Performance Analysis of UML Models Using Aspect-Oriented Modeling Techniques

Hui Shen and Dorina C. Petriu

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

Abstract. Aspect-Oriented Modeling (AOM) techniques allow software designers to isolate and address separately solutions for crosscutting concerns (such as security, reliability, new functional features, etc.) This paper proposes an approach for analyzing the performance effects of a given aspect on the overall system performance, after the composition of the aspect model with the primary model of a system. Performance analysis of UML models is enabled by the "UML Performance Profile for Schedulability, Performance and Time" (SPT) standardized by OMG, which defines a set of quantitative performance annotations to be added to a UML model. The first step of the proposed approach is to add performance annotations to both the primary model and to the aspect model(s). An aspect model is generic at first, and therefore its performance annotations must be parameterized. A generic model will be converted into a context-specific aspect model with concrete values assigned to its performance annotations. The latter is composed with the primary model, generating a complete annotated UML model. By using existing techniques, the complete model is transformed automatically into a Layered Queueing Network (LQN) performance model, which can be analyzed with existing solvers. The proposed approach is illustrated with a case study system, whose primary model is enhanced with some security features by using AOM. The LQN model of the primary system was validated against measurements in previous work. The performance effects of the security aspect under consideration are analyzed in two design alternatives by using the LQN model of the composed system.

1. Introduction

Aspect-Oriented Modeling (AOM) techniques allow software designers to conceptualize, describe and communicate separately 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 [3]. In order to build the complete solution for a system, different aspect models will be composed with the primary system model. Current AOM research is addressing the following problems: using aspects to describe crosscutting concern solutions [3, 13]; describing aspect models at different levels of abstractions (e.g., generic and mechanism specific) [5]; composition of as-

pect and primary models [3, 17, 2]; automation of the AOM approach [8]; analysis of composed models to identify and resolve conflicts and undesirable properties that may arise as a result of the composition [5, 3].

According to [7], there are two broad categories of concerns: a *concrete concern* can be directly realized by some model elements that specifically address it (e.g., security), whereas a *qualitative concern* is based on intrinsic qualities of a system (e.g., performance). This paper proposes an approach for analyzing the system-level performance effects of a concrete concern realized as an aspect model, after its composition with the primary model. In other words, it becomes possible to analyze the combined effects of any concrete concern with a specific qualitative concern (i.e., performance). In order to avoid confusion, the term "aspect model" will be used in the rest of the paper for the concrete concern only.

Over the years, many modeling formalisms, methods and tools have been developed for performance analysis. The challenge is not to reinvent new analysis methods for UML models, but to bridge the gap between UML-based software development tools and different existing performance analysis tools.

Software Performance Engineering (SPE) is a methodology introduced in [16] that promotes the integration of performance analysis into the software development process from the early stages and continuing throughout the whole software life cycle. The "UML Performance Profile for Schedulability, Performance and Time" (SPT) standardized by OMG enables the application of the SPE methodology to systems developed with UML [14]. The SPT Profile defines a set of quantitative performance annotations (such as resource demands made by different software execution steps and visit ratios) to be added to a given UML model. An annotated UML model can be transformed into a performance model and analyzed with known analysis techniques and tools. Since the introduction of SPE, there has been a significant effort to integrate performance analysis into the software development process by using different performance modeling paradigms: queueing networks, Petri nets, stochastic process algebras, simulation, etc. [1]. The performance modeling formalism used in this paper is the Layered Queueing Model (LQN) [18]. The transformation from UML to LQN used in this paper was developed in previous research for systems designed without AOM [10, 6, 9, 12, 19].

The paper is organized as follows: section 2 presents the overview of the proposed approach; section 3 describes how performance annotations are added to aspect and primary models and how are handled during the composition, which is approached as a graph rewriting problem; and section 4 analyzes the performance effects of a concrete aspect under consideration and discusses different design alternatives. The case study used throughout the paper is an existing application, named the Document Exchange Server (DES) that was implemented and measured in previous work [12]. DES is enhanced in this paper with some security features by using AOM. The approach for defining generic and context-specific aspect models and for combining the aspect with the primary model is inspired from the work of France et al. [3, 5]. The original contribution of this paper is two-fold: a) adding performance analysis to UML models developed with AOM, and b) approaching the composition of the behavioural representation from the aspect and primary models as a graph rewriting problem applied to activity diagrams with composite activities.

2. Overview of the Proposed Approach

The long-term goal of the research presented in this paper is to provide tool support to software developers who are using AOM techniques for assessing the performance effects of different aspect realizations early in the development cycle. This paper is just the first step on the road toward such a goal. Fig. 1 illustrates the high-level view of the proposed approach.

