Analysing Software Requirements Specifications for Performance

Dorin Petriu, Murray Woodside,
Dept. of Systems and Computer Engineering,
Carleton University, Ottawa, Canada
{dorin|cmw}@sce.carleton.ca

Abstract

The earliest moment when performance issues can be addressed is the initial specification of a software system, during the formulation of the architecture, and well before the design stage. A common form of specification at this stage is a set of scenarios to be executed by the system, which embody the Use Cases, and identify the sequence of responsibilities to be carried out in different kinds of responses. On the basis that earlier analysis is better, a performance modeling capability has been installed in a scenario modeling tool for Use Case Maps that is part of a proposed standard for User Requirements Notation. Using examples, the paper shows how this kind of early analysis can address high-level performance questions, at a comparable level of abstraction to the specification. The imprecision of early knowledge, and the risk of ignoring some performance limitations, are key factors whose impact is addressed.

Keywords: specifications, performance, Use Case Maps, completion, non-functional requirements

1 Introduction

The earliest decisions that can constrain the performance of a software system are made during requirements analysis, when the sequence of operations is developed. This work considers the stage of developing the system Use Cases into scenarios to be executed, with sequences of responsibilities and with data operations. The specification uses Use Case Maps, which are then analyzed for performance. A second opportunity for performance analysis arises a little later, after the architectural design stage.

We can see two approaches have been taken to early performance analysis, for both of these stages. The first approach is a qualitative analysis in terms of possible impacts of system aspects on product qualities such as performance. For example, Chung et al. have developed a general approach for analyzing non-functional requirements that they call NFR [6], which has been applied for performance by their co-author Nixon [11]. A similar qualitative analysis applied to the slightly later stage, of system architectural design, is described by Bass, Clements, and Kazman [2]. The second approach is quantitative, using models. Smith developed an approach to building a queueing model based on scenario-like “execution graphs” [17] [19] that are specially built for performance analysis. Balsamo, Inverardi and Mangano described an architecture modeling technique [1], and a general framework for modeling called “resource architecture” is described in [21].

The authors of this work on evaluation have always acknowledged the difficulties posed by early analysis, some of which are:

- incompleteness of the specification, because it is so abstract. This includes open choices of design approaches, algorithms and components to be used, and lack of definition of the final execution environment
- lack of knowledge of the computational effort required
- other aspects such as ignorance of the workload intensity,

However they have tended to concentrate on showing that an analysis of some kind is feasible, and not to address the difficulties directly. What we intend in this paper is to identify the sources of weakness in the analysis, and to estimate their impact. This will be done in the context of automated model-building from software design products, which may be part of the mainstream process of creating the software. The automated model-building method technique has been described in [12] and [13].

The specification language used in this work is called Use Case Maps, and is intended to be used after identifying Use Cases, and before developing other models of architecture or behaviour (class or collaboration models in UML, for instance). Thus it addresses very early issues in analysis. If one wishes to address the same issues in the UML [3] one would have to create a set of sequence, activity or collaboration models, plus information linking them together.

2 A Use Case Maps Example: an E-Commerce Site

UCM specifications, and the analysis of them, will be presented using a simplified example of an e-commerce site. This is not intended to represent a state-of-the-art design of such a site, but to provide a familiar context in which to show the analysis steps and to explore issues and possibilities.

The example will be called RADSbookstore (selling books on Real-time And Distributed Systems), and it will provide:

- an interface for customers, to browse the catalogue and to buy books,
- an interface for bookstore administrators, to examine the inventory and the sales data,
- databases for the inventory, and the customer accounts,
- applications to manage customer accounts, shopping cart objects, and inventory,
- a subsystem to track and fill back-orders (orders to be filled for books that are yet to arrive)

We may expect this system to fit into the broad category of 3-tier client-server systems (with interface, application, and database tiers).

The requirements specify that the customers are supported by the following set of Use Cases, each of which will be developed as a scenario in the Use Case Map:

- *browse*: a list of items is retrieved from the database
- *view_item*: a detailed description of an item is retrieved from the database
- *add_to_cart*: an item is added to the customer's shopping cart
- *remove_from_cart*: an item is removed from the customer's shopping cart
- *checkout*: the customer buys the items contained in the shopping cart
- *login*: the customer's login is verified in and if valid the customer's account information is retrieved from the database
- *register*: a new customer is registered with the bookstore

The Administrator is supported by two Use Cases/scenarios:

- *update_inventory*: the stock for each item is updated
- *fill_backorders*: outstanding backorders are filled (following an inventory update)

The Use Case Map specification technique that will be used was developed by Buhr and Casselman [4] [5] to facilitate reasoning about the earliest stages of design, and particularly the allocation of functions to components. It falls into the general class of scenario specifications for requirements analysis, and its relationship to other scenario descriptions, particularly to Message Sequence Charts, is described in [10]. An analysis in terms of UCMs is typically followed by further development in UML or SDL.

A Use Case Map defines scenario paths as lines joining responsibilities in a causal sequence, possibly drawn over components. These paths execute responsibilities along the way in order to capture the functionality involved in the execution of a given scenario. Figure 1 shows the top level UCM with customer and administrator components interacting with the RADSbookstore. The UCM paths - denoted by lines starting with filled circles and ending with bars - show the typical interaction scenarios of a customer and an administrator. The customer or administrator makes a request of the bookstore at the x-shaped UCM *responsibilities* labelled 'makeRequest'. The path then crosses over to the RADSbookstore *component* where the generic customer or administrator service operations are represented

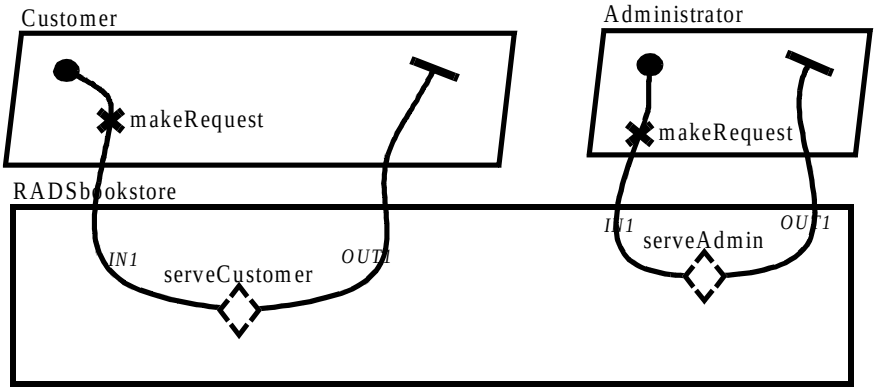

Figure 1. The top level Use Case Map showing the service of Customers and Administrator

by the UCM *stubs* labelled ‘serveCustomer’ and ‘serveAdmin’ respectively. The dashed outlines of the stubs denote that they are *dynamic stubs* which can have multiple *plug-in UCMs* to show more detailed behaviour. There is a different plug-in for each separate Use Case or scenario listed above, seven scenarios for the Customer and three for the Administrator.

