Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML Models

Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Jing Xu, Tauseef Israr
Carleton University, Systems & Computer Eng., Ottawa, ON, Canada, K1S 5B6
{cmw | petriu | dorin | xujing | tisrar} @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 University of Science and Technology, Trondheim, Norway
siv.hilde.houmb@idi.ntnu.no

Jan Jürjens
The Open University, Computing Department
j.jurjens@open.ac.uk

Abstract

Aspect-Oriented Modeling (AOM) allows software designers to describe features that address pervasive concerns separately as aspects, and to systematically incorporate the features into a design model using model composition techniques. The goal of this paper is to analyze the performance effects of different security features that may be represented as aspect models. This is part of a larger research effort to integrate methodologies and tools for the analysis of security and performance properties early in the software development process. In this paper we describe an extension to the AOM approach that provides support for performance analysis. We use the performance analysis 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 composition of the aspects with the primary (base) model is performed at the CSM level. A new formal definition of CSM properties and operations is described as a foundation for scenario-based weaving. The proposed approach is illustrated with an example that utilizes two standards, TPC-W and SSL.

Keywords: Software Performance Engineering, Aspect-Oriented Modeling, Scenarios, Layered Queueing, Security.

1. INTRODUCTION

Security and performance goals may easily be in conflict in a complex distributed system with sensitive data and many users. An example is the redesign of the SSL (Secure Sockets Layer) protocol proposed in [12], which was motivated by the goal to reduce the performance cost on the server in a SSL connection. While the performance goal was achieved, the main security goal of the SSL protocol, namely to set up a secure connection, was violated in the redesigned version (see [2] for more details). If both security impacts and performance impacts are analyzed early in the software development, then drastic and expensive changes may be avoided. The work presented in this paper is part of a larger research effort by the authors to integrate methodologies and tools that support the analysis and tradeoff of security and performance early in the software lifecycle.

A general approach for analyzing the quality of a design expressed in UML is to annotate the UML specification with properties needed for a particular analysis, extract an analysis-domain model, and determine the properties from it using tools from that domain. This approach underlies security analysis in [10],[12],[13], as well as many efforts in performance analysis [3], of which the PUMA system [31],[32] is an example. In PUMA, different kinds of UML views are transformed into a single intermediate scenario form, which captures the use of resources by the system behaviour, before generating a performance model. The scenario form, called Core Scenario Model (CSM) may be derived from different kinds of UML diagrams and supports different performance formalisms.

In Aspect-Oriented Modeling (AOM), features that address pervasive or "cross-cutting" concerns are described separately as aspect models (e.g., for security, [6],[25]). Aspect models are composed with a primary design model expressed as UML class diagrams [6],[26], interaction diagrams [6],[14],[29], statecharts [8],[14], and activity diagrams [4],[25],[27]. An aspect or primary model may require different diagram types; for example, an aspect model can consist of both a class diagram and a sequence diagram. In this case, the model composition takes place by independently composing diagrams of the same type, which raises the need to ensure that information is represented consistently across the different types of diagrams in the composed model. Transforming the different types of diagrams to an intermediate form such as CSM and performing the composition on the intermediate form eliminates the need to check for consistency across different diagram types in a composed model.

Previous work on the performance effects of security aspects modeled in UML employed the PUMA approach to evaluate performance after composing the security aspects at the UML level [25],[27]. Another approach for studying the performance impact of security aspects is presented in [5], where the aspects are first woven into a UML architecture design, which is then translated into Rapide, a formal architecture description language, allowing for the simulation of a system's response time.

Performance concerns require changes to the usual aspect-oriented modeling and model composition, including the use of concurrency and deployment model views that are essential for performance analysis; also, special attention must be paid to resource usage, and to composing the performance parameters. Here, the primary model and aspect models are converted to the CSM form first, and then composed. The advantages of CSM-level composition are three-fold:

- The UML models can include any of the UML behaviour formalisms, and can combine a mixture of interaction diagrams, activity diagrams and statecharts with class and deployment diagrams into a single analysis. These UML diagrams are annotated with performance properties using the standard UML Profile for Schedulability, Performance and time (SPT) [17]. The aspect weaving algorithm must address all the possible combinations.
- The extracted CSMs are smaller than the original UML model, because CSM has a much smaller metamodel specialized for scenario modeling. Therefore the composition process is both simpler and more robust, as it does not involve composing a wide variety of model elements in different kinds of diagrams.
- The CSM form combines behaviour, resources, component instances and performance attributes (such as CPU consumption; an aspect may even have different CPU consumption when composed in different locations). Thus CSM has all the necessary information, in a form suitable for the composition which must address all these factors.

The CSM is designed particularly for performance analysis, but the concept can be extended to other analysis domains that are based on scenarios (such as reliability and schedulability). Figure 1 shows the processing flow used here. This paper is concerned with the step labeled “compose”. In order to be able to integrate the aspect composition tool with existing model transformation tools (from UML to CSM, and from CSM to performance models) a requirement for this work is to keep the existing CSM metamodel unchanged.

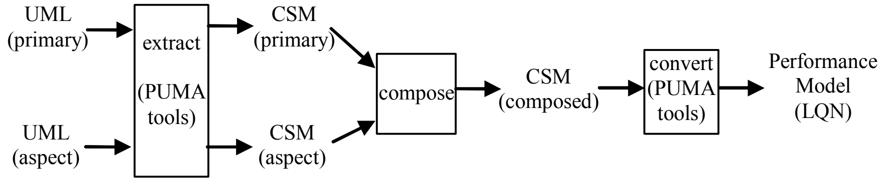

Figure 1. Flow of processing to combine aspects in CSM and to evaluate performance

Other work on scenario composition includes [14] (not necessarily for aspects), considering very general scenarios but without alternative paths. Conditions were found for correctly inserting scenarios at given points. A scenario-related version of aspects, specified as a kind of process algebra (which does include alternative paths), was also considered by Andrews [1].

Point cuts (specification of locations for composition) were given as conditions expressed in predicate logic. The CSM description used here includes scenarios with alternative paths and describes the behaviour as a directed graph, which is closer to the behaviour representation in UML. It uses predicate logic to specify point cuts.

A preliminary version of this work in [24] used an ad-hoc composition of security aspect behaviour into the primary scenario, at locations (join points) specified directly by the designer. This paper develops a new formal framework for general aspect composition of scenarios in CSM, useful for all kinds of aspects. Properties of the CSM model elements are expressed in a form suitable for operating on the model, and operations that are useful for AOM are defined. Very general predicate-logic conditions govern the location and style of composition, and the composition of quantitative performance attributes is specified separately. Special attention is paid to point cut conditions that are useful for security. This paper also extends the example studied in the previous paper [24] to illustrate the case where the aspect and primary models are specified with different kinds of UML diagrams (for instance one as an activity diagram, and the other as an interaction diagram).

The paper is organized as follows: section 2 briefly describes the PUMA approach for transforming UML+SPT models into performance models; sections 3 and 4 discuss aspect-oriented modeling of security mechanisms and describes the example system based on TPC-W and SSL, both in UML and in CSM; section 5 introduces formal definitions of CSM operations required for expressing aspect composition, section 6 introduces a new formal approach for composing aspects at the CSM level; section 7 presents the LQN model and some performance results and section 8 gives the conclusions and future work.

2. PERFORMANCE ANALYSIS OF UML MODELS

To support early performance analysis of software specifications, UML has been extended by the standard SPT Profile ([17], currently being upgraded [18]), 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 2.

A wide variety of performance modeling approaches can be applied to software [3],[28]. Performance by Unified Model Analysis (PUMA) [31] unifies the creation of performance models from software design models by using an intermediate performance metamodel called Core Scenario Model (CSM) introduced in [19] and revised in [20] (which is the version used in this paper). As shown in Figure 2, CSM captures the information needed for performance analysis from design models created by any UML tool and from different UML representations of system structure (e.g., class, component, deployment diagrams) and behavior (e.g. interaction diagrams, activity diagrams). CSM provides a single source for creating different kinds of performance models (e.g. queueing, layered queueing, Petri nets, simulation) as described in [31],[32]. In this paper we show how security aspects can be inserted into the primary model of a system at the CSM level, and we do the performance analysis with a layered queueing model (LQN) [34].

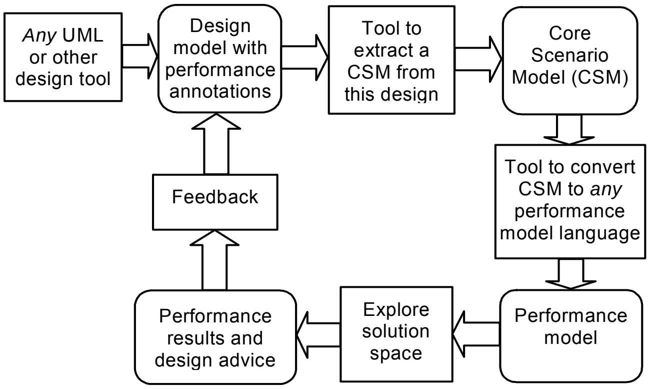
Any UML or other design tool
Design model with performance annotations
Tool to extract a CSM from this design
Core Scenario Model (CSM)
Tool to convert CSM to *any* performance model language
Performance model
Explore solution space
Performance results and design advice
Feedback

Figure 2. Tool interactions and information flows in the PUMA architecture [31]

A CSM model defines a set of *Scenarios* composed of operations (called *Steps*) with precedence relationships and resource requirements and demands. Precedence patterns include sequence, fork/join, and 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, while 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 arrival of requests to execute the scenario. Examples are given in Section 4 below.

2.1 The Core Scenario Model (CSM) from PUMA

The CSM metamodel in Figure 3 (from [20]) is based on the domain model for performance annotations in the SPT profile [17],[22], and SPT stereotypes are mapped to CSM classes, and SPT tagged values to CSM class attributes. Computations are represented by *Steps*, with precedence structure established by *PathConnections*. A *Step* is executed by a *Component*, and uses *Resources*. A *Step* may represent a "basic" computation, or a composite step refined by a sub-scenario (specified by the "subscenario" association role in Figure 3). Two specialized steps, *ResourceAcquire* and *ResourceRelease*, are provided to acquire and release resources explicitly. A CSM model is a collection of top-level scenarios and sub-scenarios, where the top-level scenarios are driven by open or closed *Workloads* representing arrival of requests for computation, and the sub-scenarios are refinements of steps.

The CSM model elements have attributes that correspond to SPT tagged values and serve as inputs for performance analysis, including host execution demands, repetition counts, branch probabilities, arrival rates or number of users. The instances of *PathConnection* subclasses (*Sequence*, *Branch*, *Merge*, *Fork*, *Join*) are derived directly from the UML behaviour specification, while the *Components* and host resources (*Processors*) are derived from both behaviour and deployment diagrams.

