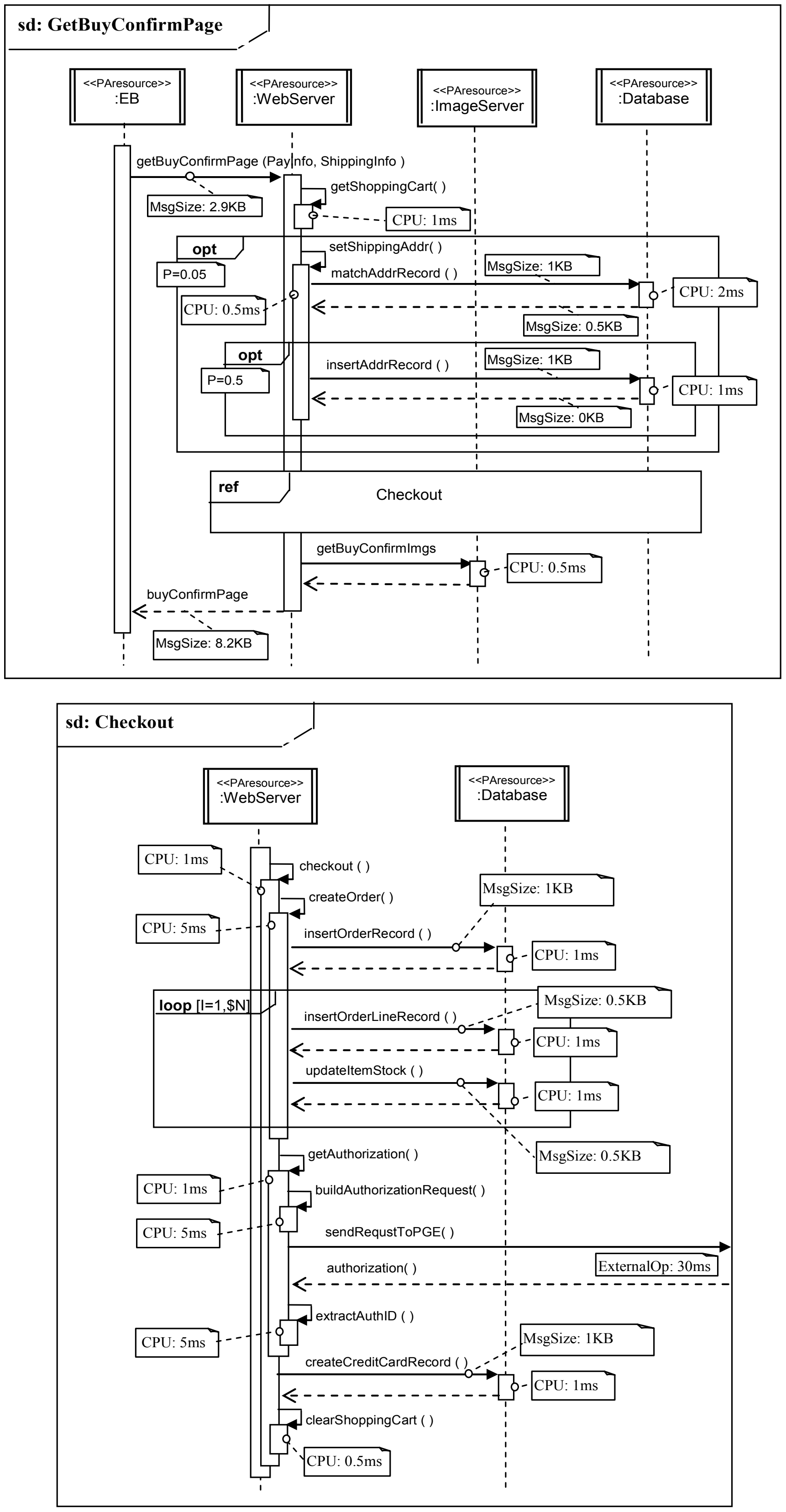


Figure 4. TPC-W UML primary model: scenario GetBuyConfirmPage

- Deployment of high-level software components to hardware devices (Figure 2)
- One or more key performance scenarios annotated with performance information according SPT [12], modeled in this case as interaction diagrams (Figures 3 and 4).

The deployment diagram shows the software components, their corresponding artifacts and the deployment of artifacts on processing nodes. The `DBProc` node is stereotyped with both PAhost and PAresource since it has a multiplicity of 5. The PAhost stereotype identifies the node as a host, while the PAresource stereotype is needed to specify the multiplicity. If the multiplicity is not specified, it's always assumed to be 1.

The interaction `GetCustRegPage`, shown in Figure3, returns the registration web page to EB. This scenario is interesting because it starts with a non-secure message between `EB` and `WebServer`, but ends with a secure reply. The User will use the returned page to register as a known or new customer in another interaction (not shown here). The following operations are performed:

- `EB` issues a request for the customer registration page;
- `WebServer` gets the necessary images (company logo, button images, etc) from `ImageServer`;
- `WebServer` constructs the html customer registration page and returns it to `EB`.

The scenario `GetBuyConfirmPage` is described in two interaction diagrams shown in Figure 4. The top interaction transfers the shopping cart content into a newly created order for the registered customer and executes a full payment authorization, then returns a web page containing the details of the newly created order to the `EB`. The following operations are performed:

- `EB` issues a request to `WebServer` for "buy confirm page";
- `WebServer` gets the corresponding shopping cart object;
- With 5% probability, a shipping address is passed from `EB`.
  - `WebServer` tries to match the shipping address in the corresponding table in the database
  - If no address record is found, insert a new address record
- Invokes the `Checkout` sub-scenario (as a **`ref`** fragment)
- `WebServer` gets necessary images from `ImageServer`
- `WebServer` constructs the html code for the buy confirm page and returns it to `EB`.

The `Checkout` scenario is represented by the second interaction diagram from Figure 4. It creates a new order in database, with all the items in the cart turned into order lines. Then an authorization is obtained from the Payment Gateway Emulator (`PGE`) that is an external system. Finally the credit card is registered in database and the cart is cleared.