Figure 3 shows the ‘checkout1’ plug-in map for the ‘serveCustomer’ stub. It introduces a Shop-

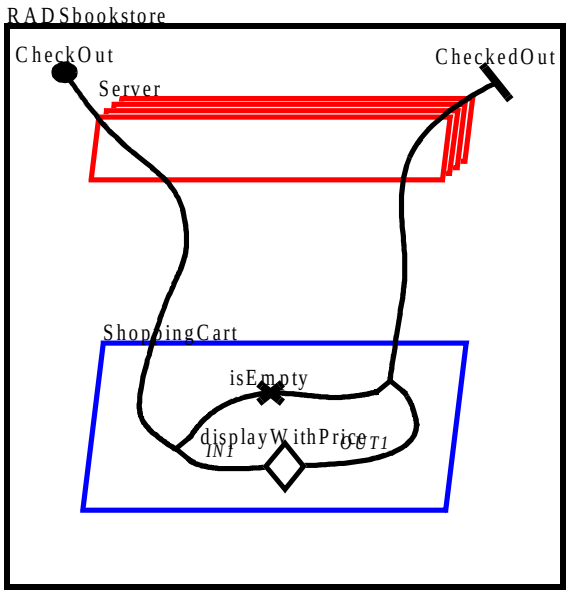

Figure 2. The first level of breakdown of the Checkout scenario

pingCart component to contain the data associated with a customer’s purchases. The Server directs the checkout request to the ShoppingCart, which in turn determines whether or not there are items to check out, and if there are indeed items in the cart displays the invoice information by executing the ‘display-WithPrice’ stub. This stub conceals the remaining detail of the scenario.

In the most detailed plug-in map in Figure 3 we see most of the conceptual architecture for the system sketched out, with components for applications called Catalogue, Inventory Manager, Backo-rder Manager, and Customer Account, and two databases. The pricing information is gathered from the Catalogue and the more complicated undertaking of shipping the items is initiated in parallel with the return of the invoice. According to their availability in inventory, items ordered in the ShoppingCart are put into two lists of items to ship now, and items to be back-ordered. Then the Customer Account component takes over recording the sale and billing the customer.

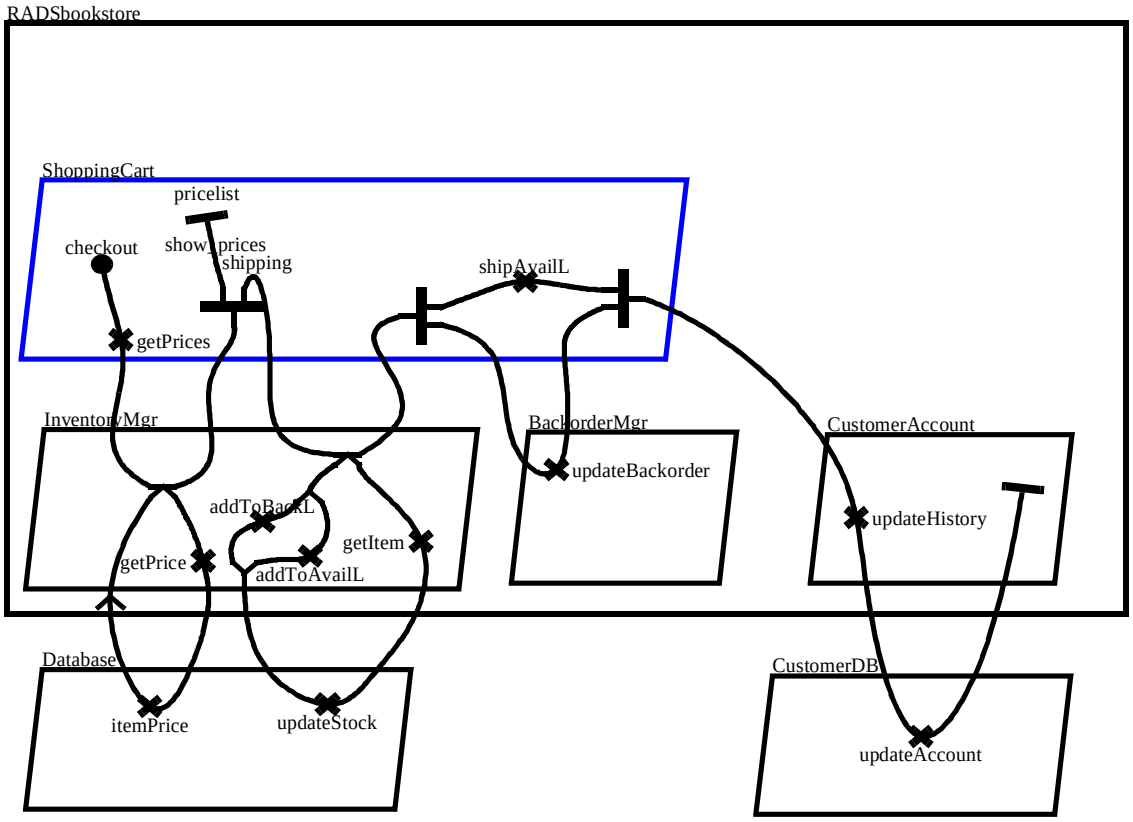

Figure 3. The detailed checkout sub-scenario, shown by the plug-in to go into the stub “display-with-Price” in the top-level map in.

The other scenarios listed above have also been described by other plug-ins, mostly less complex than Checkout. The whole collection of maps is maintained together by the Use Case Map Navigator tool [10].

Some requirements that may affect performance could include:

- integrity of information relating to fulfilling commitments, such as back-orders,
- security of customer account data
- availability (quick recovery after a failure, as well as no loss of data).

Some of these can be addressed by using a database system that ensures transaction properties of certain operations. Additional design-related issues that affect the analysis, and that arise at this point, could include:

- should the catalogue and inventory be integrated into a single database or even a single file?
- representation of the ShoppingCart as an object, as a thread, or as an entry in a database?
- should the customer be informed if some items have to be back-ordered (they are out of stock)?
- when to require authentication of a customer, as having an account already established? (at login, at checkout, or in between?)

Software Design with UCMs

Most of the techniques for building models of software require a performance-oriented specification tool such as Smith’s execution graphs or SPEED [19]. While UCMs have a superficial resemblance to execution graphs, and can record similar information, they are a software specification tool developed for this purpose, to which performance annotations have been added. This is significant because there is a movement towards this approach to modeling, by adding annotations to a software design language. However, many of the reports on ways to do this are still using the software language to create a performance-oriented specification. This point is important, because this is NOT the kind of specifica-

tion that most developers will create routinely. For example, Kahkipuro [9] has described a modeling language based on UML collaborations, but that approach uses UML to describe the performance abstractions, rather than describing software abstractions. The hope is that there will eventually be a correspondence. On the other hand, Schmeitendorf and Dimitrov [14] described in general how performance information can be integrated into software specifications, and the present work fits into their first stage ("conception and analysis"), and the analysis of use cases.

Use Case Maps are used to dig deeply into Use Cases, specify them as scenarios, introduce components, and reason about how the components should be incorporated. The definition material in [4], and the discussion in [5] describe this in depth. For example, in [15], five different concurrency architectures were developed for the same scenarios, to reason about their performance potential. Once a specification is decided, Message Sequence Charts can be generated automatically from the UCMs [10], and then the design can be carried into SDL or UML to develop it further.