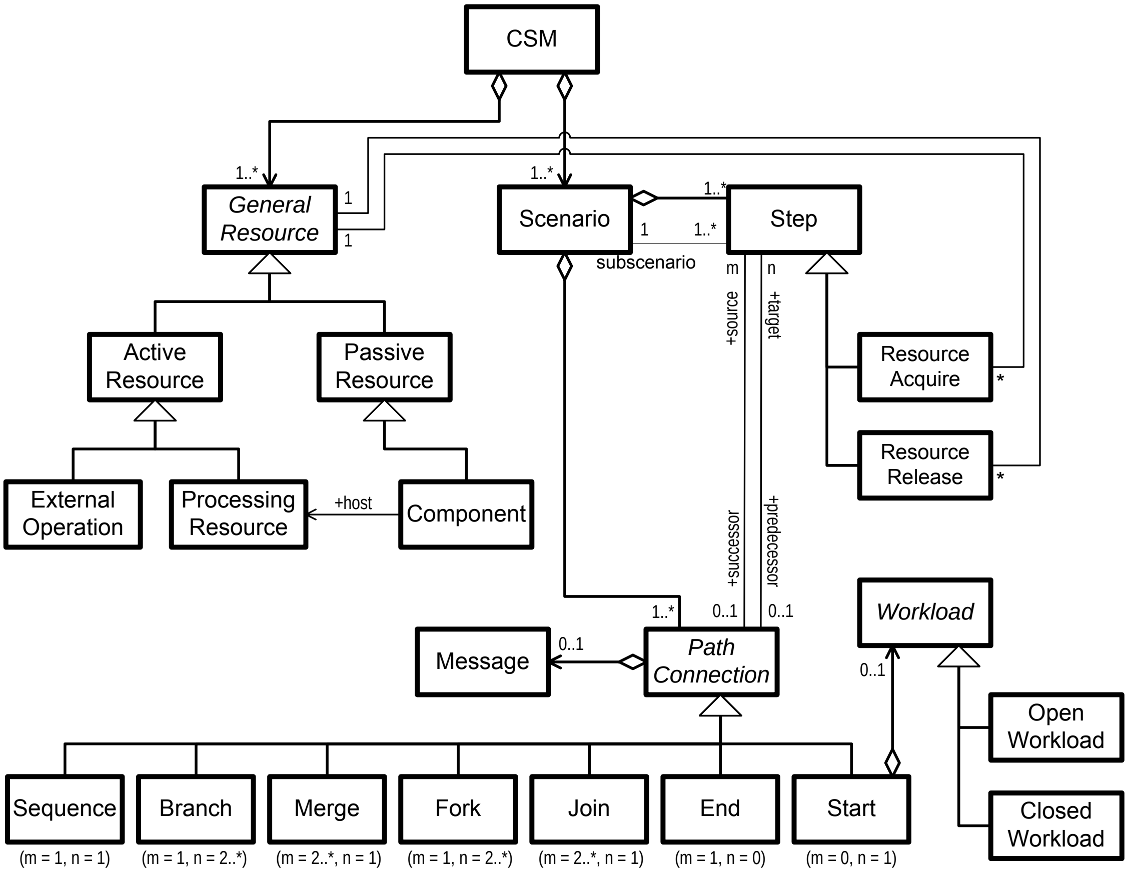

Figure 3. The Metamodel for CSM (the Core Scenario Model)

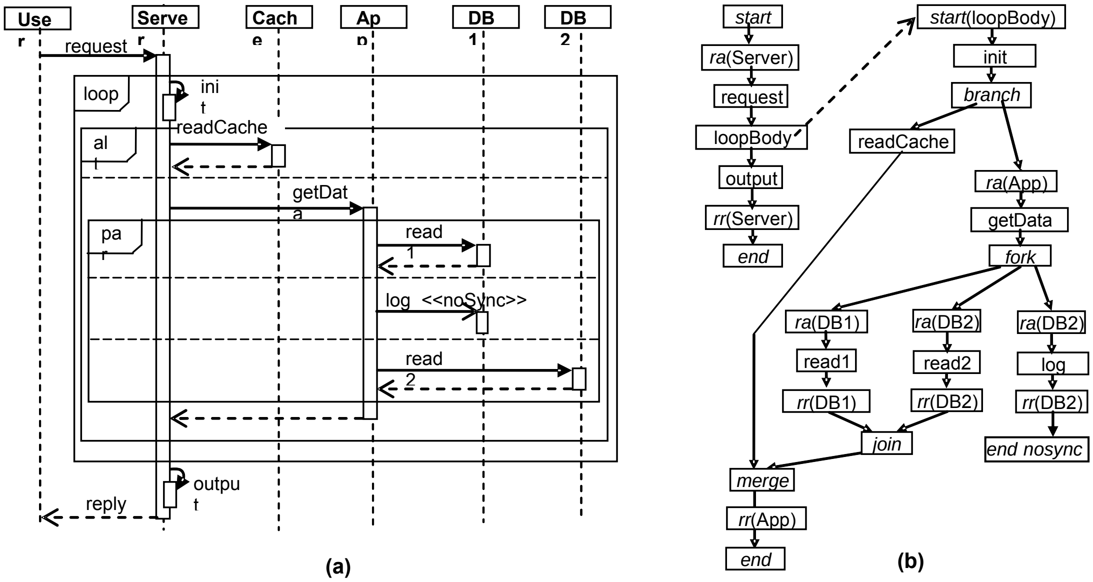

Figure 4. Behaviour features in: (a) UML sequence diagram, and (b) the corresponding CSM

Figure 4 shows how various features of a UML Sequence Diagram are represented in CSM. The top-level scenario starts with `User` sending a `request` message to `Server`. This is represented in CSM by a *ResourceAcquire* step `ra(Server)` (indicating that the `Sever` resource must be acquired first), followed by the `request` step, which represents the UML *ExecutionOccurence* that is the effect of receiving the message. The next step, `loopBody,` is refined as a sub-scenario whose starting step is linked to its owner by a dotted arrow. After completing `loopBody`, the server executes the step `output` invoked by a self-message, then replies to `User`, effectively completing its service (represented by a *ReleaseResource* step `rr(Server))`. The effect of the `reply` message is the step `end` of `User`. Note that the actual messages in the sequence diagram are not represented as CSM steps, but as CSM model elements of type *Message* attached to the path connector preceding the step that corresponds to the effect of that message. The `loopBody` sub-scenario begins with the `init` step, followed by a branch connector with two branches that are merging later. The right branch is the most interesting, as it contains a parallel structure forking three other branches, but only two join later. One of the messages sent in parallel, namely `log,` is stereotyped `<<noSync>>`, which indicates an asynchronous message starting a parallel branch that will not join. This stereotype is not in SPT, but was added in [19],[20] and [31].

3. ASPECT-ORIENTED MODELING OF SECURITY MECHANISMS

An aspect-oriented model includes a base model called the *primary model*, which reflects core design decisions, and a set of *aspect models*, each describing a system property, or feature, that crosscuts the primary model [6]. Aspect models are composed with the primary model to produce an integrated design model.

An aspect model defines a crosscutting feature as a pattern, independent of any primary model it may be composed with. As in [6] it is termed a *generic aspect model* and may be thought of as a template with formal parameters. For each insertion in the primary model the template is instantiated and its formal parameters are bound to values from the primary model, using binding rules, to give a *context-specific aspect model.*

As in other AOM work (e.g., [1],[14]) we use here terms from Aspect-Oriented Programming: an *advice* is an aspect model that will be *woven* into (*composed* with) the primary model; a *point cut* is a specification of conditions that identify locations in the primary model where the aspect should be woven, and a *join point* is a location (i.e., a set of model elements) in the primary model that matches the point cut specification; there may be several such locations.

3.1 UML primary model of an on-line bookstore

CSM scenario weaving will be illustrated with an example based on TPC-W, a benchmark of the Transaction Processing Performance Council [30], 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) a tier consisting of a set of emulated web browsers (EB), b) a web tier that includes Web Servers, Image Servers and Web Caches and c) a persistent storage tier. TPC-W emulates customers browsing and buying products from a website, with 14 different web pages that correspond to typical operations performed by a customer. The user starts at the "Home" page which includes the company logo, promotional items and navigation options to best selling books, new books, search pages, the shopping cart, and order status pages. At every page, the user is offered a selection of pages that can be visited next. The user may browse pages containing product information, perform searches with different keys and put items in the cart.

The pages for ordering and buying will be made secure. A new customer has to fill out a customer registration page; for returning customers, the personal information is retrieved from the database. 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 obtains credit card authorization from a Payment Gateway Emulator, PGE, and presents the user with an order confirmation page. At a later date the user can view the status of the last order.

The TPC-W benchmark specifies a workload mix in which 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 two administration web pages. Of the 20% ordering web pages, a quarter of the accesses are to secure web pages requiring SSL encryption. This paper focuses on the composition of security aspects, so we consider only two scenarios that access secure pages: a simple one shown in Figure 6 that returns the customer registration page and a more complex one shown in Figure 7, to buy a product. To support performance evaluation the UML model also includes the deployment of the major software components to hardware devices shown in Figure 5. Deployment shows the software components, their corresponding artifacts and the deployment of artifacts on processing nodes. The `DBProc` node is stereotyped as `<<PAhost>>` to show it is a host, and as `<<PAresource>>` to indicate that it is a computational resource.

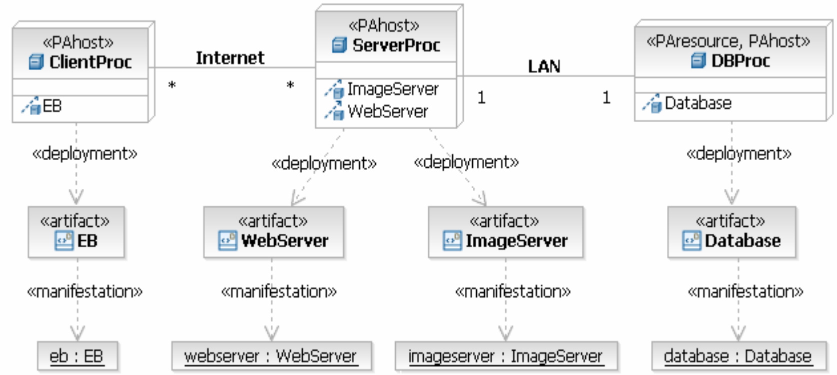

Figure 5. UML deployment diagram for TPC-W

The scenario `GetCustRegPage`, shown in Figure 6, 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 needed to protect private information. 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 7. The upper diagram transfers the shopping cart content into a newly created order for the registered customer, executes a payment authorization, and returns a page with the details of the 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 (modeled as an **opt** block)
  - 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 (modeled as a nested **opt** block)
- 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.

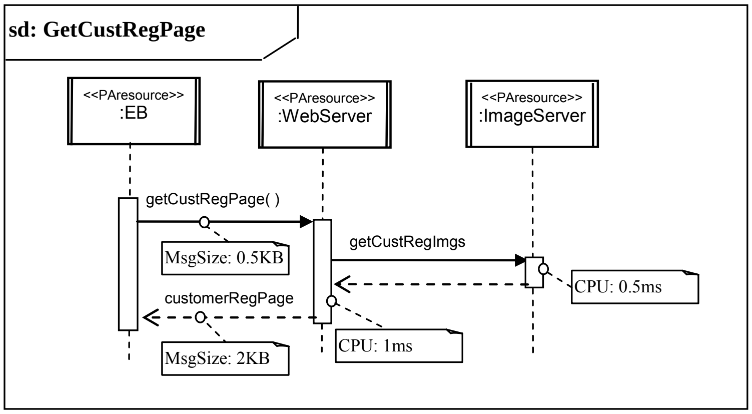

Figure 6. TPC-W UML primary model: scenario GetCustRegPage

The Checkout sub-scenario is represented by the lower diagram in Figure 7. 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 the database and the cart is cleared.