The SPT Profile performance annotations used in this primary model are:

- CPU host demand in milliseconds for operations (applicable to execution occurrences and message stereotyped as <<PAstep>>)
- probability PAprob for **`alt`** and **`opt`** interaction operands stereotyped as <<PAstep>>
- repetition count PArep for **`loop`** interaction operands stereotyped as <<PAstep>>
- PArate in processor operations per millisecond for host devices (applicable to nodes stereotyped as <<PAhost>>)
- PAcapacity for device multiplicity (applicable to nodes stereotyped as <<PAresource>>)
- The operation performed by the `PGE` external system is represented as an SPT "external operation" (a tagged value of the stereotype PAstep that indicates the name of the external operation and the number of visits [12]). It will be represented in the performance model as a new task.

Please note that we have not given in Figures 3 and 4 the full SPT syntax for performance annotations, in order to limit the clutter. However, the complete stereotypes and corresponding tagged values were defined with the IBM Rational Software Architect (RSA) tool used to generate automatically the CSM models shown in the next section. An additional performance annotation for message size in kilobytes $MSG_SIZE is also shown in Figures 3 and 4. (It is used later in Table 2 for computing concrete parameter values, such as the number of message fragments).

3.2 Generic Aspect model

A generic aspect model describes the solution proposed by the aspect in a general way, not related to the specific primary model in which will be eventually inserted. According to [4], a generic aspect model can be instantiated multiple times to produce multiple *context-specific aspect models* based on different *binding rules*.

The generic aspect model used in this paper describes the general structure and behaviour of the SSL protocol, without any reference to the system to which SSL will be applied. SSL is the most common authentication protocol used for web-based secure transactions [11]. It handles mutual or one-way authentication and preserves the integrity and confidentiality of data exchange between clients and servers. SSL has two phases: a *handshake* phase and a *data transfer* phase. Each phase represents a different functionality that should be inserted in the primary model at different join points. Therefore, each phase is to be modeled as a separate aspect model. Due to space limitations, we will describe in this paper only the data transfer aspect model.

The handshake phase allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys to be used during the data transfer phase. The encryption mechanisms are different during the two phases: public key encryption is used for the handshake, and symmetric encryption for data transfer. Symmetric encryption is much faster than public key encryption [11].

The SSL data transfer is modeled as a generic aspect in Figure 5, which shows a deployment diagram describing constraints on the structure, as well as an interaction diagram describing the behaviour. In this case, the structural constraint is that the SSL proxies must be located on the same node as the processes they are associated with. All the nodes are generic; they will be bound to concrete nodes in the process of instantiating the context-specific aspects, as described in section 4.2. The interaction diagram in Figure 5.b involves four generic roles: `|sender` (the data source), `|senderSSL` (data source SSL proxy), `|receiver` (data target) and `|receiverSSL` (data target SSL