A primary model and one or more generic aspect models with performance annotations, produced with an UML tool, are exported to XMI. The first phase is to instantiate the generic aspect model, producing a context-specific one as in [3, 5], by following a set of binding rules provided by the designer. The binding rules are augmented with instructions on how to transform the parametric annotations of the generic aspect model into concrete ones. The next step is to compose the context-specific aspect model(s) with the primary model, according to a set of composition directives. The result is a composed annotated UML, which can be transformed automatically into a performance model (LQN in this case) by using the transformation techniques from [19]. The LQN model is analyzed with an existing solver for different workloads and conditions, and the analysis results are used to draw conclusions about different design alternatives. The process will be eventually completed with a feedback path, shown with dotted arrows, whereby the performance results are inserted into predefined annotation placeholders in the XMI file of the composed model, which will be imported back in the UML tool for display. The composed model can be also imported directly into the UML tool for display without performance results.

The focus of this paper is on the instantiation and composition steps, and especially on the treatment of performance annotations. The paper also illustrates the application of the proposed approach to enhance an existing application, the Document Exchange Server, with some security features (namely authorization). The LQN model of the primary system was previously validated against measurements in [12]. The performance effects of the aspect under consideration are analyzed in the paper by solving the LQN model of the composed system for two design alternatives.

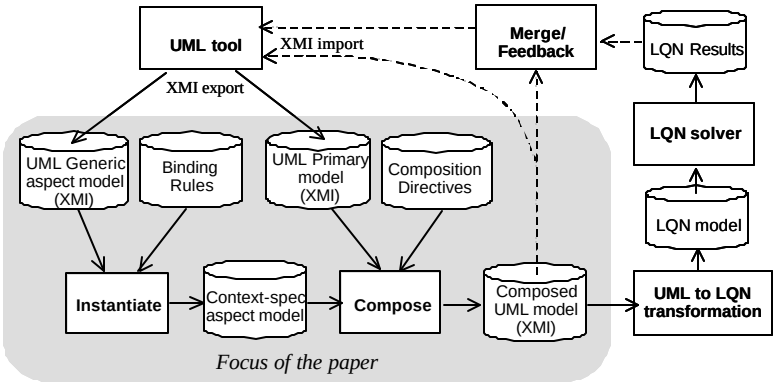


Fig. 1. Approach for performance analysis of UML models using AOM

3. Aspect Oriented Models with Performance Annotations

The SPT Profile [14] contains the Performance Subprofile that identifies the main basic abstractions used in performance analysis. Scenarios define response paths through the system, and can have QoS requirements such as response times or throughput. Each scenario is executed by a workload, which can be closed or open, and has the usual characteristics (number of clients or arrival rate, etc.) Scenarios are composed of scenario steps that can be joined in sequence, loops, branches, fork/joins, etc. A step may be an elementary operation at the lowest level of granularity, or may be a complex sub-scenario. Each step has a mean number of repetitions, a host execution demand, other demand to resources and its own QoS characteristics. Resources are another basic abstraction, and can be active or passive, each with their own attributes. A more detailed description of the way to apply the Performance Subprofile is given in [11]. Please note that SPT was standardized for UML 1.4; until SPT will be upgraded for UML 2, we apply its stereotypes to UML 2.

3.1. Primary Model

The primary UML model contains different views necessary for performance evaluation [10]:

– High-level software architecture represented by one or more class or components diagrams showing the concurrent (distributed) component instances (Fig. 2).