Figures 6 and 7 show the SPT annotations in an abbreviated form, without the full stereotype/tagged value syntax, to limit clutter. However, the complete stereotypes and corresponding tagged values were defined with the industrial UML editor [11] used to generate automatically the CSM models shown in the next section. An additional performance annotation for message size in kilobytes, shown as the variable $MSG_SIZE, is also shown in Figures 6 and 7.

Some example SPT profile performance annotations used in this primary model are:

- CPU execution demand PAhostDemand in milliseconds for operations (applicable to execution occurrences and messages 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, that is for nodes stereotyped as <<PAhost>>)
- PAcapacity for resource multiplicity, which may denote a multiprocessor or a multithreaded process (applicable to elements stereotyped as <<PAresource>>)
- a demand for an external operation (by a resource which is not modeled in UML), applied to operations stereotyped <<PAstep>> (here, the payment system PGE)

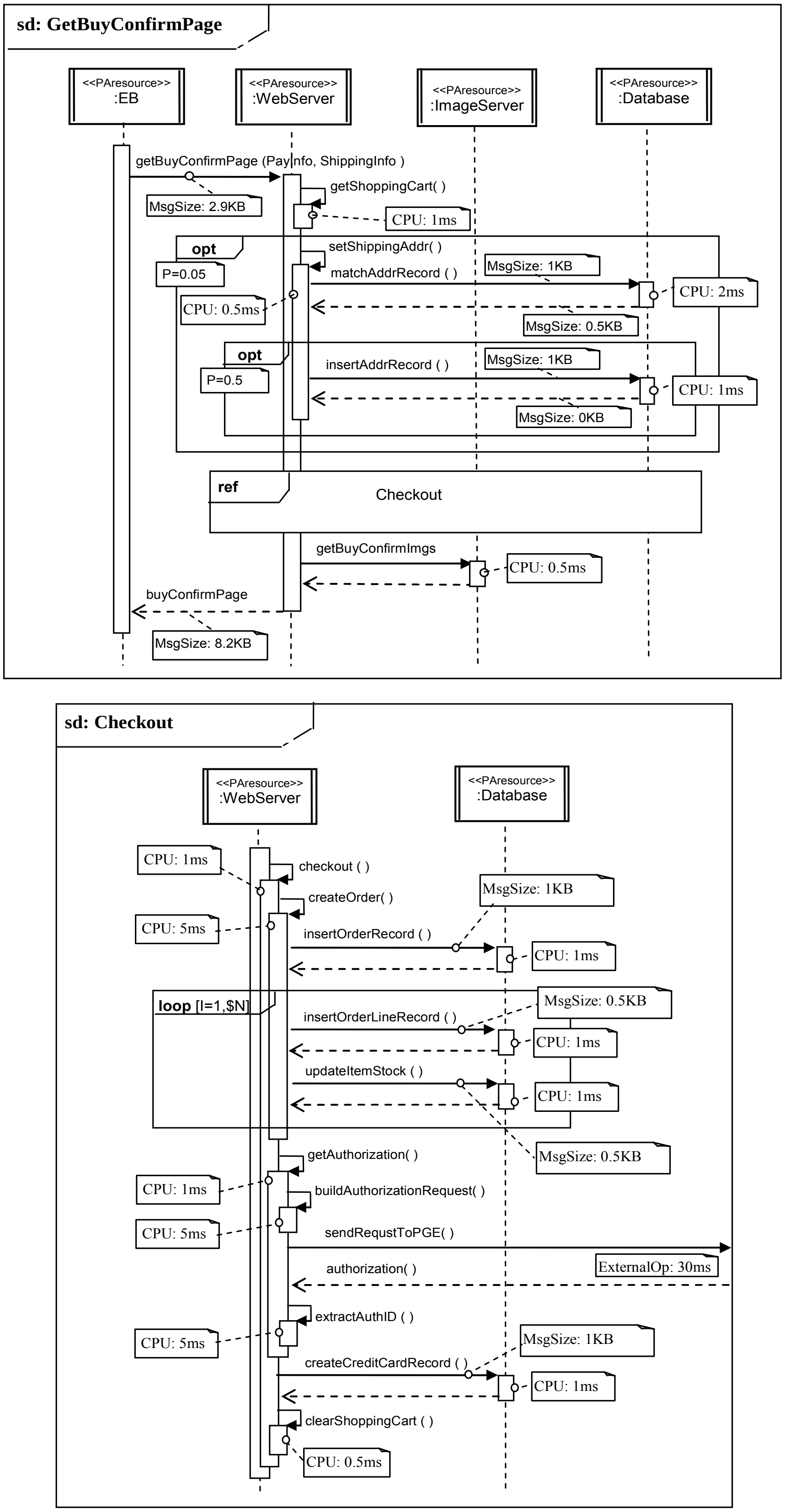

Figure 7. TPC-W UML primary model: scenario GetBuyConfirmPage

3.2 UML generic aspect advice model for SSL

SSL is the most common authentication protocol used for web-based secure transactions [16]. 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, so each represents a sub-aspect of SSL.

The generic aspect model for SSL describes the structure and behaviour of the protocol, without reference to the system to which SSL will be applied. Each sub-aspect has its own model, which may use different names for the generic roles. Figures 8(a) and 8(b) represent the deployment diagrams, and Figures 8(c) and 8(d) the behaviour for SSL handshake and SSL data transfer, respectively. All nodes are generic, and will be bound to concrete nodes in the process of instantiating the context-specific aspects, as described in section 4.2. We use the convention that generic role names start with a ’|’, similar to [6]. The SSL handshake model involves four generic roles: |server (the server side of the communication), |serverSSL (server side SSL execution), |client (the client side of the communication) and |clientSSL (client side SSL execution), and two generic processing nodes, |ServerProc and |ClientProc.

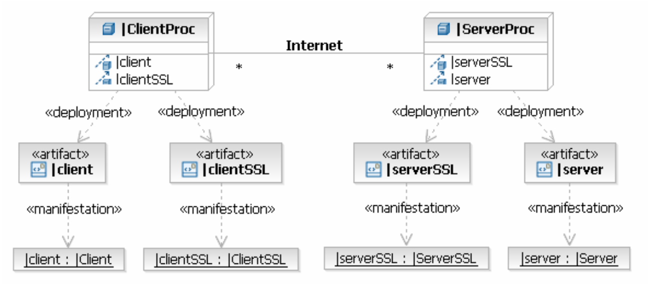

Figure 8(a). Deployment diagram for SSL handshake

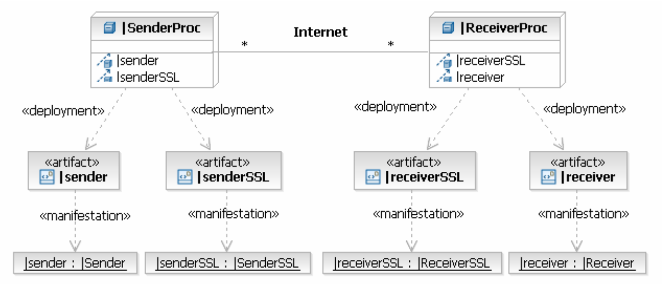

Figure 8(b). Deployment diagram for SSL data transfer

Figure 8(c) describes the behaviour of SSL handshake as an activity diagram, rather than a sequence diagram, only to emphasize that by dealing with aspects at CSM level, we have the ability to compose scenarios defined in UML with a mixture of behaviour diagrams. We demonstrate one-way authentication, where a server is authenticated to a client as part of the handshake.

The handshake phase allows the |client to authenticate the |server 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 authentication and key exchange, and symmetric encryption (which is much faster [16]) for data transfer. The handshake starts from the |client, which sends through its |clientSSL a message to the |serverSSL containing information about the SSL version, the cryptographic algorithms the |client supports, and a session identifier. The |serverSSL responds with a certificate, that the other side attempts to validate. If the server certificate is not validated by the |clientSSL, then an alert is sent to the |serverSSL, and the connection is aborted.

If the server certificate is validated, the handshake continues to set up privacy and message integrity mechanisms. The |clientSSL creates a random string from the server public key, called the pre_master_secret, which is encrypted using the same server public key, and is then sent to the |serverSSL. Symmetric keys are computed from the unencrypted version of the string, and a secret digest string (to be used for message integrity) is also created by both sides independently, without exchanging additional information. Finally, the integrity mechanism is used to ensure that the security of the handshake information has not been compromised. This is done in two steps. First, the |clientSSL creates a digest of all the handshake messages seen thus far, encrypts it, and sends it to the |serverSSL. The |serverSSL checks the digest against the same set of handshake messages. An alert is sent to the other side if the digest is wrong, otherwise the integrity check continues. Now the |serverSSL performs the same operations, including the newest handshake message from the client SSL, and this digest is checked by the |clientSSL. If the digest is wrong, the |clientSSL sends an alert to the server side, otherwise the handshake is complete. As in the case of certificate validation, if there are any errors associated with either of these digests, the entity that discovers the problem sends an alert to terminate the connection.