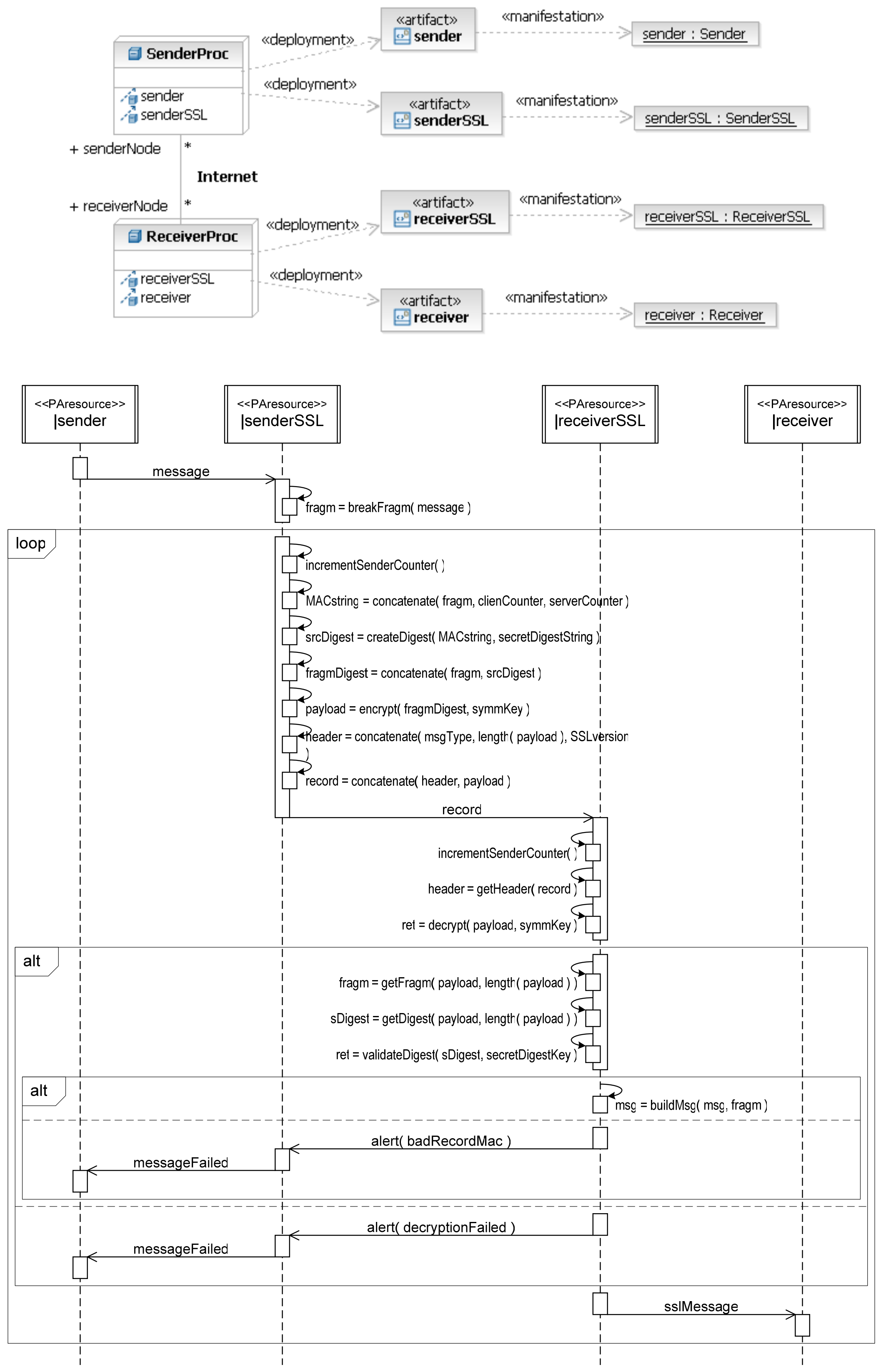

SenderProc
sender
senderSSL
«deployment»
«artifact» sender
«manifestation»
sender : Sender
«deployment»
«artifact» senderSSL
«manifestation»
senderSSL : SenderSSL
+ senderNode
*
Internet
+ receiverNode
*
ReceiverProc
receiverSSL
receiver
«deployment»
«artifact» receiverSSL
«manifestation»
receiverSSL : ReceiverSSL
«deployment»
«artifact» receiver
«manifestation»
receiver : Receiver
<<PAresource>> |sender
<<PAresource>> |senderSSL
<<PAresource>> |receiverSSL
<<PAresource>> |receiver
message
fragm = breakFragm(message)
loop
incrementSenderCounter( )
MACstring = concatenate(fragm, clienCounter, serverCounter)
srcDigest = createDigest(MACstring, secretDigestString)
fragmDigest = concatenate(fragm, srcDigest)
payload = encrypt(fragmDigest, symmKey)
header = concatenate(msgType, length(payload), SSLversion)
record = concatenate(header, payload)
record
incrementSenderCounter( )
header = getHeader(record)
ret = decrypt(payload, symmKey)
alt
fragm = getFragm(payload, length(payload))
sDigest = getDigest(payload, length(payload))
ret = validateDigest(sDigest, secretDigestKey)
alt
msg = buildMsg(msg, fragm)
alert(badRecordMac)
messageFailed
alert(decryptionFailed)
messageFailed
sslMessage

Figure 5. Generic Aspect UML model for SSL Data Transfer: deployment and interaction diagrams

proxy). We use the convention that generic role names start with a '|', similar to [4]. These roles are to be bound to application components when the generic aspect is instantiated to a specific context. A message from `|sender` is first broken into fragments by `|senderSSL`. The number of fragments depends on the length of the data to be transferred. For each fragment, the source counter is incremented; this is the unique counter for the data source. Both the target and source counters are appended to the fragment and a digest is created across this string, using a secret digest string. The digest is appended onto the fragment and then encrypted using the symmetric key exchanged during the handshake phase (resulting in a payload). A header is pre-pended to this information, which contains the type of the message, the length of the fragment and digest, and the SSL version number used by the data source. This entire entity is the record that is sent to the target; `|receiverSSL` increments the source counter, extracts the header, decrypts the payload using the symmetric key, extracts the fragment and digest, and validates the digest using the secret digest string. If either the decryption or the digest validation fails, the receiving target sends an alert to the data source that indicates the failure type. Depending on the overall application protocol (which is independent of the SSL protocol), the data source may attempt to re-send the record, or terminate.

It is important to mention that the performance annotations in generic aspects use variable placeholders instead of concrete values for the tagged values PAdemand, PAprob, etc. These variables will be assigned concrete values only after the instantiation of the generic aspect to specific contexts (as described in section 4.2).

The description of the SSL data transfer aspect model raises a general issue: what level of detail is appropriate for the UML model when trying to integrate the analysis of multiple non-functional properties - in this case, security and performance. The interaction diagram from Figure 5.b gives a detailed functional description that is necessary for the logical verification of the security mechanisms by using a first-order logic model, as in [9]. However, for performance analysis a coarser granularity level would be more appropriate. For instance, many of the small sequential steps could be aggregated into larger steps, which would need fewer performance annotations.

On one hand, it would be preferable to have a different UML view of the system under development for each kind of analysis we intend to perform. However, the problem is that these different views need to be maintained separately as the system evolves, so there is a danger that they could get out of synch. Hence, there is a strong argument for keeping a single UML model as the input for different analysis techniques and tools. The implication is that automatic model transformations will be required to raise the level of abstraction to an appropriate level for different analysis techniques. For instance, in the case of performance analysis, the aggregation of different steps could be done automatically, under the user's guidance. The user needs to be involved if he/she would have to enter performance annotations only for the coarser-granularity steps obtained by aggregation.

3.3 Aspect Composition

Before composing the aspect with the primary model, we need to instantiate the *generic aspect model* for a given application context by binding the roles to application-specific values. As already mentioned, a generic aspect model can be instantiated multiple times to produce multiple *context-specific aspect models* based on different *binding rules*. Because SPT annotations are necessary for performance analysis, the binding rules have two parts: one for binding generic roles to components/nodes from the primary model, and the other for giving concrete values to the performance parameters, as described in section 4.2.

The composition of the aspect models with a primary model can be performed at different levels: UML or CSM. There are many papers in the literature focused on the composition at the UML level. For instance, in [4][17], the generic aspect models are defined by using the concept of *UML templates*. The context-specific models are obtained from generic models by binding the parameter templates to values from the primary model context. The advantage of composing at the UML level is that the resulting model is also in UML; therefore, it may be further visualized, developed, transformed or analyzed with tools that operate directly on UML models. A disadvantage is that the UML metamodel is very complex; this has a direct impact on any composition algorithms.