3 Making the Performance Model

The steps to create a performance model are:

1. Performance parameters are added to the specification, including:
   - assignment of components to processors
   - probabilities for choice points, and for choice of dynamic plug-ins
   - mean loop counts
   - workload intensities such as arrival rates, or customer populations and their think times.

   Alternatively, the parameters can be added later to the model. For consistent understanding of the workload in the context of the specification, it is better to add them first.
2. Missing elements in the system, which can be inferred from the way the system is to be deployed, are identified and added. These elements were termed "completions" in [22], where they were discussed at length. Completions may be added using stubs, and in principle tool support could be created to patch them in semi-automatically.
3. A performance model is generated. UCM Navigator includes an automated performance model generator described previously in [13].

The target performance model is a Layered Queueing Network (LQN), which is a superset of queueing networks, with logical servers which may be software tasks or other logical resources. The concepts of LQNs are described in [7], [8] and [20], and are related to software architecture in [21]. Briefly, they capture the workload within "activities" which belong to an "entry" of a "task" running on a "host device". The host device is usually a processor, a task is often an operating system process, but may also be an object, and an entry is like a method. Activities are operations connected by sequence relationships, including parallelism. Multiple threads in a task are represented either by making it a multi-server task (for multiple symmetrical threads) or by parallel streams of activities in the task. An entry makes synchronous or asynchronous requests to other entries and to its host, in a layered fashion. Jobs are represented by autonomous tasks which generate requests into the system.

The information requirements in steps 1 and 2 are discussed together in the next section, after a brief presentation of the capabilities of automatic modeling from scenarios.

Converting the UCM to an LQN

The performance model is generated by the UCM2LQN converter [11] which is integrated into the UCMNav editor. The model format is suitable for the LQNSim simulator or the LQNSolve analytical solver.

The converter performs a traversal of the paths in the UCM design and creates appropriate corresponding LQN objects (activities, entries, tasks and processors). Activities and entries capture execution sequence, and tasks and processors represent software and hardware resources. UCM components are converted to LQN tasks, points where a path crosses into a component become entries, and responsibilities become activities. The path traversal assumes that messages are sent between components whenever a path traverses a component boundary - the path leaving one component implies a message

being sent by the corresponding LQN task, and its entering another component implies the same message being received by the other LQN task.

Messages belong to interaction patterns that must be discovered by the converter algorithm. Messages may be either *asynchronous requests* (the sender continues execution), *synchronous requests* (the sender blocks and waits for a reply), requests that are part of a *forwarding pattern* (the original synchronous request is passed through a chain of servers, with only the last server sending a reply), or *replies* to synchronous requests. Messages that do not fit any other pattern are assigned as asynchronous.

The UCM2LQN converter is able to convert all the common UCM path constructs, such as responsibilities, AND and OR forks and joins, loops, stubs, and plug-ins bound to stubs.

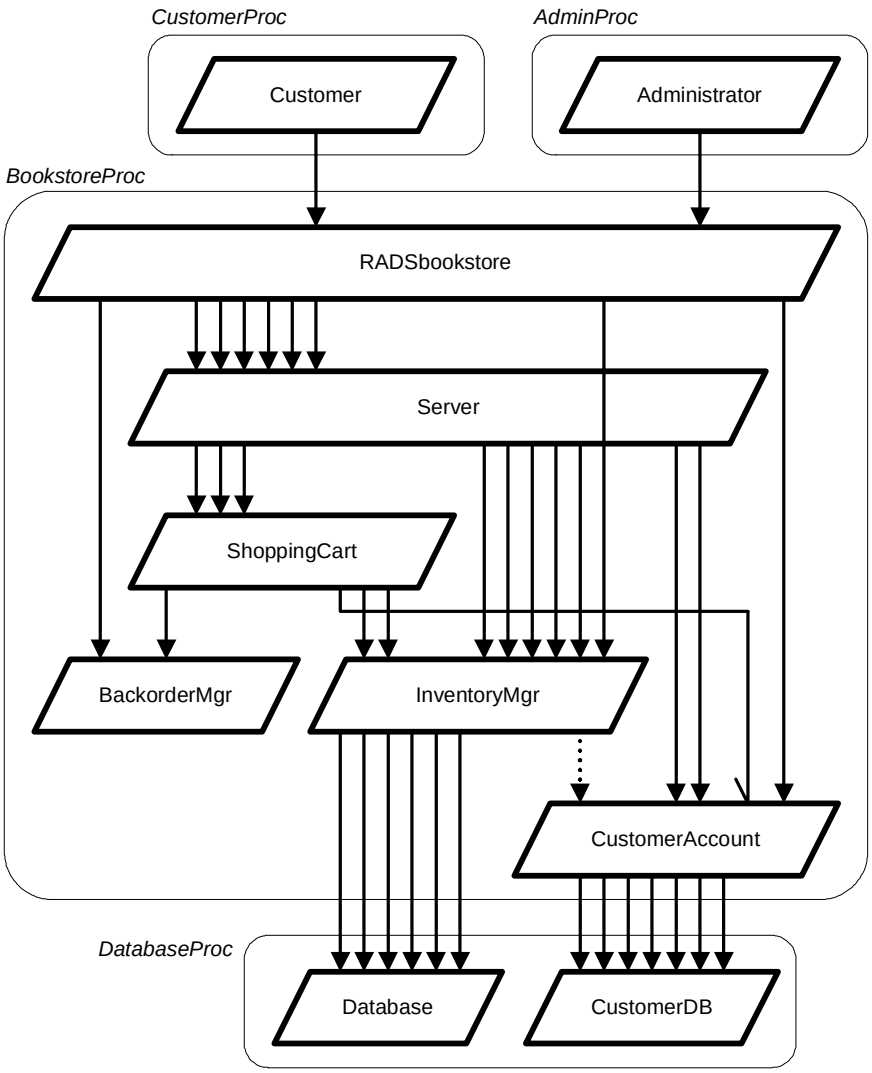

Figure 4. The RADSbookstore LQN generated by the UCM2LQN converter.

Figure 4 shows the output of the converter, but suppresses the detail of entries and activities, and the workload parameters. The multiple interaction arrows show the different numbers of different kinds of access made from one task to another. For example, tracing the paths in Figures 1, 2 and 3 leads us from the set of Customer tasks, to the RADSbookstore task representing the system as a whole (a task without functions) to the Server. The Checkout scenario then goes to the ShoppingCart task which manages the processing of checkout, with service in turn from the InventoryMgr (twice, and using the Database), the BackorderMgr, and the CustomerAccount task which uses CustomerDB to update the account.

The forwarding path shown by the dashed arrow to the CustomerAccount task is part of one of the Administrator's scenarios to ship books to a customer when a back order arrives. The inventory manager is updated and carries out the shipping, and then passes on the information to CustomerAccount for the accounting and billing.