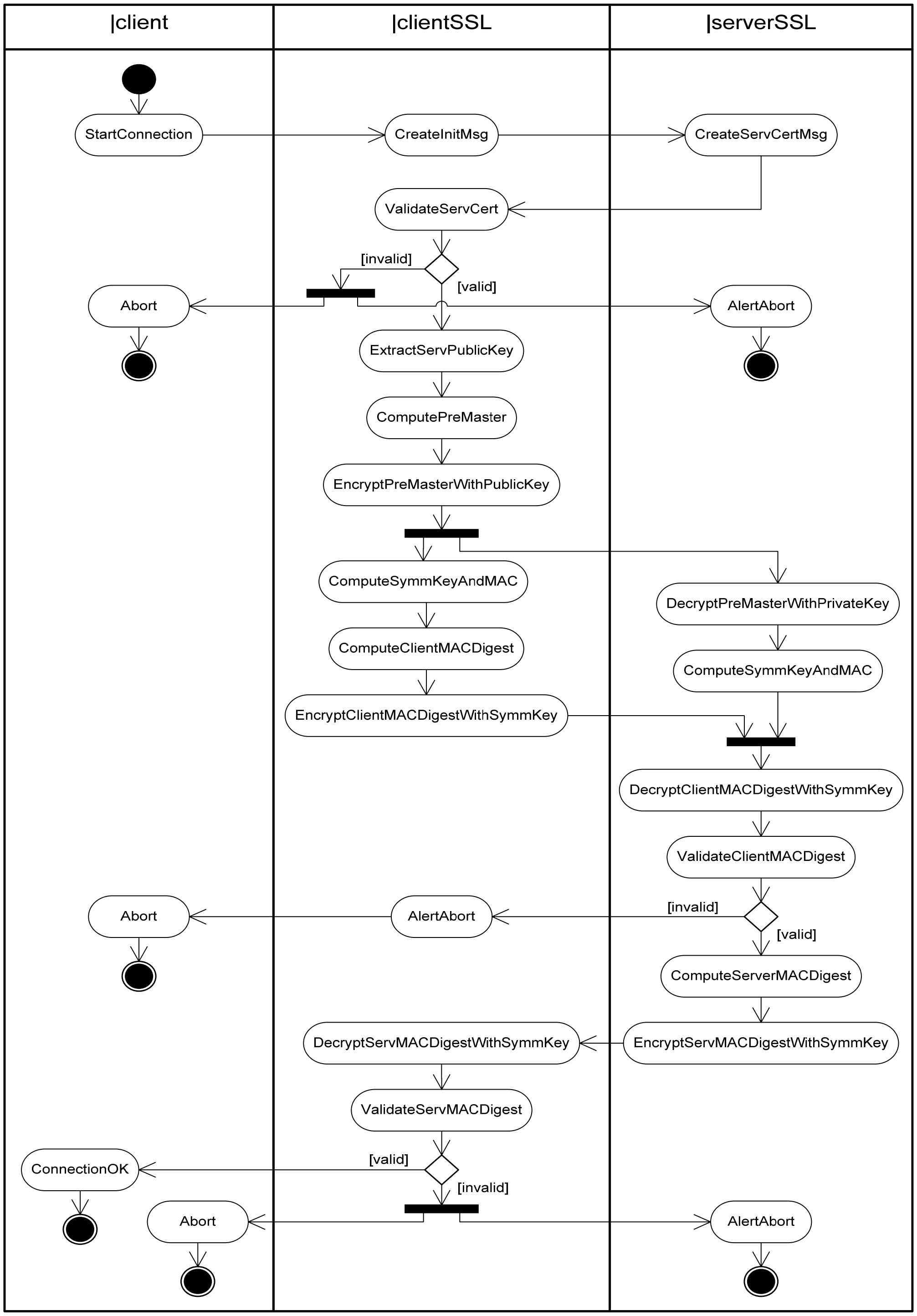

Figure 8(c). Generic SSL aspect model: behaviour view of SSL handshake

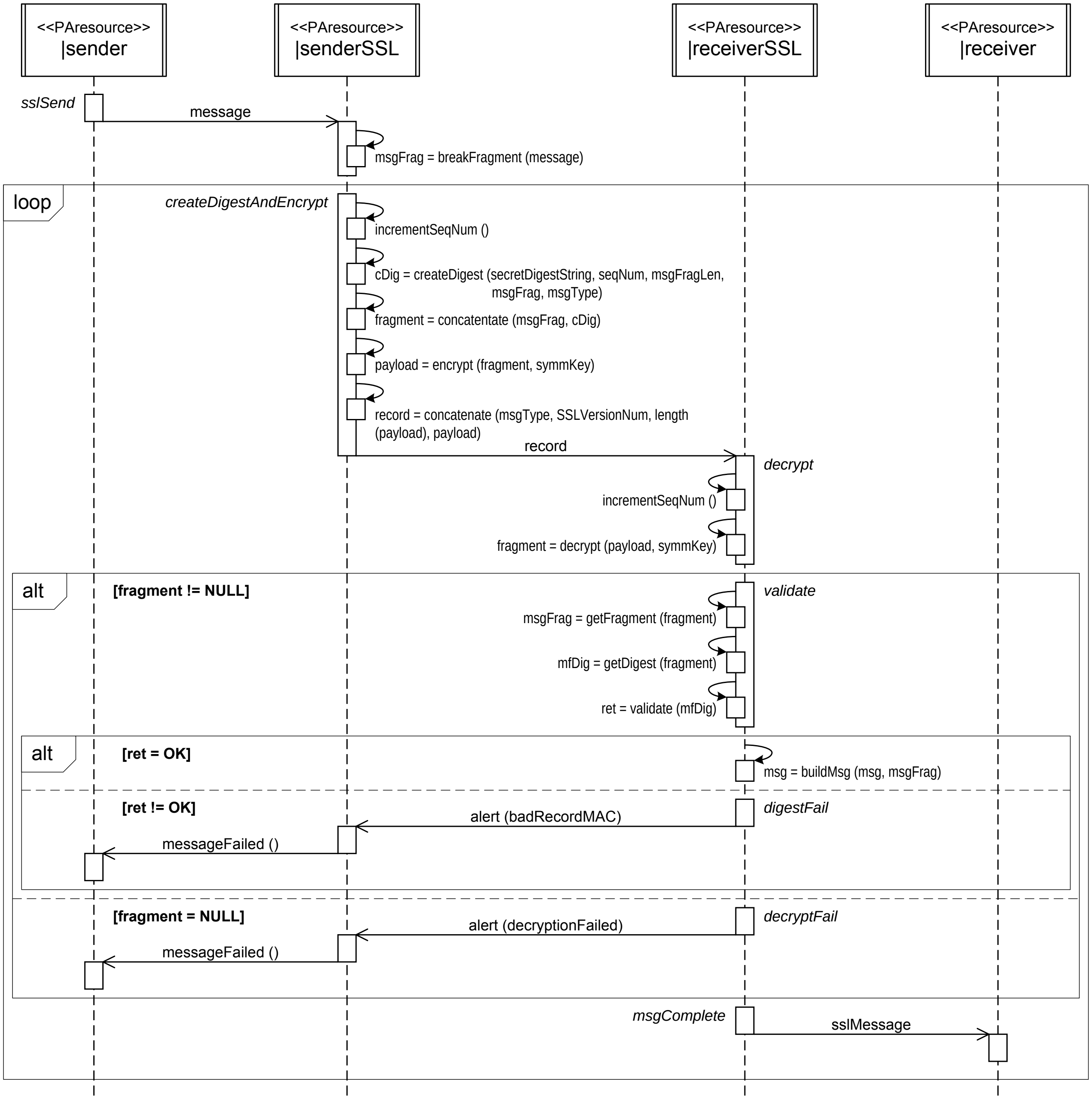

Figure 8(d). Generic SSL aspect model: behaviour view of SSL Data Transfer

Figure 8(d) describes the behaviour of the SSL data transfer as a sequence diagram. 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, a sequence number is incremented. A digest is created using the sequence number, fragment length, message type (e.g. application data or handshake), fragment, and 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. This entire entity is the record that is sent to the target; |receiverSSL increments the sequence number, 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.

The generic aspect model contains more information than is necessary for performance analysis. Hence, the level of abstraction may be raised, for instance by combining sequential steps into more abstract steps in Figure 8(d). For example, the three |receiverSSL invocations begun by receipt of the record message are abstracted into a single invocation called decrypt. These abstractions are shown in Figure 8(d) in an italicized font next to the group of invocations they abstract. Some of these names appear as *Step* names in the CSM of Figure 10. Details of abstraction are beyond the scope of this paper; please see Petriu and Sabetta [23] for details.

The SPT performance annotations (which are not shown in Figures 8(c),(d)) use variable placeholders instead of concrete values for the values PAdemand, PAprob, etc. These variables will be assigned concrete values depending on the location of aspect insertion.

4. CSM primary and generic aspect advice model

In the PUMA toolset, CSM models are automatically generated from UML+SPT models created with an industrial UML editor [11] either from sequence or from activity diagrams [31]. For this work, we used CSM models obtained with the CSMGenerator [32], an Eclipse-based RSA plug-in that traverses UML 2.0 Sequence Diagrams with performance annotations and generates one or more CSM *Scenarios* for every diagram.

UML sequence diagrams or activity diagrams with a PAworkload annotation 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 9 shows the CSM top-level scenario for the TPC-W `GetBuyConfirmPage` primary model and its `Checkout` sub-scenario from Figure 7. The top-level scenario shows the CSM Steps corresponding to the execution occurrences stereotyped as <<PAstep>> in the interaction diagram as well as explicit resource acquisition and release. A ResourceAcquire is generated whenever a lifeline first receives a synchronous or asynchronous call message or whenever execution first starts. A ResourceRelease is generated whenever a lifeline sends an asynchronous call message or whenever execution finishes.

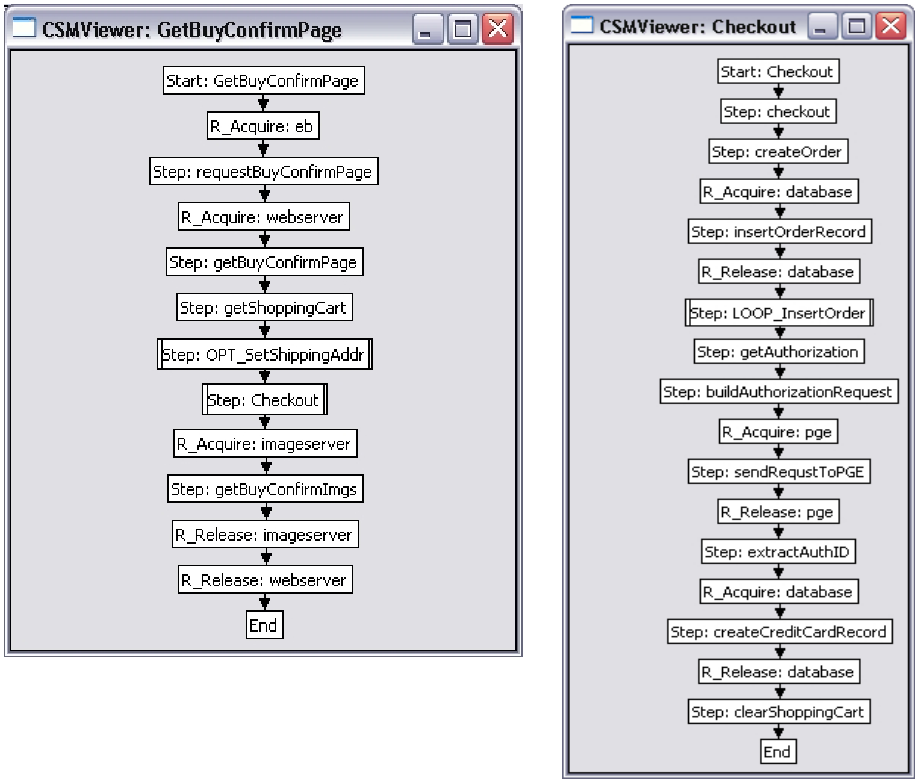

Figure 9. CSM GetBuyConfirmPage primary model

`Opt` and **`loop`** combined fragments are generated in CSM as *composite steps* with the **`loop`** interaction operands as sub-scenario *refinements* for those composite 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` composite 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` composite step and its corresponding sub-scenario (not shown).

Figure 10 shows the complete CSM scenario for the generic `SSLtransfer` aspect introduced in Figure 8(b) and 8(d). 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 PathConnections with a composite step and a sub-scenario for each alternate operand. In Figure 10 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 PathConnections with a composite step (refined by a sub-scenario) for each parallel operand.

The lifelines from the sequence diagram give rise 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.