In this paper we propose for the first time to perform the aspect composition at the CSM level. One advantage is that CSM was defined to be a unique target for transformation from many UML versions and diagrams, and a source to many performance models. For instance, we are able to compose aspects with primary models even if they are originally defined in different UML behaviour diagrams (i.e., any mix of activity, sequence, communication, and interaction overview diagrams can be handled at the CSM level). Another advantage is that the CSM metamodel is much simpler than the UML metamodel, and therefore the composition algorithms are easier to design and implement. An obvious disadvantage is that CSM, whose purpose is to model scenarios, is much more restricted in scope and usage than UML.

4. CSM ASPECT COMPOSITION

4.1 CSM Generic Aspect Model

In the PUMA toolset, CSM models are automatically generated from UML+SPT models created with the IBM Rational Software Architect (RSA) either from sequence or from activity diagrams [22]. For this work, we used CSM models obtained with the CSMGenerator, an Eclipse-based RSA plug-in that traverses UML 2.0 Sequence Diagrams with performance annotations and generates a CSM *Scenario* for every diagram.

UML scenarios with a PAworkload generate top-level CSM Scenarios, while UML scenarios without a workload annotation generate CSM sub-scenarios. Lifelines in the interaction diagram generate CSM *Components* which can have host associations with CSM *ProcessingResources* that are specified as nodes in UML deployment diagram.

Figure 6 shows the CSM top-level scenario for the TPC-W `GetBuyConfirmPage` primary model, as well as the `Checkout` sub-scenario, shown in Figure 4. The top-level scenario shows the CSM *Steps* corresponding to the execution occurrences stereotyped as <<PAstep>> in the interaction diagram as well as explicit *ResourceAcquire* and *ResourceRelease* elements for the resources corresponding to the lifelines.

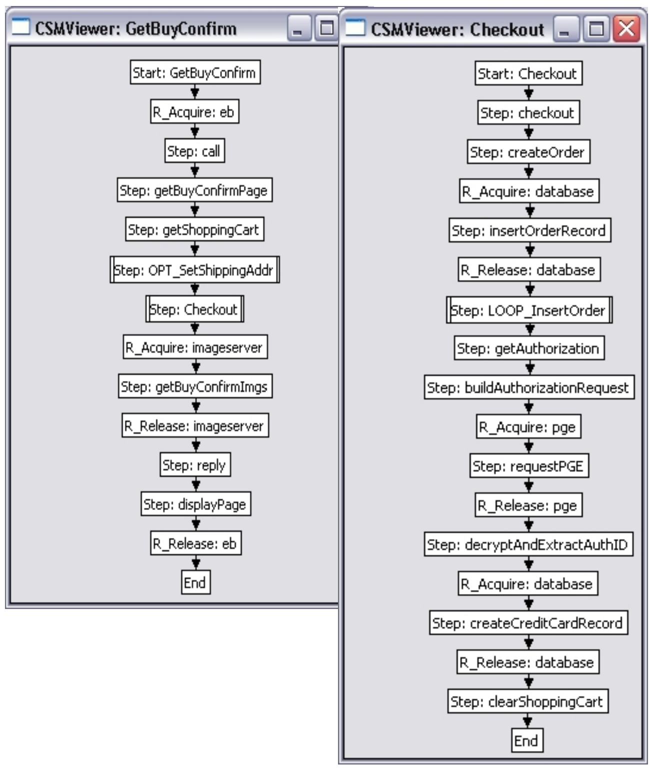


Figure 6. CSM GetBuyConfirmPage primary model

A *ResourceAcquire* is generated whenever a lifeline first receives a synchronous or asynchronous call message or whenever execution first stars. A *ResourceRelease* is generated whenever a lifeline sends an asynchronous call message or whenever execution finishes.

Opt and **loop** combined fragments are shown as CSM *complex steps* with the **loop** interaction operands as *refinements* for those complex steps. The interaction operand details are shown as CSM sub-scenarios. For the `GetBuyConfirm` top-level scenario, the **opt** combined fragment for setting the shipping address is shown as the *OPT_SetShippingAddr* complex step with a corresponding refinement as the *OPT_SetShippingAddress* sub-scenario (not shown). Similarly in the `Checkout` scenario, the **loop** combined fragment that inserts the order details is shown as the *LOOP_InsertOrder* complex step and its corresponding sub-scenario (not shown).

Figure 7 shows the complete CSM scenario for the generic *SSLtransfer* aspect introduced in Figure 5. For the *SSLtransfer* example, the loop step and the corresponding sub-scenario are both called *LOOP_Fragments*. The **alt** combined fragments are shown as matched *Branch* and *Merge* constructs with complex steps for every alternate interaction operand. The details of the interaction operands are shown as separate sub-scenarios for every operand. In Figure 7 the alternatives are: *ALT_DecryptOK*, *ALT_DecryptFail*, *ALT_DigestOK*, and *ALT_DigestFail*.

Par combined fragments are treated similarly to **alt** combined fragments. They generate matched CSM *Fork* and *Join* constructs with complex steps for every parallel interaction operand.