Workload Parameters

The workload parameters can be inserted into the UCM and carried into the model by the converter. The UCM notation includes annotations for performance parameters, including:

- open or closed process for generating arrivals at the start point, with an arrival process or open workloads and a population size and delay (think time) for closed workloads,
- processors, each with a speed factor. As a default, the system always includes an infinite processor with a speed factor of unity. Unassigned tasks are assigned here.
- the association of each component with a processor to run on (default: any component without a processor is assumed to run on the infinite processor)
- processing demands for responsibilities, in the form of a CPU time or a number of processor operations (default, one unit of demand)
- probabilities for each branch at an OR fork, and for each possible choice for a dynamic stub (default: all branches are assumed to have equal probability)

These information requirements are very similar to those of an execution graph in the work of Smith [17][19], and the methods described by Smith can be used to obtain the parameter values. UCM responsibilities can also be annotated with additional demands:

- for disk operations, including the mean number of operations and the operation time,
- for named services (with a mean number of service requests). A named service identifies an operation which is not provided by any component in the specification, but will be provided by some infrastructure components to be identified. Named services are added to the model after it is generated, as *completions* (discussed below).

Values for these performance parameters can come from requirements, known performance figures for similar systems, or educated guesses. A well-equipped performance group may have built up submodels for components, or resource function libraries [23] for the typical demands made by different algorithms or operations.

The results shown in Figure 5 were computed by the analytic approximate LQN solver and illustrate this stage of the study, when the first solution is obtained. There are multiple active Customers, with a 3-second think time between interactions with the system, and one Administrator with a 100-second average interval between interactions. The administrator examines inventory and updates it when orders and backorders arrive, but (in this workload) at a constant average rate independent of the number of customers. The system throughput saturates with around 50 customers. The Customer response time is considered satisfactory up to half a second, which is reached at about the same point.

Increasing the number of customers degrades the administrator's response time up to this point, to about 12 seconds (a high level, not very satisfactory), after which it is level. Since the administrator is unaffected by customer class saturation, the administrator's bottleneck resource must be different from the Customers'. Analysis of the results shows that the Customers are queueing at the Server task, which has only 5 threads, and that within the system the Inventory manager is 100% saturated, mostly due to waiting for the Database which is 80% busy. The Administrator does not go through the Server, but accesses the Inventory Manager directly.

These results are interesting, but before reacting to them we should consider what is missing in the model.

4 Improving the Model

If we were to consider the validation of the model generated directly from the specification, we could quickly identify several types of deficiency which are characteristic of models made from specification. These deficiencies reduce the ability of the model to fulfil its purpose, which is to predict the

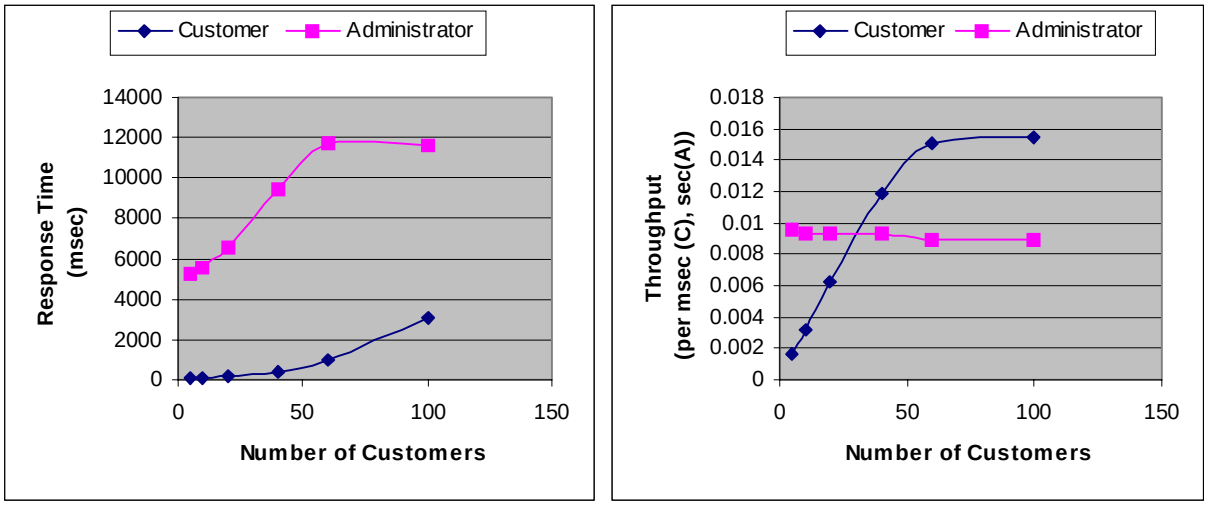

Figure 5. Response time and throughput results for the base case of the RADSbookstore.

performance of the final product, if it is implemented to satisfy the specification. Our purpose here is to overcome the deficiencies where possible, and otherwise to understand their potential impact.

Because of the nature of specifications, the performance model is likely to suffer from incompleteness and errors:

- *implied elements:* specifications normally leave out the description of the environment and services (operating system, file system, web servers, database) except to indicate where they are used; these elements are implied by decisions recorded elsewhere, or by the usual practices of the developers. This first kind of incompleteness relates to well-defined elements that are known but not defined in the specification, and is addressed through *completions* described below.
- *unknown elements:* this second kind of incompleteness relates to elements that are not well-defined, or even known to the specifiers. We attempt to address unknown elements by making allowances for them.
- *abstraction:* a third kind of incompleteness is found in an abstract description of a responsibility or operation which is specified with insufficient detail. For example a component in the specification might be planned to eventually be a subsystem with several concurrent tasks, running on more than one processor. However, there is no detail as yet for this component. This can be addressed using a suitably abstract workload description.
- *errors:* differences between the numerical values of workload parameters and the values for the software which is not yet implemented, which may be seen as parameter "errors".

If we are serious about evaluating the specification then we must be somewhat tolerant of errors and omissions due to work which is not yet done, however we should do what we can to minimize these deficiencies, or to recognize their potential impact. The following sections consider each of these problems in turn.

Completions

We will begin here with a brief summary of a treatment for missing *implied elements*, described elsewhere [16]. For implied elements, it is feasible to create submodels which can be added to the specification, or to the performance model. These submodels are called *completions* in [22], which describes possible cases in some detail, and considers how they can be composed with the model. There is considerable similarity to the problem of building a system from components, with completions playing the role of the components. As discussed in [16], automated insertion of completions can be carried out at both the UCM level and the LQN model level, although this is not yet available in the present UCM tools.

Completions typically include system services and inter-element interactions. In the present system we could consider completing the database to fill the Database component; putting in a web server for the "Server" component, including its disk storage; modeling the Internet connections between the Customers and the bookstore; modeling LAN communications between the service processors; or modeling ORB-based communications between the bookstore components and the two database components. In a typical specification the majority of the elements and the workload may be provided by completions.

In the treatment of the example in this paper, completions will be ignored just because they are such a major subject, and they have been treated elsewhere. In this paper we will assume that the workload captured in the specification parameters has been adjusted to capture the workload of the completion elements.