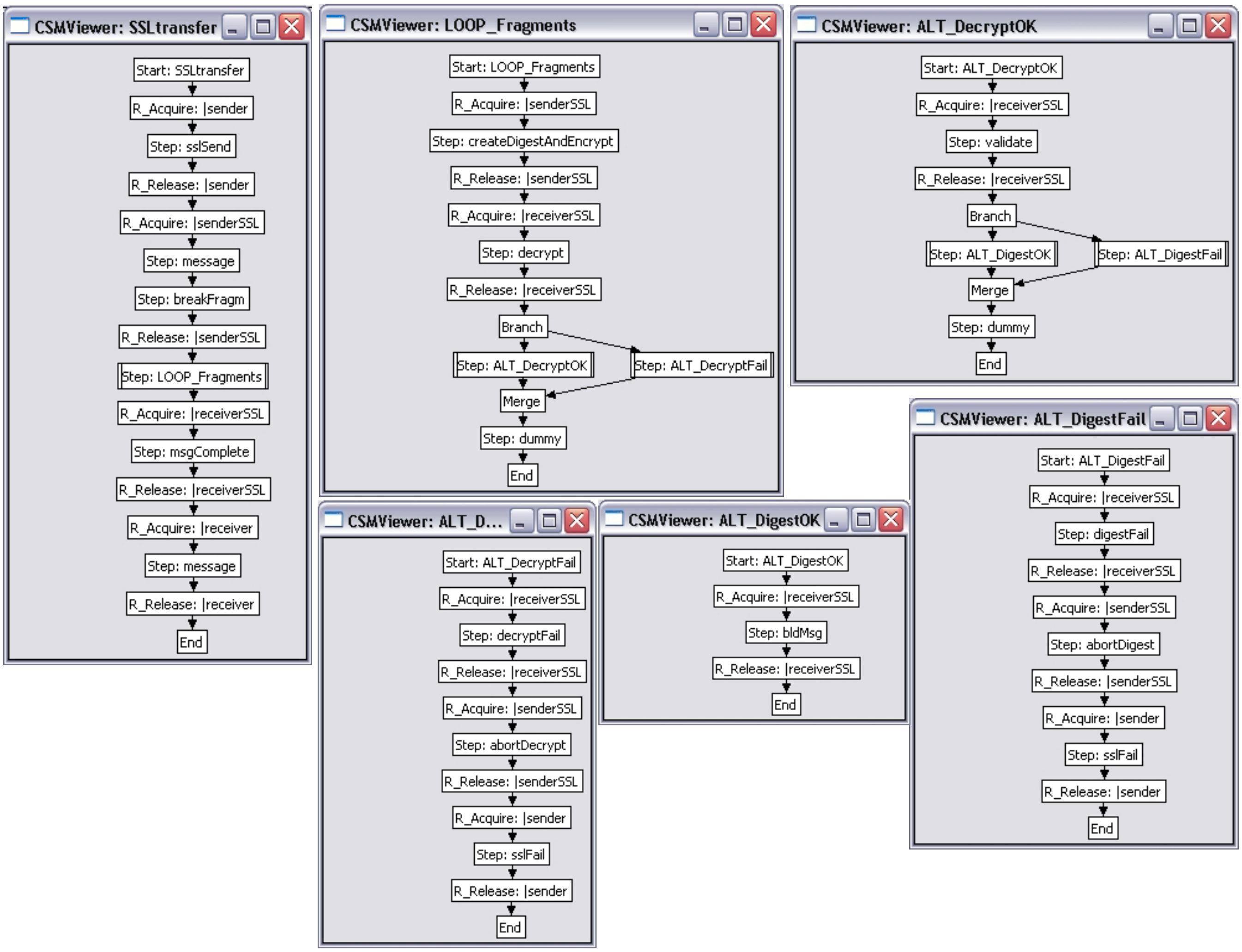

Figure 10. Generic Aspect CSM model: SSLTransfer

4.1 The Aspect Scenarios

The `SSLHandshake` scenario is created in a similar way and will not be shown here. For weaving into the primary scenario, either the `SSLTransfer` scenario is inserted for data transfer, or, the first time in the primary scenario, the combination of the two is inserted, first the handshake and then the transfer. For the later case, a `SSLCombined` scenario is constructed by adding a new top-level scenario with just two steps, as shown in Figure 11. They are composite steps, the first for `SSLHandshake`, and the second for `SSLTransfer`. Furthermore, even if it is the first transfer in the scenario, there is a probability that, at the system level, this scenario may be executed in combination with other secure scenarios, so we specify the probability `$probFirst` indicating whether we need to execute the handshake. Thus the SSL aspect has two sub-aspects, `SSLCombined` (for the first transfer), and `SSLTransfer` (for subsequent transfers).

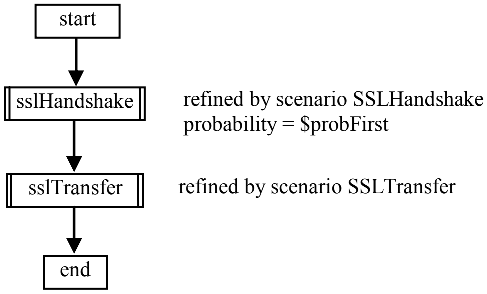
start
sslHandshake
refined by scenario SSLHandshake
probability = $probFirst
sslTransfer
refined by scenario SSLTransfer
end

Figure 11. The `SSLCombined` scenario as a CSM with Sequence connectors suppressed

5. SUPPORTING CSM COMPOSITION

In this section we introduce a notation for describing and manipulating CSM scenarios, suitable for expressing properties and operations needed to specify point cut specifications and composition-related operations. The proposed notation is referred to as CSM-Op, and is supported by a tool CSMAspects implemented as an Eclipse plug-in.

5.1 Basic Notation for Properties of Model Elements

In the following we define the CSM-op notation. Sets will be indicated in boldface type. The definition of the CSM-op notation was driven by the decision to keep the CSM metamodel unchanged in order to retain the existing tool support. All the CSM-op properties are extracted from CSM models, which in turn conform to the CSM metamodel [20]. Some properties are obtained directly from the attributes of the model element they refer to, while other properties are derived by navigating different associations between model elements.

A CSM is a set of Scenarios and a set of Resources, while a Scenario is a set of Steps, connected through PathConnections. Here we will refer to PathConnections as Connectors for readability. Since aspect composition is focused on weaving scenarios and resource information needs to be carried along, we add to the CSM-op notation for a scenario SC the resources it uses, which is a derived property:

SC = (**S**, **C**, **R**)

in which:

> **S** = **Steps**(SC) is the set of Steps, with their CSM attributes and associations as defined in [20], of which some are listed below along with some derived attributes.
>
> **C** = **Connectors**(SC) is a set of Connectors (an alias for PathConnections) with the associations of each connector to its preceding and succeeding Steps**,**
>
> **R** = **Resources**(SC) is the set of Resources used by the Steps in **S**.

Note that the resource set of a CSM model is the union of the sets **R** of all its scenarios.. We will assume that **C** includes one Start connector, one End connector, and any number of EndNoSync connectors (which represent End connectors with a label "noSync", discussed in Section 2.1).

Connectors in **C** can be described by their type and their source and target sets:

- Seq(StepA, StepB) is a Sequence type connector from StepA to StepB,
- Branch(StepA, **SetOfSteps**) and Fork(StepA, **SetOfSteps**) are branch and fork type connectors from StepA to a set of Steps, respectively,
- Merge(**SetOfSteps**, StepB) and Join(**SetOfSteps**, StepB) are merge and join type connectors, respectively,
- Start(SC) is the unique Start connector, the following Step is called First(SC).
- End(SC) is the unique End connector; the preceding Step is called Last(SC).

Point-cut specifications and composition operators may need to refer to properties (attributes and associations) of steps. These properties can be accessed using the following CSM attributes and associations:

- Step.name is the name of the action (e.g. the UML message name that invokes the Step). In this work we take this name as the name of the *service* performed by the Step.
- Step.subscenario is a scenario that refines this Step.
- Step.rep is the number of times the Step is repeated (shortened from Step.repetition count in [20])
- Step.prob is the probability this step is executed, as an optional step or after a branch connector (shortened from Step.probability in [20]).
- Step.component is the component executing this Step, i.e., the Component in the resource context (the set of held resources) which was most recently acquired. Resource contexts are described in [20] and [33].
- Step.component.host is the host ProcessingResource of this Step

Two shorthand attributes and one new attribute derived from the CSM are also part of the proposed CSM-op notation:

- Step.type is the type of message in the UML that triggers the Step. It can have one of the following values: call, reply, async. Step.type is a derived property corresponding to Step.predecessor.message.type in [20]. Its default value (if not defined, as it happens for example in activity diagrams), is async.
- Step.msgSize is a derived property corresponding to Step.predecessor.message.size, the size of the message that triggers the Step.
- Step.**ResContext** is a new attribute of Step derived from its scenario SC, the set of resources that have been acquired and not yet released, from Start(SC) up to and including this Step.

Due to the fact that steps and connectors alternate, the preceding element to StepA is always a connector, denoted •StepA. This connector is in turn preceded by a set of Steps denoted **{••StepA}**. The set of all preceding Steps and connectors (the transitive closure of precedence) is denoted **{•!StepA}**. Similarly the one succeeding connector of StepA is StepA•, its immediately succeeding Steps are the set **{StepA••}**, and the set of all succeeding steps is **{StepA•!}**. First(SC) = Start(SC)• and Last(SC) = •End(SC)**.** The following make use of the preceding expressions:

- To(SC, StepA) is a Scenario representing behaviour in scenario SC that precedes and includes StepA:

  To(SC, StepA) = ({**•!StepA**} U StepA, **Connectors'(**SC**)** U End, **Resources'(**SC)).

- From(SC, StepA) is the Scenario including and following StepA:

  From(SC, StepA) = (StepA U {**StepA•**!}, **Connectors'(**SC**)** U Start, **Resources'(**SC)).

In these expressions, **Connectors'**(SC) and **Resources'**(SC) have been pruned to remove elements referencing Steps in SC that are not in the scenario.

Below are some additional expressions that can be used to support the definition of CSM composition:

- Component(StepA) is the component executing StepA. If the property StepA.component is undefined, the most recently defined component (along any path going back in the CSM) is returned. If the backward path branches, giving a set (such as {**••StepA**}) then an arbitrary step is selected from the set.

- Name(StepA) is the name of the service performed by StepA. If the property StepA.name is undefined, the most recently defined name (along any path going back in the CSM) is returned. As for Component, if the backward path gives a set then an arbitrary step is selected from it.

5.2 Narrow and Broad Interpretation of a Scenario Definition

A scenario with one or more composite steps may be interpreted in two ways, *narrowly* including only the set of steps in that scenario, or *broadly* to include the expansion or flattening of all refinements of composite steps. In the broad interpretation of scenario SC, composite steps are expanded recursively until there is a single flattened scenario which we will term SC*. In SC* there are no composite steps. Thus **Steps**(SC*) and **Resources**(SC*) are the unions of the steps and resources of all sub-scenarios, and the set **Connectors**(SC*) is the union of the connectors, adjusted to replace the Start and End connectors of sub-scenarios with the connectors to the higher level Step.

5.3 Some Operations on Scenarios

To illustrate the use of CSM-Op, we will apply it to the insertion of an aspect scenario SZ into a primary scenario SC. In the next section a composition process that provides a context for these insertion operations will be described.

We assume the primary scenario is SC = (**STX**, **CX**, **RX**) and the advice (aspect scenario) is SZ = (**STZ**, **CZ**, **RZ**). There are three possible ways in which SZ can be inserted into a primary scenario: replace a step (Rule 1), before a step (Rule 3) and after a step (Rule 4).

Rule 1: (replace StepA by an aspect scenario; uses Rule 2 to update its connectors)

SC' = ReplaceStep(SC, StepA, SZ) is a function which returns a scenario SC' in which StepA is replaced by SZ in **Steps**(SC) and all connectors to StepA are replaced by connectors to First(SZ) and Last(SZ):

ReplaceStep(SC, StepA, SZ) = ((**STX** \ StepA)U **STZ**,
ReplaceConn(**ReplaceConn**(**CX**, •StepA, StepA, First(SZ)), StepA•, StepA, Last(SZ))
U **CZ** \ {Start(SZ), End(SZ)}, **RX** U **RZ**).

Rule 2: (update connectors when a step is replaced)

C' = **ReplaceConn**(**C**, ConnA, StepA, StepZ) is a function returning a set of connectors **C'**, in which the connector ConnA has been modified by replacing references to StepA by references to StepZ.