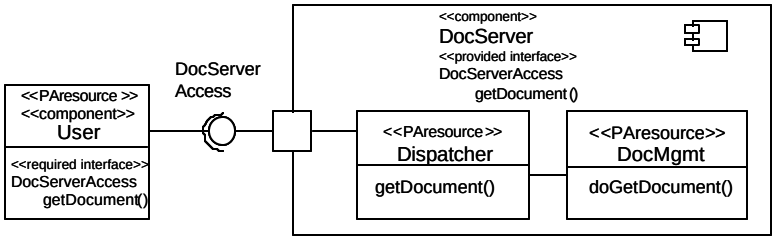


Fig. 2. DES primary model: component diagram with performance annotations

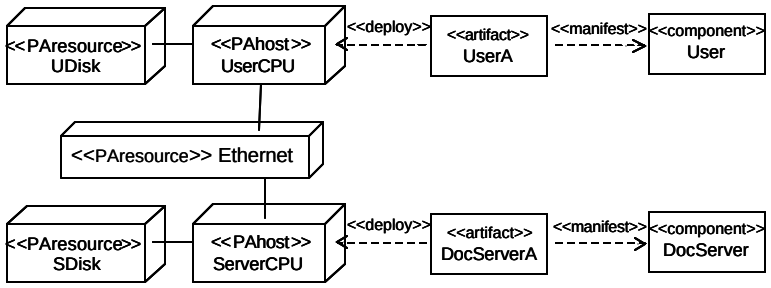
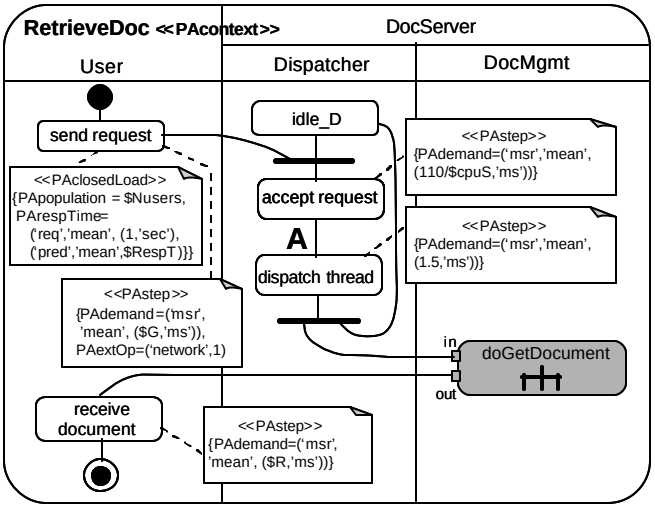
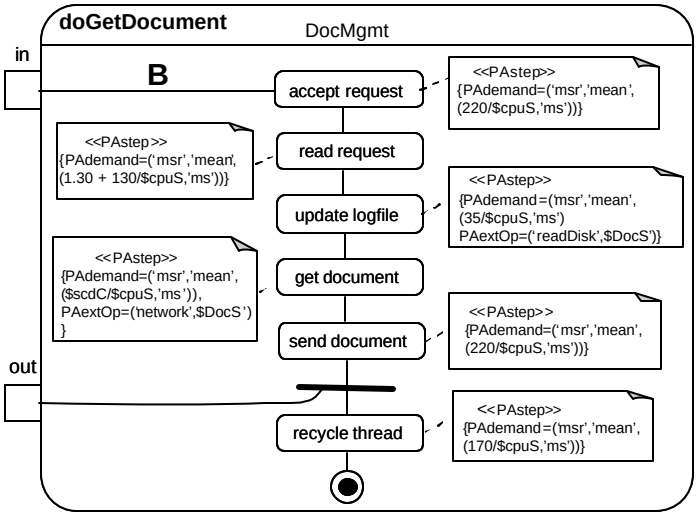


Fig. 3. DES primary model: deployment diagram with performance annotations

– Deployment of high-level software components to hardware devices (Fig. 3).
– One or more key performance scenarios annotated with performance information according to the SPT Profile [14], modeled by interaction or activity diagrams [11]. In the paper we consider the scenario modeled by the activity diagrams in Fig. 4.



(a) DES primary model: high level activity diagram for `RetrieveDoc` scenario



(b) DES primary model: nested activity diagram `doGetDocument`

Fig. 4. DES primary model: scenario `RetrieveDoc` with performance annotations

The DES system was previously implemented with the ACE reusable frameworks [15], and its LQN model was validated against measurements [12]. DES consists of a document exchange server and multiple clients. There are two types of users: regular users and system administrator. A regular user can get the document directory from the server, upload new documents and retrieve documents stored at the server. In this case study, we will focus on the scenario for retrieving a document, `RetrieveDoc`, as the key scenario for performance analysis. The UML model shown here does not represent the complete design of the DES system , just the elements necessary for performance analysis. The performance annotations are shown in notes in Figures 2 to 4 to make them visible. Due to space limitations, we give here just a brief overview of the most important stereotypes and attributes.

The high-level architecture contains two components, `User` and `DocServer` communicating through the interface `DocServerAccess`, which contains the operation `getDocument()`. The server component is multithreaded, containing a `Dispatcher` thread that accepts the requests and dispatches them to a number of worker threads named `DocMgmt`, organized in a thread pool. In Fig. 2, the stereotype `<<PAresource>>` is used to indicate those software units that are running under their own thread of control (in this case, the user component and each server thread). DES is deployed on a distributed system connected through a local area network, as shown in Fig. 3. The shared documents are stored on the server's local disk. A processor is modeled by the stereotype `<<PAhost>>`, which has attributes that define its scheduling policy, processing rate, context switching time and performance measures such as utilization and throughput. Other non-processing hardware devices that introduce contention in the software may be modeled as `<<PAresource>>`.

The activity diagram for the `RetrieveDoc` scenario (Fig. 4) shows that the `Dispatcher` loops infinitely, going back to accepting a new request once the previous one was dispatched. The detailed processing of the operation `doGetDocument()`performed by a `DocMgmt` thread is encapsulated in the composite activity given in Fig.4.b, which has an input pin `in` and an output pin `out`. The call of this operation is shown in gray in the main activity diagram from Fig.4.a. The labels A and B are used to mark possible insertion points for the aspect behaviour during model composition, as discussed later in section 3.4.

The main activity diagram is stereotyped as an SPT analysis context `<<PAcontext>>`, and each activity as a scenario step `<<PAstep>>`. The first step carries the workload stereotype `<<PAclosedLoad>>` with a given number of users `$Nusers`, and the scenario overall performance measures, which can be required (`'req'`), measured (`'msr'`), estimated (`'assm'`) or predicted (`'pred'`). (Note that in SPT variable names begin with `'$'`.) For example, the scenario from Fig. 4 has a required mean response time of `1` second for the specified number of users `$Nusers`, and the response time predicted by the LQN model will be stored in the variable `$RespT`, as indicated in the following tagged value:

```
PArespTime= (('req',mean,(1,'sec')),('pred',mean,$RespT))
```