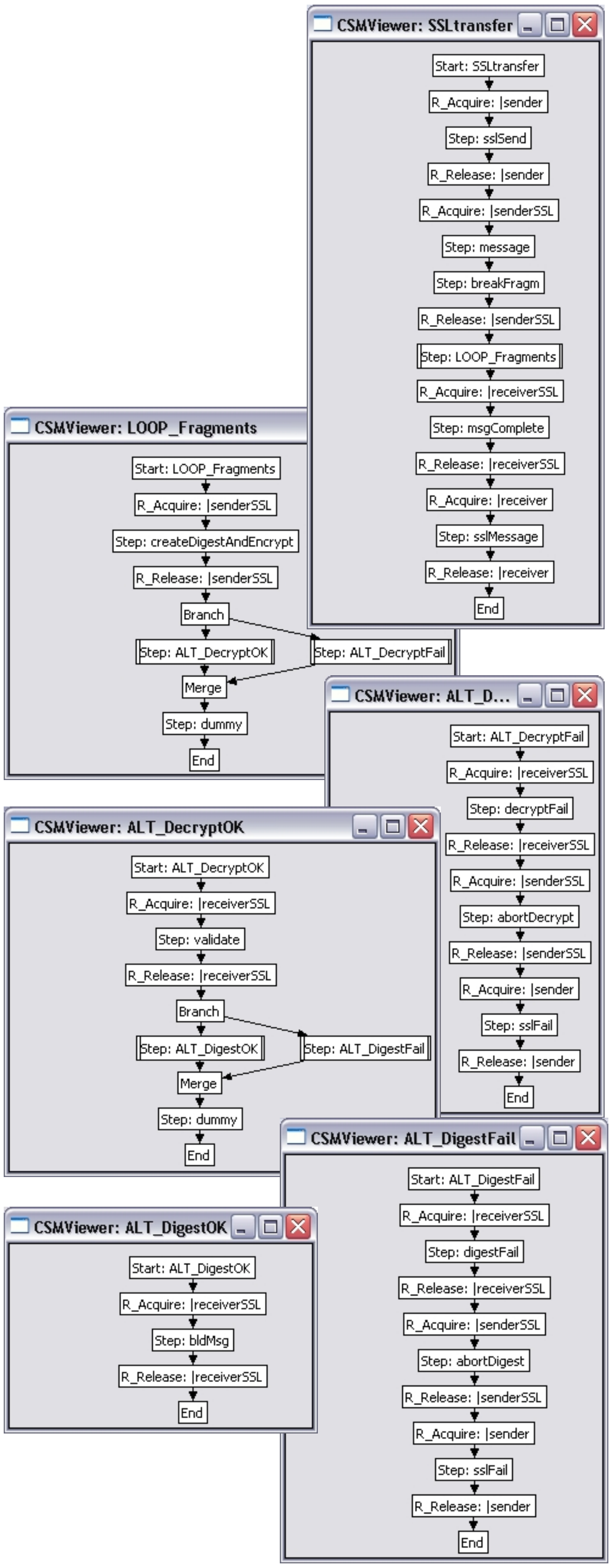


Figure 7. Generic Aspect CSM model: SSLTransfer

The details of the interaction operands are then shown as sub-scenario refinements for the complex steps. The lifelines from the sequence diagram correspond to generic CSM *Components* that retain the role names. The generic components either have no host associations if the roles have no deployment constraints, or have host associations to CSM *ProcessingResources* that correspond to nodes in any constraining UML deployment diagrams.

4.2 CSM Context-Specific Aspect Model

Generic aspect models are transformed into context-specific aspect models by binding the resource roles to actual resources and then assigning context-specific performance values to step processing demands, branching probabilities, optional probabilities, and loop repetition counts.

The first step in transforming the generic aspect model into a context-specific aspect model involves binding the generic resource roles `GR` to context-specific resources `SR`. These context-specific resources can be either existing resources `PR` from the primary model, or new resources required by the aspect.

The resource binding algorithm is as follows:

```
for all $GR_i$:
  if $GR_i$ has a corresponding $PR_j$ then
    $SR_i$ = $PR_j$
  else
    $SR_j$ = instantiate( $GR_i$ )
```

In the TPC-W `GetBuyConfirmPage` example, the generic *SSLtransfer* aspect is bound to two different context-specific aspects; a context-specific *SSLcall* aspect for the `EB` calling the `WebServer`, and a different context-specific *SSLreply* aspect for the `WebServer` replying to the `EB`. The resource bindings for the context-specific aspects are given in Table 1.

Table 1. Context-specific aspect resource bindings

Generic Aspect	*Context-Specific Aspect*	
SSLtransfer	**SSLcall**	**SSLreply**
|sender	eb	webserver
|senderSSL	ebSendSSL (new)	webSendSSL (new)
|receiverSSL	webRcvSSL (new)	ebRcvSSL (new)
|receiver	webserver	eb
|senderProc	ClientProc	ServerProc
|receiverProc	ServerProc	ClientProc

The concrete annotations can be either values or expression (for instance, the number of loop repetitions depends on the message size, which depends on the join point into the primary model).

The values for the performance annotations used in the *SSLcall* and *SSLreply* top-level scenarios are given in Table 2. The step service demands have literal values, while the repetition count for *LOOP_Fragments* is an expression indicating that the repetition count is equal to the message size divided by the fragment size (512 bytes) and rounded up to the nearest integer.

The performance values are the same for both the *SSLcall* and *SSLreply* context-specific aspects because the two aspects are symmetrical in this example. This is not always the case, and it may be possible to have different context-specific aspects based on the same generic aspect, which have different performance values (e.g., different service demands for encryption and decryption due to using different encryption algorithms).

Table 2. Performance values for the top-level scenario in both SSLcall and SSLreply context-specific aspect models

Step	**Service Demand**
sslSend	0.1
message	0.1
msgComplete	0.1
sslMessage	0.1
Loop	**Repetition Count**
LOOP_Fragments	ceiling($MSG_SIZE / 512)

It is worth mentioning that the communication between the `WebServer` and `PGE` taking place in the `Checkout` sub-scenario must be secure, as well. This would require the instantiation of the SSL generic aspects to another context. However, in this paper we have not done this instantiation and composition for the sake of simplicity. We have assumed instead that the latency of the external operation that accesses `PGE` includes the overhead for SSL transfer.

4.3 CSM Composed Model

The CSM composed model is generated by weaving the context-specific aspect models into the primary model. The weaving involves identifying the appropriate join points in the primary model behaviour and inserting the context-specific aspects at those join points. As part of the weaving, an aspect's resource context must also be reconciled with the primary model resource context at the join point. Finally, the woven aspects are inspected for any remaining performance annotations that can be further resolved.

Figure 8 shows the CSM composed model for `GetBuyConfirmPage` with SSL data transfer between `EB` and `WebServer`. For this example, the join point for the *SSLcall* aspect is the *call* step, while the join point for the *SSLreply* aspect is the *reply* step. The weaving is done by replacing the join point steps with complex steps – *call* and *reply* are replaced with *SSLcall* and *SSLreply* – and using the context-specific aspects as refinements for those complex steps.