Rule 3: (insert a scenario SZ preceding StepA; uses Rule 2 to update its connectors)

InsertBefore(SC, StepA, SZ) = (**STX** U **STZ**,
ReplaceConn(**CX**, •StepA, StepA, First(SZ)) U **CZ** \ {Start(SZ), End(SZ)} U Seq(Last(SZ),StepA),
RX U **RZ**).

Rule 4: (insert a scenario after StepA; uses Rule 2 to update its connectors)

InsertAfter(SC, StepA, SZ) = (**STX** U **STZ**,
ReplaceConn(**CX**, StepA•, StepA, Last(SZ)) U (**CZ** \ {Start(SZ), End(SZ)} U Seq(StepA, First(SZ)),
RX U **RZ**).

Rule 5: (an additional operation to flatten any composite step; uses Rule 2 to update its connectors)

SC' = Expand(SC, StepA) is a function returning a scenario SC' in which the composite step StepA (in scenario SC), which references a sub-scenario SZ, has been flattened by inserting the Steps of the sub-scenario in place of StepA. If SC = (**STX**, **CX**, **RX**) and StepA references sub-scenario SZ = (**STZ**, **CZ**, **RZ**) then:

Expand(SC, StepA, SZ) = ((**STX ** StepA) U **STZ**,
ReplaceConn(**ReplaceConn**(**CX**,•StepA, StepA, First(SZ)) ,StepA•, StepA, Last(SZ)) U (**CZ**
\ {Start(SZ), End(SZ)}, **RX** U **RZ**).

6. CSM ASPECT COMPOSITION

This section applies CSM-Op to aspect composition. The composition is carried out under the following conditions:

- If there is more than one aspect, the aspects are composed in a specified order.
- For each generic aspect GenericA_i (comprised of a top-level scenario and possibly a set of sub-scenarios), the following are defined:
    - a point cut, that is a set of rules $\mathbf{J_i} = \{J_{i1}, J_{i2}, ..., J_{in}\}$ identifying join-point steps $\mathbf{JS}_i = \{JS_{ik}\}$ in the primary model, which are the locations for inserting the aspect,
    - a set of RoleBinding rules, which depend on the location of the join-points (elements of $\mathbf{JS_i}$) for instantiating a context-specific aspect from the generic one
    - a composition rule that determines how the aspect will be inserted at the join-points and connected to the primary model.

The composition tool accepts the following input: a) one or more CSM models representing generic aspects models; b) a CSM model representing the primary model; c) for each generic aspect: a point cut specification, role-binding rules and composition directive. The composition of an aspect with the primary model (also comprised of a top-level scenario and a set of sub-scenarios) is performed in three steps:

1. Find the locations for composition, that is determine the join-point steps, **JS** in the primary model with the help of the point cut specifications.
2. Given a join-point step $JS \in \mathbf{JS}$, apply binding rules for roles and resources to all the scenarios of the GenericA_j to obtain a corresponding context-specific aspect model SpecificA_j.
3. For each SpecificA_i apply its composition rule, which has one of the following forms:
    - InsertBefore(SC, JS, Top(SpecificA_i)) (Rule 3)
    - InsertAfter(SC, JS, Top(SpecificA_i)) (Rule 4)
    - ReplaceStep(SC, JS, Top(SpecificA_i)) (Rule 1)

where SC is the scenario of the primary model containing JS; JS is the join-point step; and Top(SpecificA_j) is the top-level scenario of the context-specific aspect.

6.1 Point Cut Specification: Transitions in Location

Locations for composition are identified by a point cut defined as a set of conditions for identifying join-point steps, and is applied to the flattened scenario SC* corresponding to SC. It expresses where the advice should be inserted, using the properties of the Steps in SC. The conditions must be customized for every aspect, but we will consider some examples that can be used in many particular cases related to security. We begin with auxiliary conditions that identify various operations between system elements.

Identifying transitions between system elements

For security aspects, an important class of potential join points is found at transitions from one host (or component) to another. The next two rules define join points at such transitions.

Let Stepα and Stepβ be dummy step names, to be matched within a flattened scenario SC*. Then a pattern for transition from one host (or component) to another, is expressed as a predicate involving Stepα before the transition, and Stepβ, after it. Stepβ is identified as any step that makes the predicate true:

Rule 6: (boolean function to identify that a Step follows a transition to a different host)

$$\text{NewHost}(\text{Step}\beta) = ((\text{Step}\beta \in SC^*) \text{ AND}$$

$$(\text{for all Step}\alpha \in \{\bullet\bullet\mathbf{Step}\beta\}, \text{Component}(\text{Step}\alpha).\text{host} \neq \text{Component}(\text{Step}\beta).\text{host})$$

Rule 7: (boolean function to identify that a Step follows a transition to a different component)

NewComponent(Stepβ) = (Stepβ ∈ SC*) AND

(for all Stepα ∈ {**●●Stepβ**}, Component(Stepα) ≠ Component(Stepβ))

In the TPC-W example, we need to specify not just the host or component, but the application service that must be protected; this is addressed below.

Identifying a call

Conditions that specify insertion around a portion of behaviour can also be specified. An example describes the body of a synchronous call, where a request is paired with a corresponding reply. The request initiated in a certain component triggers an application service starting at Stepα in some other component. After the service, some Stepβ in the second component initiates the reply. This condition can be expressed by Rule 8.

Rule 8: (boolean function to identify the scope of a synchronous request between two components)

SynchRequest(Stepα, Stepβ) = SendReply(Stepα, Stepβ) AND

IsNext(SendReply(Stepα, Stepβ), Stepα, Stepβ),

where SendReply(Stepα, Stepβ) =
(NewComponent(Stepα) AND
(Component(Stepβ) = Component(Stepα) AND
(Component(Stepβ●●) = Component(●●Stepα))

and where the IsNext( ) function is defined for any condition on two steps SA and SB as follows:

Rule 9: (boolean function to identify that Step SB is the first after SA to make a given Condition(SA,SB) true)

IsNext(Condition(SA,SB), SA, SB) = NOT Exists StepC | (Condition(SA, StepC) AND (StepC ∈ {**●!SB**}))

For secure synchronous communications, an advice for sending a request would be inserted before Stepα, and an advice for the corresponding replies would then be inserted after Stepβ. Similar conditions must be constructed to express other "around" point cuts.

6.2 Point Cut Specification: Examples for Security

Examples of point cuts for security aspects will combine transitions between system elements and the level of security to be associated with those elements and their operations. For this discussion we assume just two levels, secure and insecure.

Identifying secure system elements

It is natural to identify certain hosts or components or application services that require the addition of the security features. In TPC-W for example certain web pages are defined in the specification as secure (`GetBuyConfirm`, `Checkout` and `GetCustomerRegistration`) while others involving only browsing are not. We consider a single generic property secure, and the following predicates:

- secure(Host1, parameters) to express the condition that all communications with Host1 should be secure, with some parameters for the desired security type or level which can be used to select the aspect that provides them,
- secure(Host1, Host2, parameters) to express a condition that communications between Host1 and Host2 should be secure,
- secure(Component1, parameters) to express a condition that all access to Component1 should be secure (e.g., requiring both authentication and encryption)
- secure(Service1, parameters) to express a condition that Service1 (identified in CSM by a Step *name* property) should be secure, meaning that its requests and replies should have certain security features.

The actual condition names and sets of elements would depend on the precise security condition required. For SSL for example, suitable conditions would be authentication, confidentiality, and integrity.

Join Point to assure secure messages to a secure server component: To insert encrypted message handling for all communications involving components in the set **SecureServers**, a join point for one insertion is defined as:

Rule 10:

J= any Stepα | (NewComponent(Stepα) AND (Stepα.component ∈ **SecureServers**))

Join Point to assure secure messages to or from a secure service: An application service is identified by the operation-name of its first step (call it Stepα, with name Name(Stepα)), For secure services, the name will be placed in the set **SecureServices**. To specify secure handling of all communications involving services in the set **SecureServices**, a join point could be defined as:

Rule 11:
J = any Stepα | (NewComponent(Stepα)
AND ((Name(Stepα) ∈ **SecureServices**) OR (Name({••Stepα}) ∈ **SecureServices**))

In the SSL example, this defines the insertion condition for the SSLtransfer aspect for one-way communication in the `GetBuyConfirm` CSM, in which `getBuyConfirmPage` is in the set of **SecureServices**. The first step that satisfies the condition is J = `getBuyConfirmPage` at the fifth Step in Figure 9.

Join Point to provide authentication for first use of a secure service: To insert authentication by supplying an identifier and password before the first use of any service in a set **SecureServices**, the first use must be identified. The service is identified by the operation name of the first step on a transition to a new component (traversing a component interface), so the join point can be defined as:

Rule 12: J = FirstRequest(SC, **SecureServices**)
= any Stepα | IsNext((Name(Stepα) ∈**SecureServices**) , Start(SC), Stepα)

The IsNext predicate (from Rule 9) ensures that no step earlier in the Scenario gives a service in the set of secure services. In the TPC-W example Rule 12 can be used to insert the SSL handshake at the first attempt to enter the secure pages. IsNext can also be used to identify the first secure service involving that particular pair of requesting and responding components.

6.3 Point Cut Specification for the Example

The example has two aspects with generic advice scenarios Combined and SSLtransfer. Using the above definitions, the point cut specifications that apply to both of the primary scenarios are:

- for the SSLCombined aspect:

J_1 = FirstRequest(SC, {`getBuyConfirmPage, buyConfirmPage, customerRegPage`}),

- for the SSLtransfer aspect:

J_2 = any Stepα | (NewComponent(Stepα) AND
((Name(Stepα) ∈ **SecureServices**) AND NOT
FirstRequest(SC, {`getBuyConfirmPage, buyConfirmPage`})

The second condition is based on Rule 11. In the `getBuyConfirm` primary scenario, the first condition is satisfied by the sending from `eb` to `webserver`, and the second by the reply. In the `getCustomerRegistration` scenario, the first condition is satisfied by the reply to `eb`. The composition rules are:
InsertBefore(SC, J_1, SSLCombined), InsertBefore(SC, J_2, SSLTransfer)

6.4 Construction of Context-Specific Advice Scenarios

In Composition Operation 2, the identification of join-points allows us to construct context-specific advice scenarios. The generic aspect advice scenarios are templates, with the generic roles of the aspect as template parameters (and also with performance parameters we will call parms). The roles are components, hosts and other resources.

For example in the SSLTransfer aspect (Figures 8(b) and 8(d)) the generic scenario may be expressed with parameters as:

SSLtransfer({ |sender, |senderSSL, |receiverSSL, |receiver, |SenderProc, |ReceiverProc }, parms)

The role bindings for the scenario are specified relative to the insertion point. For the first message from eb to webserver in the primary scenarios of Figures 5, 6 and 7, |sender is bound to eb in the CSM of Figure 9, |receiver to webserver, |SenderProc to the host ClientProc, and |ReceiverProc to the host ServerProc. This leaves two unfilled roles, for which entities must be created; suppose they are called ebSendSSL and webRcvSSL. Then the desired argument list for Roles is:

RoleBindings = {eb, ebSendSSL, webRcvSSL, webserver, ClientProc, ServerProc}.