A `PAdemand` tagged value indicates the execution time on the host processor for the respective step. For instance, the activity `accept request` from Fig. 4.b has:

```
PAdemand=('msr','mean,(220/$cpuS,'ms'))
```

which indicates that the mean measured value of the CPU demand is given by the expression (220/$cpuS) in milliseconds, where the variable $cpuS is the frequency of the host processor in MHz. The variables used in performance annotations capture application or platform-specific performance values. The following variables used in the example are dependent on the disk I/O mechanism and the document size:

$gcdC = CPU demand for getting a document from the disk

$scdC = CPU demand for sending a document to the network

The following variables are application dependent:

$RP = the size of a request message in data packets

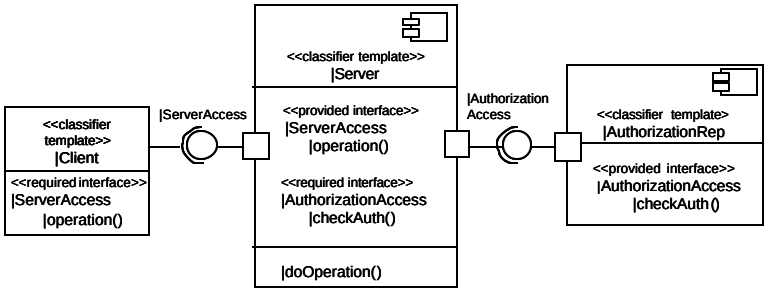
$DocP = document size in data packets (given by the ratio between the document sizeand network packet size rounded up to the closest integer).

3.2. Generic Aspect Model

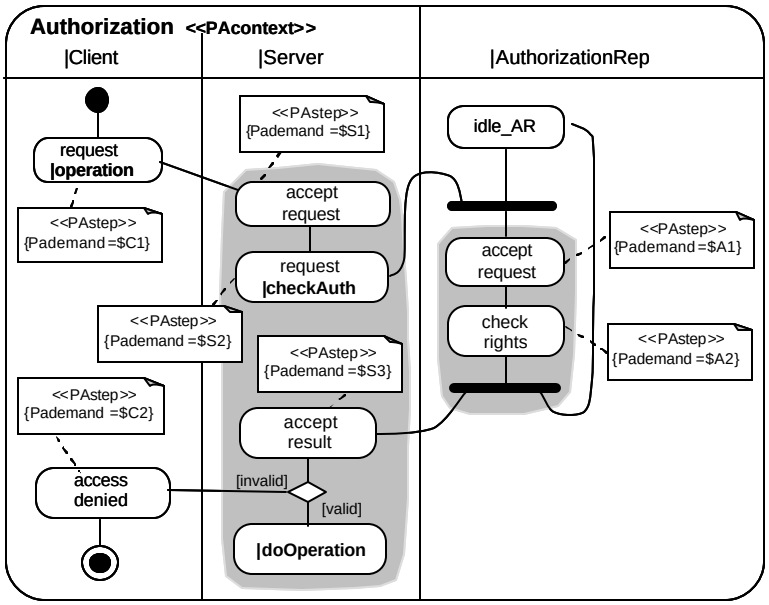
AOM is applied in this paper to extend the original DES system with a security related crosscutting concern, whereby only authorized clients are allowed to get documents from the DES server. The approach for expressing the solution to this concern as an aspect was inspired from [3], where a *generic aspect model* that describes the general structure and behaviour of a generic authorization solution is defined with UML 2 templates. More precisely, in [3] the generic aspect structure is modeled with classifier templates (classes or structured classes) and the behaviour with interaction templates. The generic aspect model is instantiated to get a *context-specific aspect model* by binding the template parameters to application-specific values.

Our approach is similar to [3], except that the software architecture is modeled with component templates (with offered and required interfaces), and the behaviour with activity templates, as shown in Fig.5. We are making use of the UML 2 feature that all subclasses of Classifier - such as Class, Collaboration, Component, Interface - and all subclasses of Behavior - such as Activity, Interaction - are templateable. Components and/or structured classes allow for a clear separation between their external use and their internal structure/behaviour, and are more suitable than the traditional class diagrams for representing the kind of systems for which performance analysis is important (usually distributed and/or concurrent systems). We chose activity diagrams rather than interaction diagrams because of their ability to describe both inter- and intra-object behaviour as flows of actions, and to express concurrency more naturally, like in Petri nets. We propose to approach the behaviour composition as a graph rewriting problem applied to activity diagrams, as described in section 3.4.

Fig. 5.a shows that there are three kinds of components in the generic aspect model: $Client, |Server and |AuthorizationRep. (Note that we use the same notation for template parameters as in [3], i.e., a parameter name begins with a '|'). The |Server component provides the interface |ServerAccess containing the operation template |operation, and requires the interface |Authorization-Access containing the operation template |checkAuth. The aspect model does not know anything about the internal structure of $Server, nor any details about the actual functionality of |operation, which are given in the primary model.