As part of the resource context reconciliation during weaving, the *SSLcall* aspect loses the *ResourceAcquire:eb* element at the beginning since `EB` is already acquired in the primary model before *SSLcall* is invoked as well as the *ResourceRelease: webserver* at the end, since it is already released in the primary model after the aspect completes. Similarly, the *SSLreply* aspect loses the first *ResourceAcquire: webserver* and the last *Resource Release: eb*.

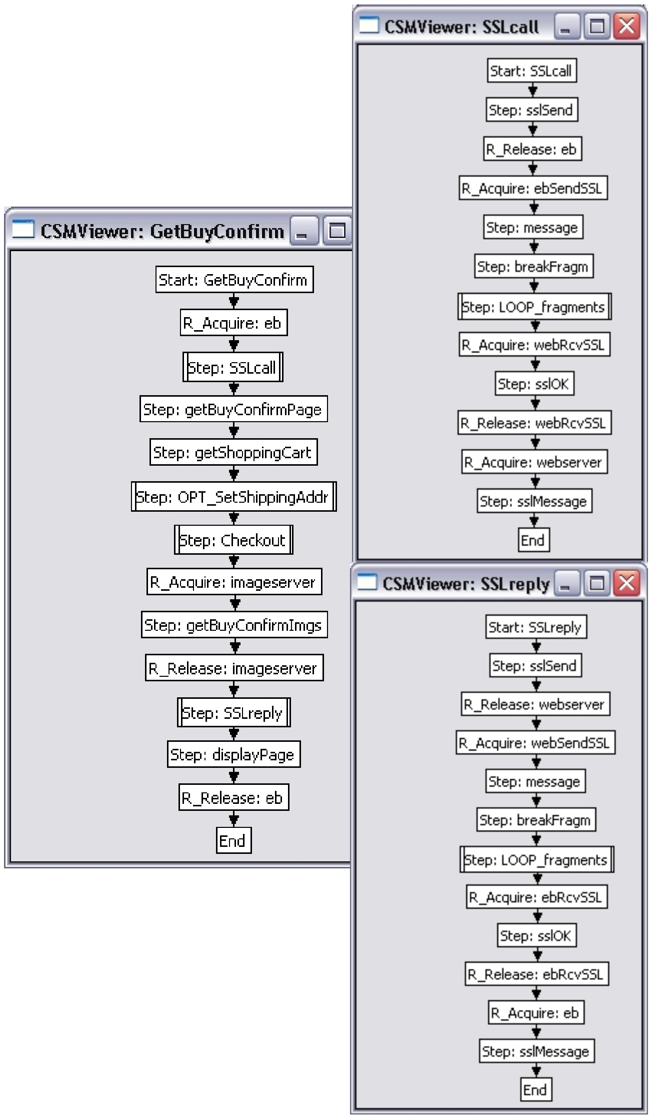


Figure 8. Composed CSM model

The aspect composition is straightforward in this case because the SSL aspect was used to replace simple non-secure messages in the primary model. This allowed us to substitute simple CSM steps with one input and one output with composed steps with one input and one output. In the general case, an aspect may require more than a single input and/or output, which leads to a more complex composition. Such an example is when an aspect model contains alternative or parallel behaviours, where the respective branches need to be "attached" to the primary model in different input or output points.

A more general composition approach involves defining aspect join contexts instead of just join points. Instead of being simple steps, join contexts are either CSM path fragments (i.e. sequences of steps with single beginnings and single ends) or combinations of CSM path fragments. Those path fragments are then used to generate sub-scenarios and are replaced by complex steps using those sub-scenarios as refinements. The aspects can be composed into the model either as sub-scenario replacements or in combination with the existing sub-scenarios. More research is necessary for developing algorithms for more complex compositions cases such as these.

5. LQN PERFORMANCE ANALYSIS

The CSM primary and composed models are automatically transformed into LQN models using the Csm2Lqn generator. Csm2Lqn is an Eclipse-based tool that implements the Scenario to Performance (S2P) algorithm described in [15].

5.1 LQN Primary and Composed Models

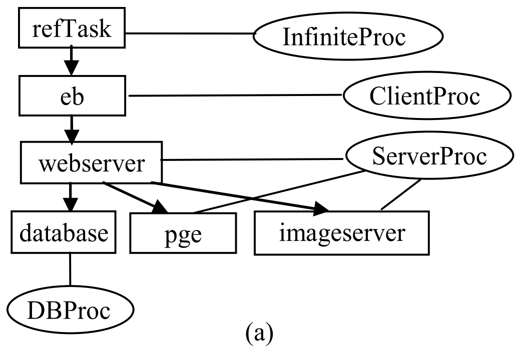
Figure 9 shows both the primary and the composed LQN models for the TPC–W `GetBuyConfirmPage` example. For simplicity, the entries and activities are hidden in these diagrams, only tasks are shown in rectangular boxes. Tasks added as a result of aspect composition are shown with a gray background. Call relationships between tasks are denoted by arrows: a) solid arrows represent synchronous calls; b) open arrows represent asynchronous calls; c) solid arrows with dashed lines represent forwarding calls. Ovals represent processors or hardware devices, while lines between tasks and processors show deployment relationship between software and hardware resources.

The primary model has a simple tiered client-server architecture with *eb* making a synchronous call to *webserver* , which also acts as a client to *database*, *imageserver,* and *pge* tasks. The composed model introduces the *ebSendSSL* and *webRcvSSL* tasks between *eb* and *webserver*, as well as the *webSendSSL* and *ebRcvSSL* tasks from *webserver* back to *eb*. The simple synchronous interaction between the *eb* and *webserver* tasks from the primary model is replaced with a forwarding chain from *eb* to *ebSendSSL*, *webRcvSSL*, *webserver*, *webSendSSL*, and finally *ebRcvSSL*. In this model the *eb* task still blocks waiting for a reply, but the reply is generated by the last task in the forwarding chain, *ebRcvSSL*.