5 Coping with Parameter Errors (Sensitivity Analysis)

Apart from the completions, which represent the use of existing components, the workload parameters are partly specifications that must be taken as assumptions (such as the number of customers, their thinking time, the number of book items they examine and the probability of a purchase) and partly predictions (such as CPU time for the operations to be implemented, the number of disk accesses and their cache hit ratio). One cannot discover errors in the predicted values, but it is possible to estimate their effect through sensitivity analysis. Then one can be forewarned about "dangerous" parameter errors, which could degrade the performance.

In general the sensitive parameters are those associated with any heavily loaded resource. In the results for the base case of the bookstore, the heavily loaded resources are the database (about 80%), the Inventory Manager (100%, of which most is blocking for the database), and the Server pool (all of 5 threads are busy). We will consider the sensitivity to a 50% increase in the database CPU demands, and in the Shopping Cart CPU demands.

Figure 6 shows the simulation results for a 50% increase in the database CPU demands. The increase in the database demand results in a similar 50% increase in both response times for lower Customer populations, and a proportional drop (by a factor of 2/3) in Customer throughput for large population. The Customer response time has a stronger sensitivity around 50 Customers (factor of nearly 3), which is then reduced for a fully saturated system. This behaviour may be due to the limited Server threads controlling the queue length at the Database. The Administrator response time is about 50% larger across the range, and there is almost no effect on throughput.

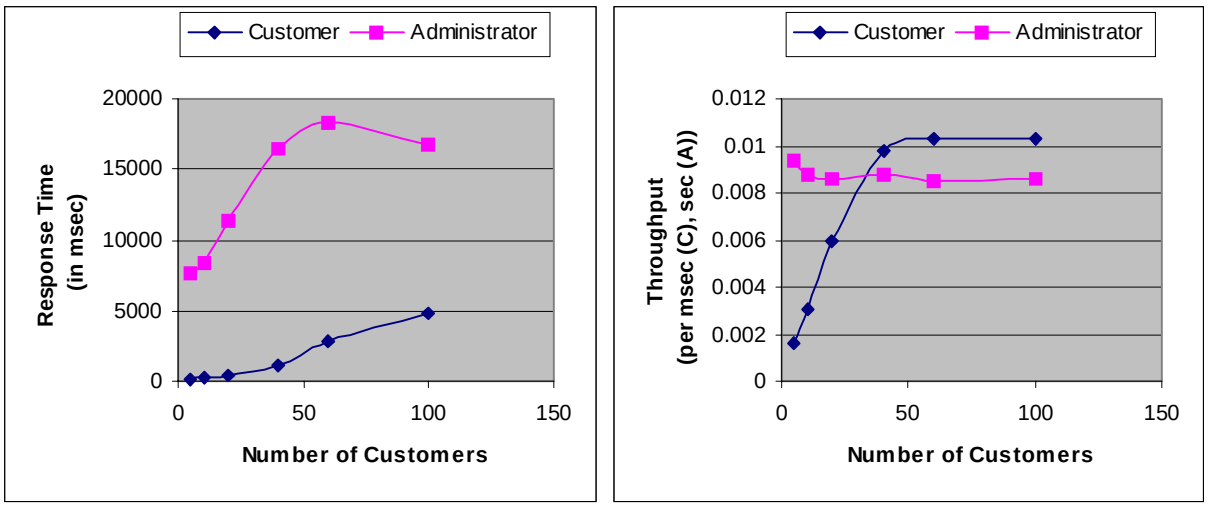

Figure 6. Response time and throughput results for the RADSbookstore with an increase of 50% in the database CPU demands.

On the other hand, the 50% increase in the CPU demand of the Shopping Cart produced a very small effect on end-to-end performance results for the Customer. This is not particularly surprising, since there is a shopping cart for each active Customer, and the processor resources they run on are not saturated in the base case (less than 20% busy, in fact). The change did produce a slight increase (about 5%) in the Administrator response time with a higher number of customers (more than 50 customers). This can be accounted for by the fact that as the number of customers increases, the Administrator competes with all of the customers in the system and the change becomes cumulative from the Administrator's point of view even though there is no discernible difference for the individual customer. However, the system can be judged not to be particularly sensitive to changes in the Shopping Cart workload overall.

The limitation of sensitivity analysis is that each parameter is investigated for a change from the base case. It ignores interaction effects. If one parameter takes a different value, the sensitivity of other parameters is changed.

Parameter Optimization by Sensitivity

Sensitivity analysis can also be applied to the planned parameters, to improve the values chosen. An example here is the value of 5 threads chosen for the Server pool; inspection of the base case model results shows a significant wait for the Server. When 50 threads were used instead, this waiting disappeared. However, as Figure 7 shows, the response times and throughput for the customer were essentially unchanged. This is because the bottleneck simply shifts downwards to the Inventory Manager, which was already effectively saturated. In the base case the Server is acting as a kind of admission control, keeping congestion out of the system without actually slowing it down. Where there is a difference in Figure 7 is the Administrator response time, which shoots up to 60 seconds at 100 Customers (compared to 12 seconds in the base case) because of the increased congestion at the Inventory Manager. Thus the change in threads consumes resources, gives no benefits and makes the Administrator response much worse.

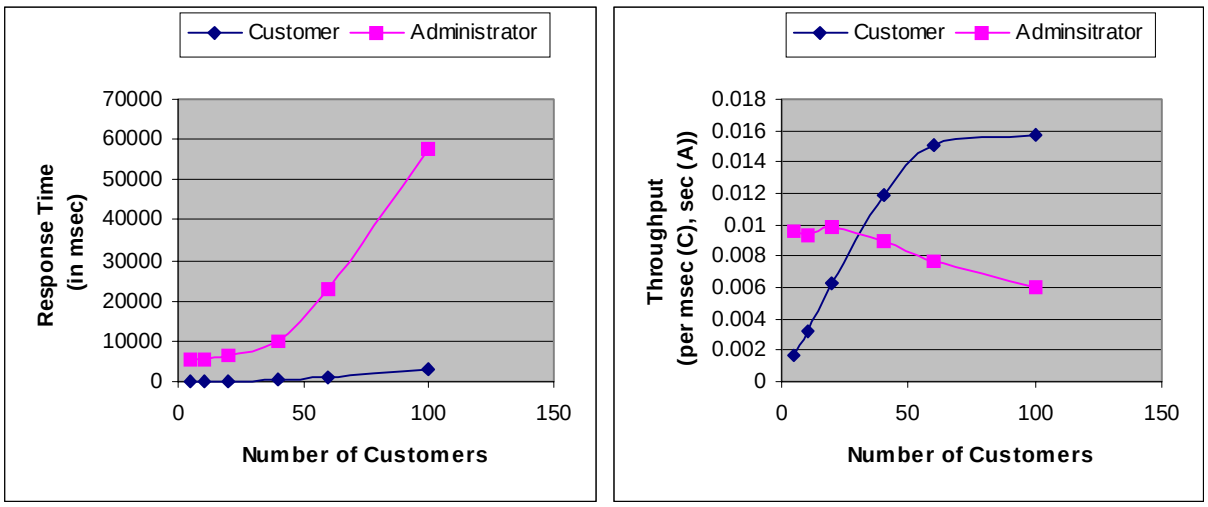

Figure 7. Response time and throughput results for the RADSbookstore with 50 Server threads.