(a) Generic Authorization aspect model: component diagram



(b) Generic Authorization aspect model: activity diagram

Fig.5. Generic aspect model with parametric performance annotations

The generic aspect model shows that, when a request for |`operation` arrives from |`Client` to |`Server`, the latter must check with the component |`AuthorizationRep` whether the client is authorized to perform the |`operation`. More exactly, |`Server` invokes |`checkAuth`, waits for the reply, and then verifies the result. If this indicates a not authorized access, then |`Server` will reply to |`Client` that the access is denied; otherwise, it delegates the actual execution of the required functionality to |`doOperation` (which will be detailed only in the primary model). The signatures and parameters of the operation templates are similar to those from [3] and are not described here due to space limitations.

It is worthwhile to mention here that the behaviour model from Fig. 5.b contains two kinds of activities: some represent the new functionality associated with the aspect (such as checking the access rights and letting the client know when the access is denied) and others represent "embedding" activities, which show where to insert the new behaviour relative to the primary model behaviour. In our example, the new authorization functionality must take place every time when the client sends a request to the server, but before the request will be actually served. More on the embedding of the aspect behaviour in the primary model behaviour will be discussed in section 3.4.

It was mentioned that activity diagram can represent both intra- and inter-object behaviour. For instance, an |`operation` request is modeled by a CallOperationAction metaobject with a transition that crosses the swimlane boundary from |`Client` to |`Server`, whereas the execution of |`operation` is represented by the activities in the shaded area from |`Server's` swimlane, starting with the acceptance of the respective call (represented by an AcceptCallAction or AcceptEventAction metaobject in UML 2), and ending with the call of |`doOperation` (represented by an InvocationAction metaobject). Similarly, the execution of operation |`checkAuth` is represented by the shaded area from the swimlane of |`AuthorizationRep`.

In what regards the performance annotations, each activity is stereotyped as an SPT `<<PAstep>>`, whose tagged values provide performance information such as CPU demand, probability of execution, external operations, etc. [14]. The stereotype attributes are not assigned concrete values, but are represented instead by SPT variables that are treated as "performance parameters". These variables will receive concrete values when it becomes known how the context-specific aspect model is instantiated, what primary model is composed with and what kind of platform the final composed model will be run on. The types of these performance parameters are defined in the SPT profile, and some can be rather complex, such as the types for time and performance values, `PAtimeValue` and `PAperfValue` [14]. However, a UML tool treats them as string values assigned to the respective stereotype attributes.

3.3. Context-Specific Aspect Model

The next step is to instantiate the template aspect model for a given application context by binding the template parameters to application-specific values. According to [5], a generic aspect model can be instantiated multiple times to produce multiple context-specific aspect models based on different binding rules. The result is a *context-specific aspect model*. In our approach, the binding rules have two parts: one for the "traditional" AOM approach, and the other for performance annotations.

In terms of "traditional" AOM binding, our approach is almost the same as in [3], except for the fact that our templates refer to components and activities. The binding rules for operation signatures, similar to [3], and are not shown here due to space limitations. The bindings for structural elements used in our case study, listed here as *(formal parameter, actual parameter)* pairs, would be normally given in a UML diagram:

– Component bindings: `(|Client, User); (|Server, DocServer); (|AuthorizationRep, DocAuthorizationRep)`

– Interface bindings: `(|ServerAccess, DocServerAccess);` `(|AuthorizationAccess, DocAuthorizationAccess);`
– Operation bindings: `(|operation, getDocument); (|doOperation, doGetDocument); (|checkAuth, checkDocAuth).`

In the structural view, some of the component (operation) templates are bound to actual counterparts that exist in the application context, while others are bound to new components/interfaces/operations (shown in boldface in the above list). For instance, `DocAuthorizationRep`, along to its interfaces and operations, is a new component that does not have a counterpart in the primary model of the application.

There is another issue concerning the model structure that has to be resolved during the creation of the context-specific model: the allocation of the software components to hardware resources. This is important for performance analysis. The rule is as follows: if a component template is bound to an existing component, then the host processor is already known from the primary model. However, for new components, the designer has to specify the deployment explicitly (either on existing nodes or on new ones).

The instantiation of the template activity diagram has two parts: one is concerned with binding activity templates from the generic to the application-specific context, and the other with assigning concrete values to the "performance parameters" identified in the previous section as part of the performance annotations. The binding of activity templates will be done according to the binding of the corresponding operation templates from the structural view. For instance, the activity template that requests `|operation` will be bound to an activity that requests `getDocument`, and so on. The activity diagram of the context-specific model is represented in Fig. 6.a.

The issue of binding "performance parameters" cannot be solved through the UML 2 template mechanism, because it requires the "binding" of new values to stereotype attributes. For instance, the variable `$A1` from Fig. 5.b, which represents the tagged value `PAdemand` of the step `accept_request`, should be assigned the value

`('assm','mean,(220/$cpuS,'ms'))`

of type `PAperfValue` defined in SPT. We propose to use an auxiliary XML file for "performance bindings" which gives all the values to be assigned to the performance parameters representing stereotype attributes in the generic aspect model.