In the example, for the first message from eb to webserver, the context-specific aspect scenario will be called SSLcall:

SSLcall(parms) = SSLtransfer({eb,ebSendSSL,webRcvSSL,webserver,ClientProc,ServerProc}, parms)

The performance parameters parms are considered in Section 6.5. The role binding rules for SSLtransfer in the TPC-W example can be expressed formally as:

|sender bindsTo Component({••JS$_j$}) (that is, to the component of the predecessor step)

|senderSSL bindsTo NEW Component, named by concat(Component({••JS$_j$}).name, "SendSSL")

|receiverSSL bindsTo NEW Component, named by concat(Component(JS$_j$.name, "RcvSSL")

|receiver bindsTo Component(JS$_j$)

|SenderProc bindsTo Component({••JS$_j$}).host

|ReceiverProc bindsTo Component(JS$_j$).host

Once a context-specific aspect has been created for one insertion point, it can be re-used at other insertion points which have the same Role list. In the example, all insertion points which are messages from eb to webserver give the context-specific aspect SSLCall given above. The same generic aspect and point cut also matches points where a reply is sent from webserver to eb, in which case the same role-binding rules give different Roles and a second context-specific aspect:

SSLReply(parms) = SSLTransfer$_1$({webserver, webSendSSL, ebRcvSSL , eb, ClientProc, ServerProc}, parms)

These two scenarios are shown in the composed resulting CSM in Figure 12, inserted as composite Steps where the point cut is satisfied. The performance parameter parms are still to be determined.

6.5 Location-Specific Performance Parameter Binding

Steps and connectors in the aspect advice have performance parameters such as processing demands, branching probabilities, optional probabilities, and loop repetition counts. These may be different at every location **JS** where a context-specific scenario is composed. In the context-specific scenario SpecificA$_{ij}$(parms) they are specified as the list parms of dummy variables whose values are determined by the insertion context. The scoping of names for these dummy variables is defined to be the set of sub-scenarios for the aspect.

The second composition operation also defines values for these dummy variables by additional rules of the form

variable = (expression), where expression may depend on parameters of steps related to join point steps in **JS**

The performance parameters further specialize each SpecificA$_{ij}$ scenario by providing location-specific values for parms.

Table 1 gives the values for some of the performance annotations used in the SSLcall and SSLreply scenarios. The step service demands are given 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 variable $MSG_SIZE is a CSM parameter taking a different value at each join point.

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

Table 1. Performance values for SSLcall and SSLreply context-specific scenarios

Variable	Expression
sslSend.hostDemand	0.1
message.hostDemand	0.1
msgComplete.hostDemand	0.1
sslMessage.hostDemand	0.1
LOOP_Fragments.repetitions	ceiling($MSG_SIZE / 512)

The communication between the WebServer and PGE (the payment server) that is part of the Checkout UML sequence diagram should be secure, as well. However, this communication is only indirectly referenced as an external service, with no details, so it cannot be modified by an aspect. We assume instead that the latency of the external operation that accesses PGE includes the overhead for SSL transfer, and we have to ignore the overhead introduced at the WebServer. This is a limitation for all external operations of a specification. A more complete approach would be to specify the external operation in a separate sub-model and compose it here.

6.6 Composition

Composition is carried out by inserting a composite step for the scenario, and reconciling the resource acquisition and release. If a bound resource is held before the insertion, and is acquired again (without being released first) by the scenario, then the second acquisition step is deleted. Similarly if a resource is released by the scenario and then released again by the primary scenario, the first release step is deleted.

For the example, Figure 12 shows the CSM composed model for GetBuyConfirmPage with SSL data transfer only applied between EB and WebServer. For simplicity the handshake has not been composed along with SSLcall. A composite step SSLcall has been inserted for the message from eb to webserver, refined by the SSLcall scenario, and a composite step SSLreply refined by the SSLreply scenario.

As part of the resource context reconciliation during weaving, the SSLcall scenario 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 scenario completes. Similarly, the SSLreply scenario loses the first ResourceAcquire: webserver and the last Resource Release: eb.

The tool CSMAspects that implements the aspect composition verifies syntactically the binding and composition rules and makes sure that the composed models do conform to the CSM metamodel. Also, the designers have the possibility to inspect the composed CSM models both graphically (with the CSM Viewer) or textually (in XML format), in order to check whether their intentions are correctly realized.

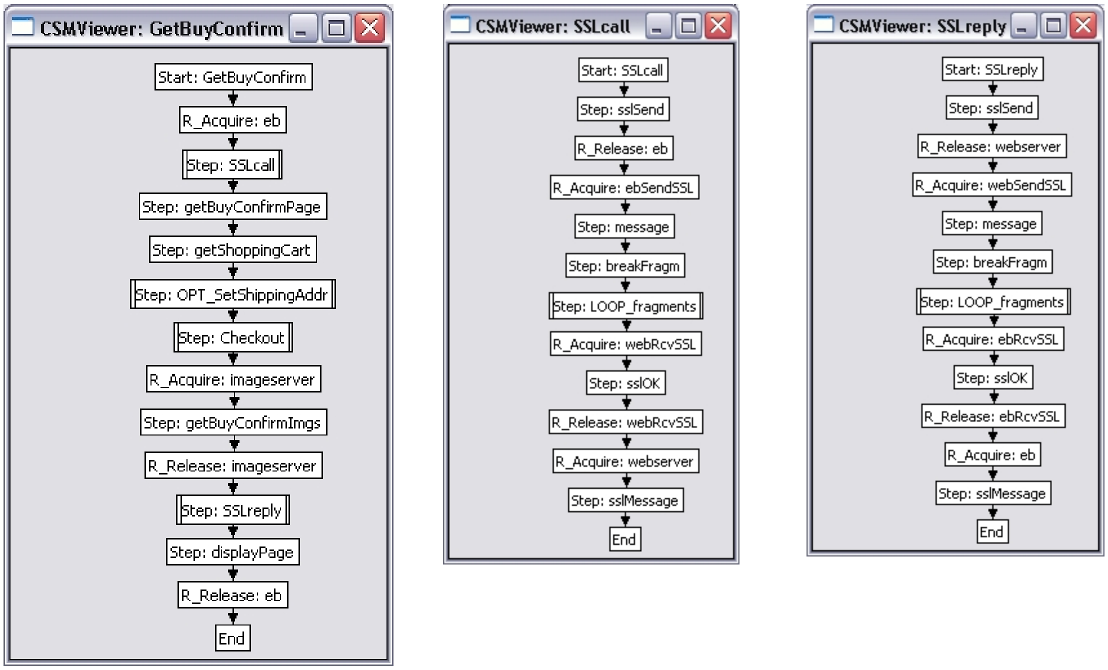

Figure 12. Composed CSM model (sub-scenarios not shown)

The composition was performed in both primary scenarios in two different ways: for both `SSLCombined` and `SSLTransfer`, and for `SSLTransfer` alone. The performance analysis will be reported first for `SSLTransfer` alone, because the transfer overheads are interesting in their own right.

7. LQN PERFORMANCE ANALYSIS

The CSM primary and composed models are automatically transformed into LQN models using the Csm2Lqn generator, an Eclipse-based tool that implements the transformation algorithm described in [21].

7.1 LQN Primary and Composed Models

Figure 13 shows the LQN models for the TPC–W `GetBuyConfirmPage` example: the primary model in Figure 13(a) and the LQN model after weaving the SSL data transfer aspect. 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 (a sequence of asynchronous messages leading to a reply to the original caller, which is blocked). 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`.

The message size parameter $MSG_SIZE was different for each insertion of the aspect scenarios, taking the values shown in the primary models.

7.2 Performance Results

The LQN performance model can be solved by either the analytical solver LQNS or simulation solver LQSim [7],[34]. 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 14 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.

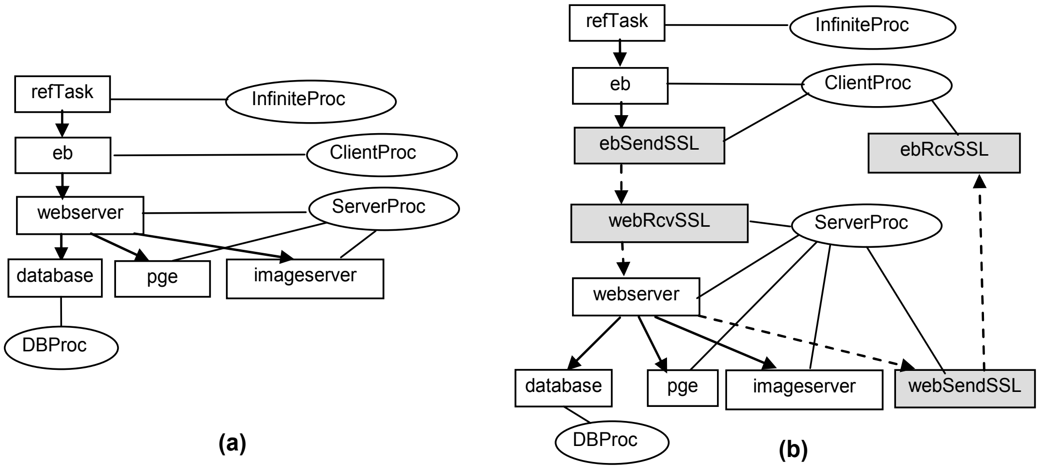

Figure 13. LQN model for GetBuyConfirmPage: a) primary model; b) composed model (`SSLTransfer` case)

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 14(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 14(b), both the primary and composed model enter saturation (where the curve begins to climb) at a rather small 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. 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.

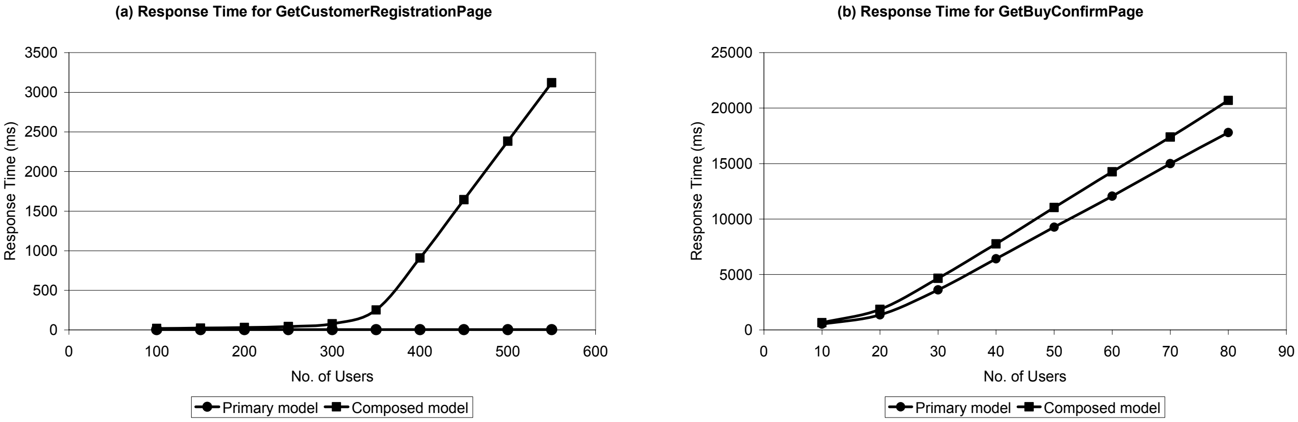

Figure 14. Performance results for: a) GetCustRegPage, and b) GetBuyConfirmPage

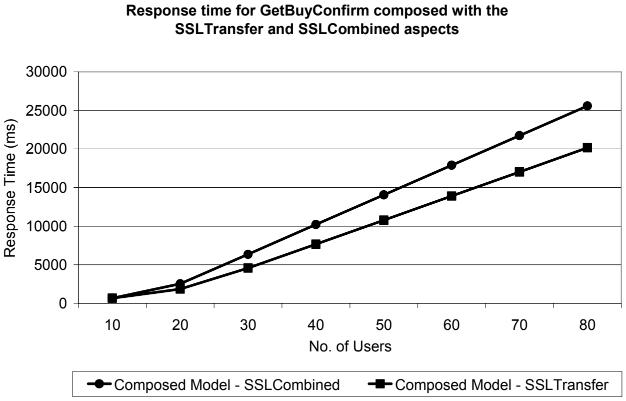

Figure 15. Performance results for GetBuyConfirm composed with the SSLTransfer and SSLCombined aspects

When the `SSLCombined` aspect including the handshake is also woven, the results are similar, although performance is again reduced. Figure 15 shows the LQN results for `GetBuyConfirmPage` composed with the `SSLCombined` aspect compared to the model composed with the `SSLTransfer` aspect. The introduction of the handshake in the `SSLCombined` aspect leads to a 30% increase in response time when using a probability of handshaking of 1 on the page request call and 0 on the reply.