6 Coping with Missing Elements

A missing element may contribute additional delay just due to its execution (an incremental effect), or it may be an unforeseen bottleneck (which can have a very heavy effect). The former can be handled by making an allowance in the overall delay for missing operations, which need not be attached to any particular parameter error. Unforeseen bottlenecks are more serious and need special treatment.

An Allowance for Unforeseen Bottlenecks

How can we provide a special kind of "allowance", that allows for bottlenecks in missing elements? To begin with, let us assume that we know which parts of the system are subject to possible missing elements; the paths through these parts of the system will be termed "bottleneck-vulnerable" (BV) paths. Each BV path has a throughput (representing messages, operations or sub-job initiations) which might be limited by missing elements. Even without identifying the missing elements, we can often provide a conservative throughput capacity estimate (in messages/sec. or operations/sec) for each BV path, based on general knowledge of the environment; let us call it *BV path assumed capacity* (BVPAC). The rate BVPAC can now be set conservatively, to values of processing or messaging rates that we are sure can be met even if there are missing elements. When the system is solved, the utilization of such a pseudo-server indicates how close the system is to reaching the rate limit. The system will be constrained within the limit, and the delay it imposes can be checked to see if it is a significant bottleneck.

For example, consider the paths into the databases at the bottom of Figure 4. Suppose the intention is to use some kind of ORB, but it is not clear what it will be. Since an ORB can impose substantial constraints, this path could be identified as bottleneck-vulnerable. However, suppose we can also assume that the communications infrastructure will be able to convey at least 100 messages/sec. on this path. Thus, the paths into the databases are BV paths, with BVPAC = 100/sec.

To study the impact of this possible bottleneck, a special kind of *rate-limiting completion* can be added to the BV path. For the paths to the database, this is illustrated in Figure 8 by the pseudo-task "RLC" in the messaging path. It takes requests, forwards them at once, and then becomes busy for 10 msec. before it can accept the next request. Thus it limits its rate of handling of messages to 100/sec. RLC can have a number of forms, representing for instance a leaky bucket rate controller. Here, we have used a *two-phase server* such as was studied in [8], with a postponed operation, which behaves as follows:

- the server begins a pseudo-operation and immediately releases the job, by forwarding it to its destination,
- the server then waits for a time (1/BVPAC), (the second phase) before accepting the next job that request service.

The figure shows how a second call that arrives before the busy time of the RLC task is finished,

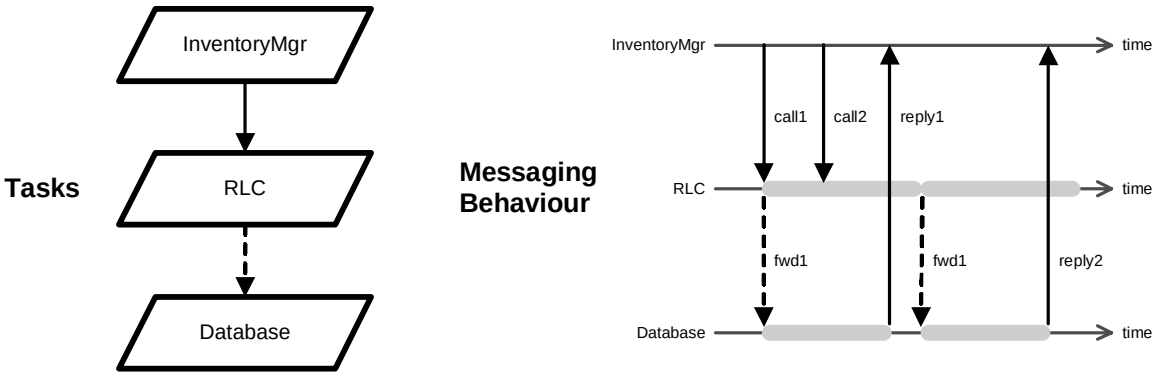

Figure 8. Inserting the Rate-Limited Completion RLC to impose a rate limit on Database accesses.

must wait, and how this imposes a throughput limit on the flow of messages. Second phases are accommodated in the layered queueing framework and in the solver used here for evaluation. It is not completely transparent at mean flow rates below its limiting rate, as it does impose some random queueing delay, however it gives an approximation to a rate-sensitive choke on a stream of jobs.

Figure 9 shows the impact of the rate-limiting completion RLC when it is added to the system analyzed in Figure 5. The customer throughput is limited at about 11/sec, compared to nearly 16/sec

before, and their delay is much increased for large numbers. The administrator throughput is almost unaffected. Thus the hypothetical bottleneck could have a significant impact. The rate of the RLC server could also be varied to estimate the sensitivity of the design to a potential bottleneck at that rate, and in that particular BV path.

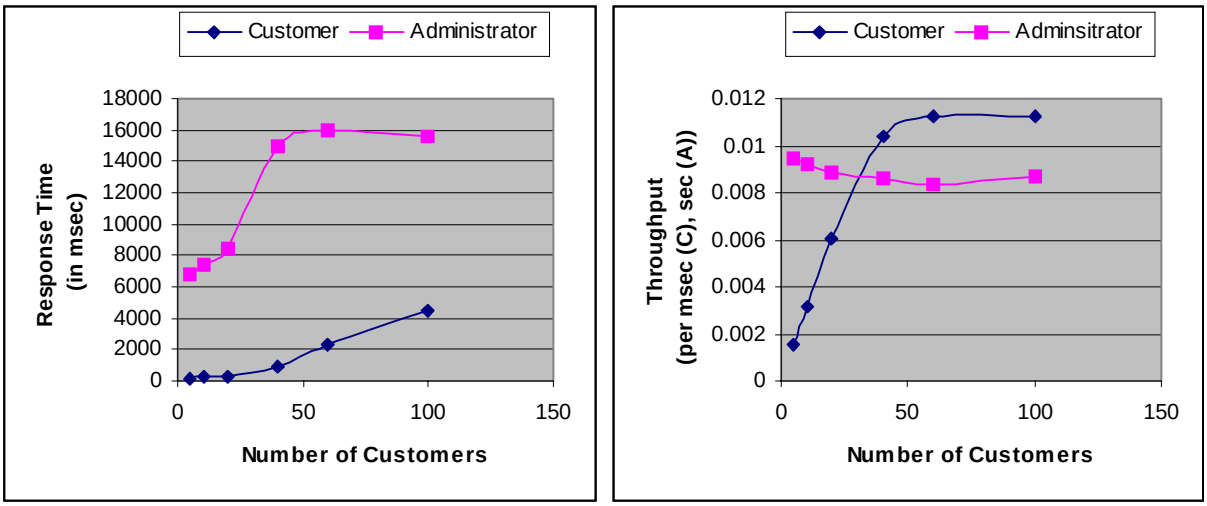

Figure 9. Response time and throughput results for the RADSbookstore with the rate-limiting task RLC between the bookstore and the database.

A similar RLC task could be used to describe an unknown service that might impose a limit on its rate.

7 Coping with Abstraction

All software specification languages support abstraction, which influences the ability to predict performance. For instance in a UCM...