Choosing the values to be assigned to the performance parameters of the context-specific aspect model is not a simple problem; some difficulties are related to performance evaluation issues rather than to UML modeling. In general, it is difficult to estimate quantitative resource demands for each activity in the design phase, when an implementation does not exist and measurements cannot be performed yet. Several approaches are used by the performance analysts to come up with reasonable estimates in the early design stages: expert experience with previous versions or with similar software, understanding of the algorithm complexity, measurements of reused software, measurements of existing libraries, or using time budgets. As the project advances, early estimate can be replaced with measured values for the most critical parts. However, this is not to say that performance analysis should be deferred until late in the lifecycle, when the system is implemented and can be measured, because by then it may be too late to correct costly performance mistakes frozen in the code (see [16] for more details on software performance engineering).

3.4. Model Composition

The role of model composition is to integrate a context-specific aspect model with the primary model in all three relevant views: architecture, deployment and behaviour.

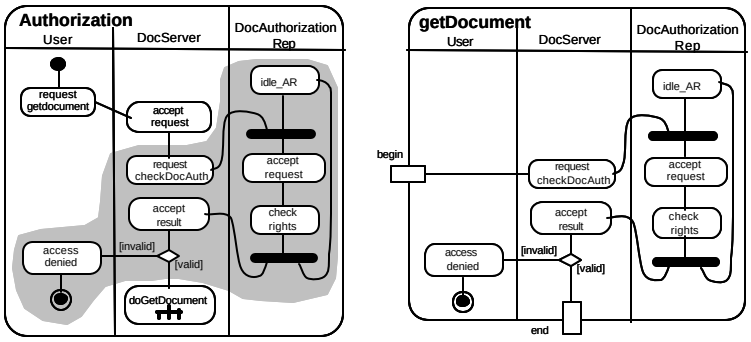
Composing the software architecture is not as difficult as composing the behaviour. The context-specific aspect model contains either components that exist in the primary model or new ones, with well-defined interfaces. The composed model will contain the union of all the components from the two models. It is however possible that a component in the aspect model does not contain the level of details from the primary model (for example, `DocServer` is multithreaded in the primary model only). A recursive approach, similar to that at the system level, will be applied to compose the internal structure of each component in turn. The composition at the deployment level, which is also a structural view, can be tackled in a similar way, adding new nodes to he ones that exist already in the primary model.

The composition of the behaviour view is more challenging. Conceptually, we propose to approach the composition of activity diagrams as a graph-rewriting problem, where a subgraph *X* found inside of a larger host graph *H* is isolated and replaced by another subgraph *Y*. Subgraph *X* is described by the left-hand-side and *Y* by the right-hand side of a rewriting rule, which also specifies how to embed (i.e., connect) *Y* within the host graph *H*. In our case, the host graph is the activity diagram of the primary model, *Y* is the subset of activities from the context-specific model that bring new functionality to the whole, and *X* is an element of the host *H* that pinpoints the insertion place. The proposed approach is illustrated in Fig. 6.

Fig. 6.a shows the activity diagram for the behaviour of the context-specific model, which contains two kinds of activities, as already mentioned in section 3.2: a) new functionality introduced by the aspect (the shaded area in Fig. 6.a), and b) "embedding" activities repeated from the primary model that indicate where to insert the new functionality (the non-shaded area). We propose to isolate the activities from the shaded area and to encapsulate them in a UML 2 *complete structured activity* with input and output pins, which corresponds to the connecting points between the new functionality from the shaded area with the embedding activities from the non-shaded area, as shown in Fig. 6.b. In this case, there is only one input and one output pin, but in general more than one input/output pins may be necessary. The designer has the responsibility to indicate which sub-area of the aspect model contains new functionality and should be converted into a complete structured activity, to play the role of *Y* in the rewriting rule.

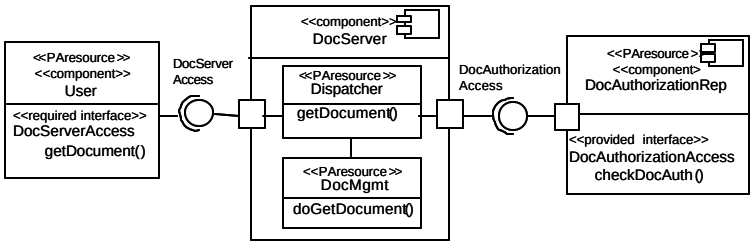
The role of *X* is played by an element from the activity diagram of the primary model that indicates the insertion place. In our case study, we have considered two design alternatives: i) insert the authorization checking functionality in the `Dispatcher` thread in Fig.4.a, point `A`, and ii) insert it in the `DocMgmt` thread in Fig. 4.b, point `B`. As shown in the next section, the choice of the insertion point will have a strong impact on the overall performance without changing in any way the aspect model or its performance annotations.

The outcome of the composition for Design A is illustrated by the component diagram in Fig. 7.a and the activity diagram in Fig. 7.b. The composed deployment diagram is not given, as it is very similar to the deployment of the primary model from Fig. 3.