8. CONCLUSIONS

This paper proposes a new approach to aspect oriented modeling which composes aspects with a primary model at a scenario level, using the Core Scenario Model (CSM). The CSM was developed for deriving performance models from UML specifications, and the composed model is used here for performance analysis of the composed aspects.

For SSL communication in the TPC-W example considered here, the aspect had a more dramatic impact on the registration scenario, because the primary model has a light workload, but it had a significant effect on the scalability of both. The consideration of data transfer encryption alone demonstrates the performance impact of the SSL data transfer overhead independent of the SSL session length. The combined case including handshaking shows the additional performance impact of establishing a new SSL session every time a user buys something.

The paper has built a formal compositional framework on CSM, supporting operations on CSM elements and sub-scenarios. The formal properties of this framework have however not been fully explored here (and there has also been comparatively little work on formal foundations for composition within the AOM community in general). Instead, it has been demonstrated in composing a frequently-used feature for secure communication over the Internet. The demonstration shows the simplicity and generality of the specification of localities for aspect composition, by this approach. It is simpler because the CSM metamodel is much simpler than that of UML, yet expresses everything relevant to performance.

The contribution of CSM composition, apart from simplicity, is that it can combine primary models and aspect advice models given in different UML diagram formalisms (activity, sequence, communication, interaction overview), since the CSM can be derived from any of them. Existing tools were used for the CSM extraction.

For performance modeling, not only is the behaviour composed, but the resource use and resource use parameters, provided by performance annotations, must also be composed. This process, and the directives necessary to control it, have been defined and demonstrated.

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 different properties - in this case, security and performance. The interaction diagram in Figure 8(d) 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 [12]. However, for performance analysis a coarser granularity level might be more appropriate. Thus, the CSM model for SSL data transfer given in Figure 10 was generated by aggregating the small sequential steps corresponding to self messages into larger steps (for instance, the UML steps corresponding to the `|receiverSSL` messages `incrementSeqNum()` and `decrypt(payload, symmkey)` have been aggregated in the CSM step `decrypt`).

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 the danger is that they may 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 [23]. For instance, in the case of performance analysis, the aggregation of different steps can be done automatically, under the user's guidance who may decide to provide performance annotations only for coarser-granularity steps.

This paper is a step toward the larger goal of integrating security solution tradeoff analysis and performance analysis. Such an integration will include (1) modeling security features separately as aspects, (2) verifying security properties, (3) analyzing security-related attributes (e.g., Return on Security Investment), (4) analyzing performance properties and (5) balancing conflicting goals, all from the same UML model. Other parts of this goal have already been addressed by some authors of this paper. For instance, a UML-based approach for verifying whether a design meets security properties was developed in [12],[13], including proof-based determination if the system model is secure. Another example is Aspect-Oriented Risk Driven Development (AORDD) [9],[10], for designing cost-effective systems with a desired level of security. In this larger context we must consider that CSM weaving is particular to performance analysis. Other analyses will either have to be adapted to scenario weaving, or will require separate weaving at the UML level.

This paper has not fully developed the formal model of CSM; for instance, it has not examined questions like equality or bisimulation of scenarios, or the determination of properties such as deadlock and termination. The presentation has been brief and somewhat informal, limited to demonstrating the utility of the formal model for composition of aspects. It will require further work to complete it as a formal tool.

REFERENCES

[1] Andrews , J.H., "Process-Algebraic Foundations of Aspect-Oriented Programming", REFLECTION 2001, (A. Yonezawa and S. Matsuoka, Eds.), LNCS 2192, pp.187–209, Springer-Verlag, 2001.

[2] Apostolopoulos, V. and Peris, V. and D. Saha, D. "Transport layer security: How much does it really cost?", Conference on Computer Communications (IEEE Infocom), pp. 717-725, New York, March 1999.

[3] 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.

[4] 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.

[5] Dai, L. and Cooper, K., "Modeling and performance analysis for security aspects", Science of Computer Programming, v 61 pp 58-71, 2006.

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

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

[8] 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.

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

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

[11] IBM, Rational Software Architect (RSA), http://www.ibm.com/developerworks/rational/products/rsa/ (last visited November 2007)

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

[13] Jürjens, J., "Sound Methods and Effective Tools for Model-based Security Engineering with UML", 27th International Conference on Software Engineering (ICSE 2005), St. Louis, Missouri, USA, pp. 322-331, 2005.

[14] Klein, J., Fleurey, F., Jézéquel, J-M., "Weaving Multiple Aspects in Sequence Diagrams", accepted for publication in Transactions on Aspect-Oriented Software Development, 2007.

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

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

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

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

[19] Petriu, D.B. and Woodside, C.M., "A Metamodel for Generating Performance Models from UML Designs", Proc. UML 2004 - Modelling Languages and Applications, 7th Int. Conference, Lisbon, Portugal, LNCS 3273, pp. 41-53, Springer 2004.

[20] D. B. Petriu, M. Woodside, " An intermediate metamodel with scenarios and resources for generating performance models from UML designs", Software and Systems Modeling, Vol.6, Nb. 2, pp. 163-184, 2007.

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

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

[23] Petriu, D.C., Sabetta, A. "From UML to Performance Analysis Models by Abstraction-raising Transformation", In From MDD Concepts to Experiments and Illustrations, (eds. J.P. Babau, J. Champeau, S. Gerard), ISTE Ltd., pp.53-70, 2006.

[24] Petriu, D.C., Woodside, C.M., Petriu, D.B., Xu, J., Israr, T., Georg, G., France, R.B., Bieman, J.M., Houmb, S.H., Jürjens, J., "Performance Analysis of Security Aspects in UML Models", Proc. 6th Int. Workshop on Software and Performance WOSP'2007, Buenos Aires, Argentina, 2007.

[25] Petriu, D.C., H. Shen, H., Sabetta, A., "Performance Analysis of Aspect-Oriented UML Models", *Software and Systems Modeling* (in press), published online April 2007.

[26] 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", A. Rashid, and M. Aksit (eds*). Transactions on Aspect-Oriented Software Development I*, LNCS 3880, pp 75-105, Springer, 2006.

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

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

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

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

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

[32] Woodside, C.M, Petriu, D.C., Petriu, D.B., Israr, T., Merseguer, J., “Methods and Tools for Performance by Unified Model Analysis (PUMA)”, 35 pages, submitted to *IEEE Transactions on Software Engineering*, December 2006 (under review).

[33] Woodside, C.M., "Software Resource Architecture", Int. Journal on Software Engineering and Knowledge Engineering (IJSEKE), vol. 11, no. 4 pp. 407-429, 2001.

[34] −, LQN Online Documentation, www.sce.carleton.ca/rads/lqn/lqn-documentation.

Murray Woodside does research in all aspects of performance and dependability of software. He has taught at Carleton University in Ottawa and done research in stochastic control, optimization, queuing theory, performance modelling of communications and computer systems, and software performance. Since retirement he holds an appointment as Distinguished Research Professor. He is an Associate Editor of Performance Evaluation, a Fellow of IEEE, and a former Chair of Sigmetrics, the ACM Special Interest Group on performance.

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

Dorin B. Petriu is close to completing his Ph.D. studies in the Department of Systems and Computer Engineering at Carleton University. His thesis is on automatic generation of performance models from software design models. His software engineering research interests are in the specification of system requirements and designs and in the early analysis of system behaviour and architecture. Dorin is Vice-President Finance of the Carleton University Graduate Students Association.

Jing Xu is a Ph.D. student in System and Computer Engineering at Carleton University. She received her Master degree in Computer Science at Beijing University of Posts and Telecommunications in 2001. Her main research interests are software performance diagnosis and design improvement, performance analysis of J2EE/EJB, software architectures and design patterns. She currently focuses on rule-based automatic performance diagnosis and improvement.

Tauseef Israr is an IT Architect at IBM. He is interested in software quality of service, especially in the performance analysis of software and business models. He received his Masters degree in the area of software engineering from Carleton University, which formed the basis of most of his research interests.

Robert France is a Full Professor in the Department of Computer Science at Colorado State University. His research interests are in the area of Software Engineering, in particular formal specification techniques, software modeling techniques, design patterns, and domain-specific modeling languages. He is an editor-in-chief of the Springer journal on Software and System Modeling (SoSyM), and is a past Steering Committee Chair of the MoDELS/UML conference series. He was also a member of the revision task forces for the UML 1.x standards.

Geri Georg is a Research Associate in Computer Science at Colorado State University. Her research interests include modeling and analysis of cross-cutting system properties in complex distributed systems. She is also interested in aspect-oriented modeling and model visualization of these properties. Dr. Georg received a Bachelor of Engineering degree in Biomedical Engineering from the University of New Mexico, Albuquerque, New Mexico, and MS and PhD degrees in Computer Science from Colorado State University, Fort Collins, Colorado.

James M. Bieman is a Professor of Computer Science and Director of the Software Assurance Laboratory at Colorado State University. His research focus is on the evaluation of software designs and processes; he studies the relationship between internal design attributes, such as coupling, cohesion, architectural contexts, and the use of design patterns, and external attributes such as maintainability, reliability, and testability. Prof. Bieman is the Editor-in-Chief of the Software Quality Journal published by Springer. He has published more than 90 refereed and invited articles in journals, books, and conferences.

Siv Hilde Houmb holds a post-doctoral position in the department of Information Systems at the University of Twente, The Netherlands. She received her Ph.D. from the University of Science and Technology, Trondheim, Norway. Her research interests are in decision support methodologies and techniques that allow architects to choose among sets of security solutions in security critical information systems. Chosen solutions must be able to balance contracted security levels, available resources, and end-user expectations while still fulfilling development, project, and financial constraints.

Jan Jürjens is a Senior Lecturer at Open University (UK) and coordinator of the Secure Software Engineering research group. He received his Ph.D. from Oxford University. He is the author of "Secure Systems Development with UML" (Springer, 2004) and of various publications, mostly on computer security and software engineering. Dr. Jürjens is the founding chair of the working group on Formal Methods and Software Engineering for Safety and Security within the German Society for Informatics (GI), and a member of the IFIP Working Group 1.7 "Theoretical Foundations of Security Analysis and Design".