- one *abstract responsibility* may stand for an arbitrarily complex behaviour, involving many detailed responsibilities over many components. We may think of this as a responsibility that needs to be expanded by a stub/plug-in, or we may think of it as a kind of aggregate responsibility.
- one component may in fact be a subsystem, running on many devices. For instance a database component implies storage devices such as disks, or perhaps a RAID array. In this case the responsibilities of the component are abstract as described in the previous item.
- a simple path crossing between components indicates a transfer of execution by some mechanism, which is not described.

A little thought shows that all these abstractions are similar, in that they place some workload on a number of devices via an unspecified path and unspecified components. Thus they are all equivalent to abstract responsibilities, as described in the first item.

To describe their workload, these abstractions could be filled in by three approaches. First, more detailed specification using stubs and plug-ins could define the missing detail. However this does not really solve the problem because abstraction of some kind is always present. Even a fully detailed program code is an abstraction for machine operations, caching, bus conflict, etc. So we ignore this approach here. Second, they could be filled by completions, which are “canned” details. We assume that where it is possible, this has already been done, so that in this section we are considering the residual abstraction. (Note that it may be inside a completion).

The third way is to approximate the workload of the abstraction, in terms suitable for the elements that are given in the specification. For an abstract responsibility, the total workload it imposes on system resources can be declared, even though the sequence of the sub-operations is not defined in detail. This approach was taken by Smith in [19] and gives useful results for systems (in this case, responsibilities) that are substantially sequential in their internal execution paths. In a layered model,

one defines a set of pseudo tasks, one hosted by each device that is involved, with the workload component for that device.

For abstract responsibilities which are known to have some internal parallelism (such as a RAID storage array, for instance, or a parallel database), one can go further and define a pseudo-task with parallel activities, which in turn access generate sequential workload components as just described.

8 Comparing Alternative Specifications

If we can evaluate, we can compare. Comparison is in some ways more meaningful than absolute evaluation, since it can be made on a level playing field, i.e. under similar limitations in knowledge. In the bookstore, the Customer class of users is bottlenecked at the InventoryMgr, which in turn mostly waits for the Database. However the Customers are mostly browsing the catalogue of books carried by the bookstore, and do not need full database capability and concurrency control. The catalogue is rarely updated, and could be separated out as a read-only database without complex concurrency control. In the present analysis a Database operation time of 20 msec was replaced by access to a separate Catalogue server with an operation time of 10 msec.

Figure 10 and 10 show the results, first without and then in combination with the hypothetical middleware bottleneck RLC. In Figure 10 the customer class sees no throughput limitation up to 100 customers, and has a constant and very small response time. The administrator response delay is also reduced from the range of 7 to 16 seconds, to a value around 5 to 6 seconds without the RLC element. In Figure 11 with RLC the Customers are unaffected by the hypothetical bottleneck. The Administrator is slightly worse, which is reasonable since that user is now the major client of the databases.

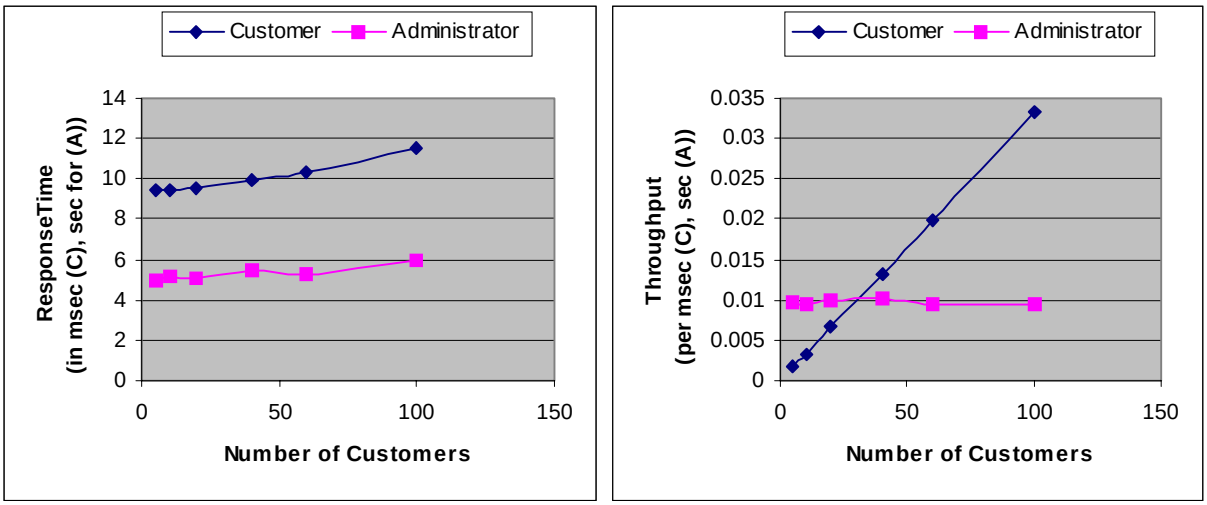

Figure 10. Response time and throughput results for the RADSbookstore with a catalogue.

It is notable that a change intended to remove a known customer bottleneck at the database has also removed a dependence on the unknown middleware to access the database.

Another possible design change is to modify the checkout path shown in Figure 3. The path is “optimistic”, in that it responds to the customer first and then completes handling the order. An alternative is to make the path “conservative”, by making the customer wait until the processing is complete. This had almost no effect on the overall average delay, because checkout is a minority of the scenarios. However a separate class analysis for checkout alone showed a modest increase in checkout response times of 15% for large populations.

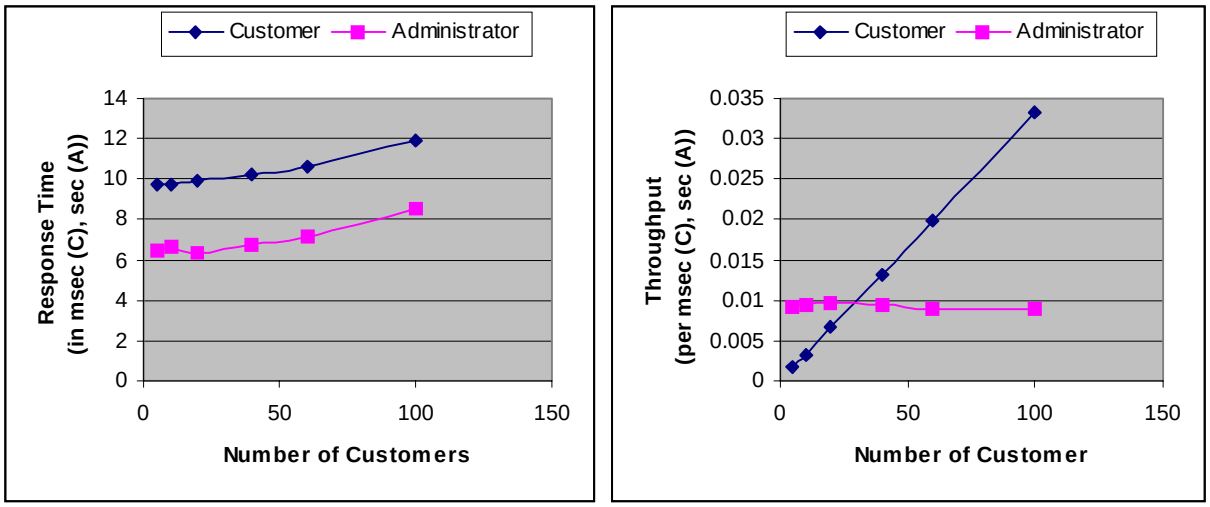

Figure 11. Response time and throughput results for the RADSbookstore with a catalogue and a middleware RLC between the bookstore and the database.