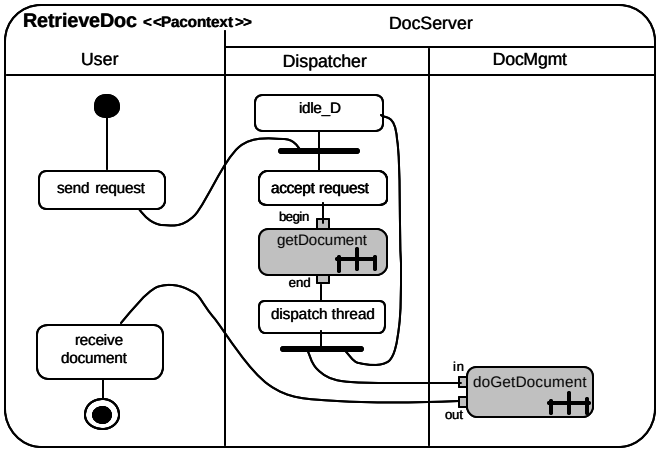


(a) Context-specific aspect model behaviour (b) Complete structured activity with pins

Fig. 6. Generating a complete structured activity with pins that contains the aspect model sub-behaviour to be inserted into the primary model behaviour



(a) Composed DES model: component diagram



(b) Composed DES model: high-level activity diagram

Fig. 7. Composed DES model

4. Performance Analysis

This section presents the performance analysis experiments conducted with the LQN models obtained from: a) the primary model, b) the composed model for design A, and c) the composed model for design B. The LQN models were obtained with the methodology from [19]. The LQN models are not described in the paper due to space limitations. The LQN model of the DES application without authorization was validated against measurements, as described in [12].

In the first set of experiments, we compared the effect of Design A authorization on the response time perceived by a user who is retrieving documents, when the number of identical users is increasing from 1 to 15. The analysis was done for two document sizes: short (5K B) and long (50 KB).

The analysis shows that the effect of the authorization aspect on the response time depends strongly on the document size: there is almost no effect for large documents (see Fig. 8), whereas there is an important effect for small documents (see in Fig. 9). To understand the reason for this performance behaviour, we looked at the utilization of different resources to identify the system bottleneck (i.e., the resource that saturates first, has the longest waiting queue and limits the system throughput). For large documents, the bottleneck device is the Local Area Network, which is utilized close to 100% for 15 users, as shown in Fig. 10. Other resources, such as `ServerCPU` and `SDisk` are utilized much less than the network (only about 55% for 15 users). However, the new authorization functionality adds no extra load on the network, but uses instead `ServerCPU` and `SDisk`, which have enough available capacity. Therefore the response time increases very little because of the additional work introduced by the authorization functionality in the case of long documents.

The situation is different for short documents, where the bottleneck is the `Dispatcher` thread, as shown in Fig. 11. The choice of inserting the authorization responsibility in the `Dispatcher` in Design A serializes considerably the execution of the requests in the system, as a lot of work is done in a single thread. This is an example of so called "software bottleneck", where none of the hardware resources gets to be fully utilized due to the low concurrency levels in the software. In order to solve the software bottleneck, we consider Design B, where the authorization functionality is inserted in each of the `DocMgmt` threads (i.e., the authorization for different request is done in parallel). This insures higher concurrency levels in the system and gives better response time than Design A, as long as `DocAuthorizationRep` is also able to process requests concurrently. Fig. 12 shows that for small messages, the response time of Design B is very close to that of the primary system. This is an illustration of the fact that a small design difference may have a big performance impact.

Fig. 13 shows that in the case of Design B, the hardware resources (such as ServerCPU) are indeed utilized at a higher level than in Design A, whereas the Dispatcher thread is no longer the bottleneck. This explains why Design B has better performance than Design A.

Performance analysis allows developers to gain insight on the location of performance trouble spots under different workload conditions. The goal is to help developers to evaluate and choose better design alternatives as early as possible in the development process.