5.2 Performance Results

The LQN performance model can be solved by either the analytical solver LQNS or simulation solver LQSim [23]. LQNS solves models mathematically, and is faster than LQSim. It works very well for models with synchronous messages, but does not handle as well models having a mix of synchronous with a lot of asynchronous messages. LQSim takes a longer time to solve a model, but gives more accurate results for complex models, especially those containing a lot of forwarding/ asynchronous interactions mixed with synchronous ones. In our case, the LQN models for `GetCustRegPage` (which are smaller) were solved with LQNS, and the models for `GetBuyConfirmPage` with LQSim.

The performance results obtained from the solvers include throughputs and service times (including queuing delays) for software resources, and utilization of both hardware and software resources. The simulator also gives the confidence intervals for all the results. The response times obtained from LQSim for the `GetBuyConfirmPage` scenario are accurate within ± 2-3% at 95% confidence level. Figure 10 shows the response times for the two scenarios studied in this paper, each giving the results for the respective primary and composed model. The impact of the *SSLtransfer* aspect on the scenario performance is noticeably different for the `GetCustRegPage` and the `GetBuyConfirmPage` cases.



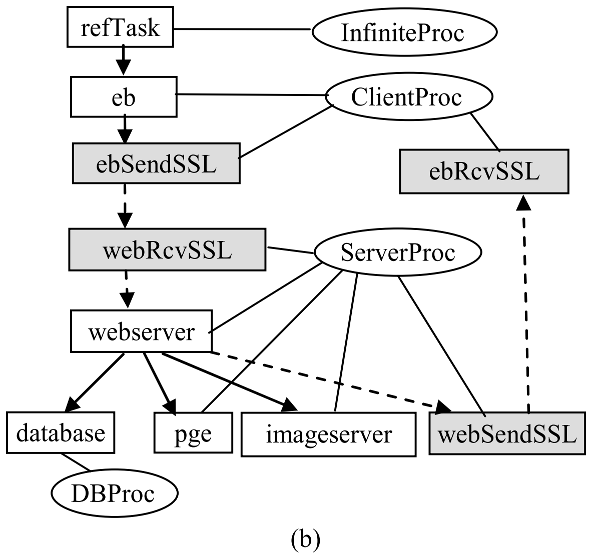


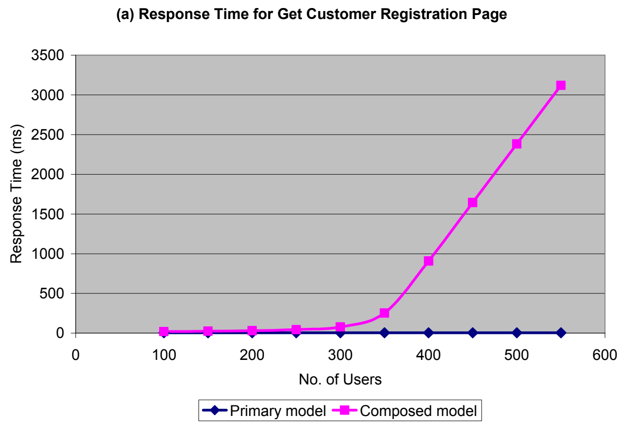
Figure 9. LQN model for GetBuyConfirmPage a) primary model; b) composed model

The `GetCustRegPage` is a light-weight scenario, which simply creates the content for a small webpage and returns it to the `EB` client. Therefore, as shown in Figure 11.a, the primary model for GetCustRegPage has a response time of less than 5ms and does not saturate even with more than 500 simultaneous users. Additional experiments show that the primary model can support 2000 users executing light-weight scenarios without saturation. However the composed model with SSL saturates with 350 users. The strong performance impact is due to the fact that the extra resource demands introduced by the aspect itself are much larger than the demands of the original scenario. The bottleneck occurs in the *webSendSSL* task, which has a utilization of 97%. This task is introduced by the security aspect, and is responsible for encrypting and sending messages from *webserver* to *eb*.

The situation is different for `GetBuyConfirmPage` where the primary model has a much heavier workload. As shown in Figure 11.b, both the primary and composed model start to saturate at a rather low number of users (less than 20). An analysis of the performance results shows that *webserver* is the bottleneck in both models. The security aspect adds even more workload to the bottleneck task and thus increases the response time, but does not move the bottleneck elsewhere. The increase in response time due to the *SSLtransfer* aspect is of about 16% for 70 users and 36% for 80 users. The experiments show that the performance effect of the *SSLtransfer* aspect on a light-weight scenario is more dramatic than on a heavy-weight scenario. If the aspect increases the demand on a resource that is already the bottleneck, then it increases the bottleneck level and makes it appear earlier. If it increases the demand on a resource that is not a bottleneck, then the bottleneck may move from other resources to this one.

We are planning on carrying out a more comprehensive performance analysis of the impact of SSL on TPC-W. For this, it is necessary to model in UML all the TPC-W scenarios, compose the primary model with both SSL aspects (handshake and data transfer) and study the performance of the whole system under the workload mix prescribed by the standard.

Gaining insight into the performance effects of different security solutions can help the designers to make tradeoffs between security and performance solutions, in order to satisfactorily balance competing system requirements.