9 Conclusions

Various kinds of problems arise when analysing the performance aspects of an early software specification, due to different kinds of missing information. It is normal for these aspects of an early specification to be incomplete. We have distinguished

- *completions*, describing elements which are definitely implied by the circumstances, and which can be filled in by adding pre-defined elements to the specification,
- *errors* in parameters, whose importance can be investigated through sensitivity analysis of the existing parameters,
- *missing elements*, which are not implied with sufficient precision to be supplied as completions, but which can be characterized by potential bottleneck effects, and
- *abstractions*, which can sometimes be characterized by abstract responsibilities which specify the workload of a subsystem, or by abstract completions.

The discussion of the example has concentrated on missing elements, relating them to completions (which have been discussed elsewhere) and to errors and parameter sensitivity, which have often been considered.

Missing elements are accounted for using a new approach for representing the impact of a capacity limit in one part of the system, on the system as a whole. A kind of capacity-limiting element, called a rate-limiting completion, was described, and a specific type of two-phase server was proposed for this purpose, and used in the example to represent a saturation limit on an intertask connection.

The paper also addressed the use of a specification designed primarily for software design, rather than primarily for performance analysis (and a language intended primarily for design rather than analysis). The results are encouraging. The example shows a considerable resemblance to the models and information expressed in previous work which was specifically for performance modeling. Perhaps this is not a surprising result, but it is helpful to confirm it. It suggests that

- the presently developing capabilities to annotate specification languages such as UML with performance information will be smoothly compatible with performance modelling tools, and
- automated transformations such as the model generator in the UCMNav scenario tool will be feasible for other languages too.

The example demonstrates the power of UCMs for our purposes, in expressing a set of interacting scenarios in a single model. This can be done in UML only by defining a family of models, linked by additional information. UCMs fill a gap that is missing in UML, for developing an initial software architecture from Use Cases.

The concept of using the model to improve the design was addressed more briefly, in the context of making comparisons between alternatives. This is a major topic for performance engineering, but it is well considered in the literature.

Model validation has been considered only indirectly. Since the model is intended for insight into potential performance pitfalls, it should be validated for:

- structure, which should not be in question if the specification is correctly interpreted, apart from completeness...
- completeness (which can be considered in the categories of completions, missing elements and abstractions introduced here). Submodels used for completions should be validated, in the usual way, against the system they represent, using the well-known techniques of model validation,
- parameter accuracy.

Parameter accuracy is a residual problem, when parameters are predicted on the basis of expertise. In [16], the approach was taken of considering the parameters to be budgets, in which case they are assumed to be accurate for the purposes of calculation.

10 References

[1] S. Balsamo, P. Inverardi, and C. Mangano, “An Approach to Performance Evaluation of Software Architectures,” in *Proc. of First International Workshop on Software and Performance (WOSP98)*, October 1998, pp. 178-190.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 1998.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[4] R. J. A. Buhr and R. S. Casselman, *High-Level Design of Object-Oriented and Real-Time Systems: A Unified Approach with Use Case Maps*. Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[5] R. J. A. Buhr, “Use Case Maps as Architectural Entities for Complex Systems,” *IEEE Transactions on Software Engineering*, vol. 24, no. 12 pp. 1131 - 1155, 1998.

[6] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Boston: Kluwer, 2000.

[7] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, “Performance Analysis of Distributed Server Systems,” in *The Sixth International Conference on Software Quality (6ICSQ)*, Ottawa, Ontario, 1996, pp. 15-26.

[8] G. Franks and M. Woodside, “Effectiveness of early replies in client-server systems,” Proceedings of Performance 99, *Performance Evaluation*, vol. 36--37, pp. 165-183, August 1999.

[9] P. Kahkipuro, "UML-Based Performance Modeling Framework for Component-Based Systems," in *Performance Engineering.*, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds. Berlin: Springer, 2001.

[10] A. Miga, D. Amyot, F. Bordeleau, D. Cameron, and M. Woodside, “Deriving Message Sequence Charts from Use Case Maps Scenario Specifications,” in *Proc. SDL Forum*, Copenhagen, June 2001.

[11] B. A. Nixon, “Management of Performance Requirements for Information Systems,” *IEEE Trans. on Software Eng.*, vol. 26, no. 12 pp. 1122-1146, 2000.

[12] D. Petriu and M. Woodside, “Incorporating Performance Analysis in the Early Stages of Software Development Using Generative Programming Principles,” in *Third Workshop on Generative Programming (ECOOP 2001 Workshop*), Budapest, June 2001.

[13] D.B. Petriu and M. Woodside, “Software Performance Models from System Scenarios in Use Case Maps,” in Proc. 12th Int. Conf. on Modeling Tools and Techniques (TOOLS 2002), London, England, April 2002.

[14] A. Schmietendorf and E. Dimitrov, "Possibilities of Performance Modeling with UML," in *Performance Engineering.*, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds. Springer-Verlag, 2002, pp. 78-95.

[15] W. C. Scratchley and C. M. Woodside, "Evaluating Concurrency Options in Software Specifications," in *Int. Conf on Modelling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, College Park, MD, Oct. 1999, pp. 330-338.

[16] K. H. Siddiqui and C. M. Woodside, "Performance Aware Software Development (PASD) Using Resource Demand Budgets", submitted for publication, 2002.
[17] C. U. Smith, "*Performance Engineering of Software Systems*". Addison-Wesley, 1990.
[18] C. U. Smith and L. G. Williams, "Performance Engineering Evaluation of Object-Oriented Software Systems with SPEED," in *Proceedings of the 9th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, June 1997, pp. 135-154.
[19] C. U. Smith, L.G. Williams, "*Performance Solutions*", Addison-Wesley, 2001.
[20] M. Woodside, "*Layered Performance Modeling and Layered Queueing: Quick Tutorial*," internal report, Carleton University, Ottawa, April 2001.
[21] C. M. Woodside, "Software Resource Architecture", *Int. Journal on Software Engineering and Knowledge Engineering (IJSEKE)*, vol. 11, no. 4 pp. 407-429, 2001.
[22] Murray Woodside, Dorin Petriu, Khalid Siddiqui, "Performance-related Completions for Software Specifications", to appear in *Proc Int. Conf. on Software Engineering (ICSE 2002)*, May 2002. Report SCE-01-09, Dept. of Systems and Computer Engineering, Carleton University.
[23] M. Woodside, V. Vetland, M. Courtois, and S. Bayarov, "Resource Function Capture for Performance Aspects of Software Components and Sub-Systems," in *Performance Engineering.*, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds. Berlin: Springer-Verlag, 2001, pp. 239-256.