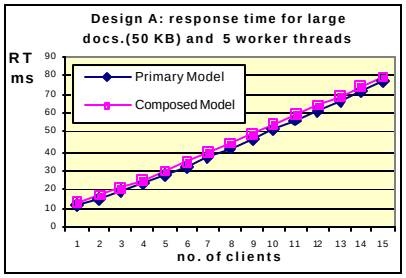


Fig. 8. Design A: Response time for the retrieval of large documents

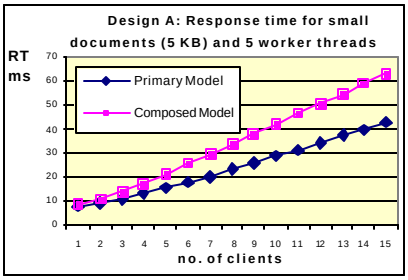


Fig. 9. Design A: Response time for the retrieval of small documents

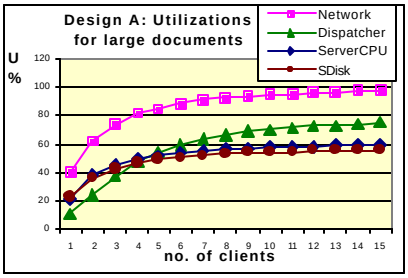


Fig.10. Design A: Utilization of resources for the retrieval of large documents

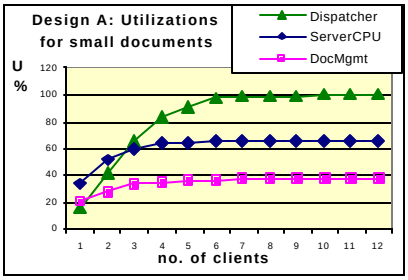


Fig.11. Design A: Utilization of resources for the retrieval of small documents

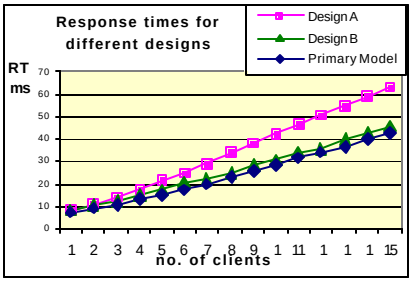


Fig.12. Response times for different designs alternatives

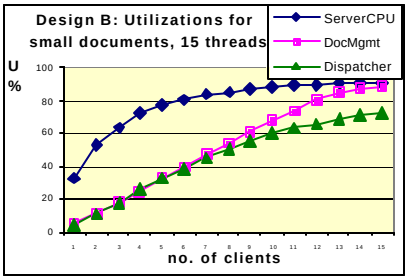


Fig.13. Design B: Utilization of resources for the retrieval of small documents

5. Conclusions

This paper proposes an approach for combining Aspect Oriented Modeling techniques with performance analysis of UML models. The long-term goal of the research is to provide tool support to software developers who are using AOM techniques for assessing the performance effects of different aspect realizations early in the development cycle. There is ongoing work to develop fully the proposed approach and to build a tool prototype.

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. Clarke, S. and Walker, R. J., "Composition patterns: An approach to designing reusable aspects", In Proc. of 23rd Int. Conf. on Software Engineering (ICSE), Toronto, Canada, 2001.
3. R. B. France, R.B., Ray, I. ,. Georg, G. and Ghosh, S., "Aspect-Oriented Approach to Design Modeling," IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4):173--185, August 2004.
4. Franks, G., Hubbard, A., Majumdar, S., Petriu, D.C., Rolia, J., Woodside, C.M., A toolset for Performance Engineering and Software Design of Client-Server Systems, Performance Evaluation, Vol. 24, Nb. 1-2 (1995) 117-135.
5. Georg, G., France, R. and Ray, I. "An Aspect-Based Approach to Modeling Security Concerns".In Proceedings of the Workshop on Critical Systems Development with UML, Dresden, Germany, 2002.
6. Gu, G., and Petriu, D.C. "XSLT Transformation from UML Models to LQN Performance Models", Proc. of 3rd Int. Workshop on Software and Performance WOSP'2002, pp.227-234, Rome, Italy, 2002.
7. Kande, M., "A Concern-Oriented Approach to Software Architecture", PhD thesis, EPFL, Lausanne, Switzerland, 2003.
8. Mekerke, F., Georg, G., France, R., and Alexander, R. "Tool Support for Aspect-Oriented Design", In Advances in Object-Oriented Information Systems: OOIS2002 Workshops. Springer-Verlag, 2002.
9. Petriu, D.B. and Woodside, C.M., "A Metamodel for Generating Performance Models from UML Designs," in In Proc. «UML» 2004 - Modelling Languages and Applications, 7th Int. Conference, Lisbon, Portugal, vol. LNCS 3273, Springer 2004, pp. 41-53.
10. Petriu, D.C. and Shen, H. "Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications", in Computer Performance Evaluation: Modelling Techniques and Tools, (T. Fields, P. Harrison, J. Bradley, U. Harder, Eds.) LNCS 2324, pp.159-177, Springer, 2002.
11. Petriu,D.C. and Woodside, C.M., "Performance Analysis with UML," in *UML for Real*, B. Selic, L. Lavagno, and G. Martin, pp. 221-240 Kluwer, 2003.
12. Petriu,D.C., Zhang, J., Gu, G and Shen, H., "Performance Analysis Based on the UML SPT Profile", to appear in *MDD for Distributed Real-time Embedded Systems* (Eds. J.-P. Babau, J. Champeau and S. Gérard), Hermes, Paris, 2005.
13. Ray, I., France, R., Li, N., Georg, G. An aspect-based approach to modeling access control concerns", *Information and Software Technology*, 46 (2004) 575–587.
14. Object Management Group, *UML Profile for Schedulability, Performance, and Time Specification*, OMG Adopted Specification ptc/02-03-02, July 1, 2002.
15. Schmidt, D.C., Huston, *S. D., C++ Network Programming Vol 2: Systematic Reuse with ACE and Frameworks*, Addison-Wesley, 2002.
16. Smith, C.U., *Performance Engineering of Software Systems*, Addison Wesley, 1990.
17. 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.
18. Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S., "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", in *IEEE Transactions on Computers*, Vol.44, Nb.1, pp. 20-34, 1995.
19. 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, Palma, Spain, July 2005.