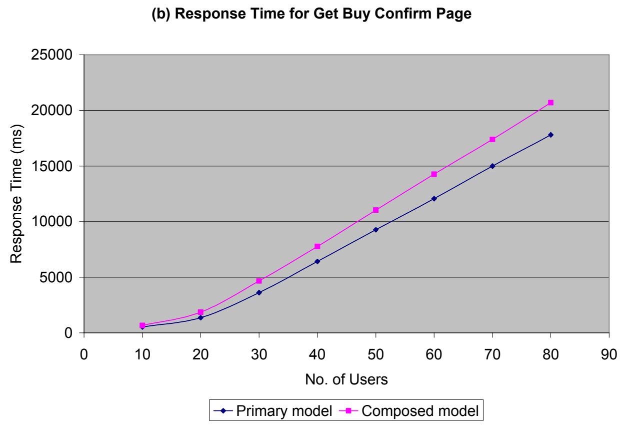


Figure10. Performance results for: a) GetCustRegPage, and b) GetBuyConfirmPage

6. CONCLUSIONS

This paper proposes a novel approach for composing aspects with the primary model at the Core Scenario Model (CSM) level. Aspect oriented modeling (AOM) allows software designers to separately address solutions for crosscutting concerns. We are applying AOM to enhance a system with security solutions; then we analyze the performance effects of these solutions on the overall system performance. Previous work has been done to compose aspects with the primary model at the UML level. Doing the composition at the CSM level has the following advantages: simpler composition algorithms due a much simpler CSM metamodel compared to UML and the ability to compose scenarios defined in UML with a mixture of behaviour diagrams (activity, sequence, communication, interaction overview).

This paper is a step toward the longer term goal of integrating security solution tradeoff analysis and performance analysis in the same development process. Integration of these techniques will enhance the current ability of analyzing separately security and performance of the same UML model, with the capability of cross-analyzing the effects of security mechanisms on system performance, and vice versa. This will allow designers to make tradeoffs between security and performance solutions. Another goal of the current research is to develop automated tool support for an integrated process, starting from the existing separate tools.

7. REFERENCES

[1] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M., "Model-based performance prediction in software development: a survey", IEEE Transactions on Software Engineering, Vol 30, No.5, pp.295-310, May 2004.

[2] Barros, J.P., and Gomes, L. "Towards the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach", Fourth Workshop on Aspect-Oriented Modeling with UML, San Francisco, 2003.

[3] Espinoza, H., Dubois, H., Gerard, S., Medina, J., Petriu, D.C. and Woodside M., "Annotating UML Models with Non-Functional Properties for Quantitative Analysis," in *MoDELS 2005 Workshops* (Jean-Michel Bruel, Ed.), LNCS 3844, pp. 79-90, Springer-Verlag, 2006.

[4] France, R., Ray, I., Georg, G. and Ghosh, S., "An Aspect-Oriented Approach to Early Design Modeling," *IEE Proceedings - Software*, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4):173-185, August 2004.

[5] Franks, G., "Performance Analysis of Distributed Server Systems," Ph.D. Thesis, Carleton University, Systems and Computer Engineering, Report OCIEE-00-01, Jan. 2000.

[6] Ho, W.M., Jézéquel, J-M., Pennaneac'h, F., Plouzeau, N., "A Toolkit for Weaving Aspect Oriented UML Designs", Proc. of the 1st Int. Conference on Aspect-Oriented Software Development AOSD'2002, pp.99-105, Enschede, The Netherlands, 2002.

[7] Houmb, S.H. and G. Georg, G., "The Aspect-Oriented Risk-Driven Development (AORDD) Framework", In O. Benediktsson et al., editor, Proc. of the Int. Conference on Software Development (SWDC.REX), pp 81-91, Reykjavik, Iceland, 2005.

[8] Houmb S.H., Jürjens, J., Georg, G., France, R. "An integrated security verification and security solution trade-off analysis", In Integrating Security and Software Engineering: Advances and Future Vision. Mouratidis, H. and Giorgini, P. (eds). Idea Group Inc., 2006.

[9] Jurjens, J., *Secure systems development with UML*. Springer-Verlag, Berlin Heidelberg, 2004.

[10] M. Mahoney, A. Bader, T. Elrad, and O. Aldawud. Using Aspects to Abstract and Modularize Statecharts. In Proc. 5th Wsh. Aspect-Oriented Modeling, Lisboa, 2004.

[11] Menascé, D., "Security Performance", *IEEE Internet Computing,* vol. 7, nb. 3, pp 84-87, May/June 2003.

[12] OMG, *UML Profile for Schedulability, Performance, and Time*, (formal/05-01-02), January, 2005.

[13] OMG, *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP*, realtime/05-02-06, 2005.

[14] Petriu, D.B. and Woodside, C.M., "A Metamodel for Generating Performance Models from UML Designs", in Proc UML 2004, LNCS 3273, pp. 41-53, Springer 2004.

[15] Petriu, D.B. and Woodside, C.M., "Software Performance Models from System Scenarios", *Performance Evaluation*, Volume 61 , Issue 1, pp.65-89, Elsevier 2005

[16] Petriu, D.C. and Woodside, C.M, "Performance Analysis with UML," in *UML for Real*, (B. Selic, L. Lavagno, and G. Martin, eds.), pp. 221-240, Kluwer, 2003.

[17] Reddy, Y. R., Ghosh, S., France, R. B., Straw, G., Bieman, J. M., McEachen, N., Song, E., Georg, G., "Directives for Composing Aspect-Oriented Design Class Models", in A. Rashid, and M. Aksit (eds*). Transactions on Aspect-Oriented Software Development I*, LNCS 3880, pp 75-105, Springer, 2006.

[18] Shen, H., Petriu, D.C., "Performance Analysis of UML Models using Aspect Oriented Modeling Techniques", In Model Driven Engineering Languages and Systems, (L.Briand and C. Williams, Eds). LNCS Vol. 3713, pp.156-170, Springer, 2005.

[19] Smith, C.U., *Performance Engineering of Software Systems*, Addison-Wesley Publishing Co., New York, NY, 1990.

[20] Straw, G., Georg, G., Song, E., Ghosh, S., France, R., Bieman, J.M., "Model Composition Directives", In Proc. «UML» 2004 - Modelling Languages and Applications, 7th Int. Conference, Lisbon, Portugal, LNCS 3273, pp 84-97, Springer 2004

[21] Transaction Processing Performance Council, www.tpc.org.

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

[23] –, LQN Online Documentations, http://www.sce.carleton.ca/rads/lqn/lqn-documentation.