Performance Aware Software Development (PASD) Using Resource Demand Budgets

Khalid H. Siddiqui, C. M. Woodside
Department of Systems and Computer Engineering
Carleton University, Ottawa Canada K1S 5B6
E-mail: {khs, cmw}@sce.carleton.ca

Abstract

Performance Aware Software Development (PASD) as described here combines a software specification, a model, and resource demand budgets. The budgets are planning figures created by the designers and managers, from the requirements and their experience. The key elements of this approach are the *planning of budgets* for the resource demands of each of the parts and operations of the system, and a *validation check* (using the model) for the required performance. The paper starts from a Use Case Map (UCM) specification, but other specification languages such as UML could equally be used. Demand budgets are allocated to responsibilities and the entire budget is verified by a semi-automated performance analysis using Layered Queuing Network (LQN) models. The key step is to add "completions" to the software system design, representing those parts of the system not defined in the software specification (infrastructure such as middleware, the environment, and competing applications), which could impact the performance. Budget adjustments are indicated by bottleneck locations and the sensitivity of results.

1. Introduction

A budgeting approach to performance applies a "divide and conquer" policy. First, it divides the problem into parts that correspond to system responsibilities, and to the work of separate developer groups, then it estimates (using a model) the overall performance that will result if all the budgets are met. The model includes the effects of the execution platform, such as contention, overheads and latencies. The model evaluates the budget, and is used to help in adjusting it.

This division gives the developer a problem scope limited to his or her responsibilities, which can be understood and solved without having to understand the entire system. The understanding of the entire system is focused where it should be, at the managerial level, perhaps with some specialist participation in working with the model. Also, information about components and the execution environment can be stored, and re-used in later work.

The budgeting process in PASD adds a layer of procedures and reasoning on top of the familiar techniques of predictive modeling in Software Performance Engineering (SPE), as pioneered by Smith [18][19]. The additions include

- using the model to define goals rather than to predict outcomes,
- steps to create a feasible set of budgets (to meet the overall performance requirements)

In line with a trend also seen in other work in SPE, the model is tied closely to the software specifications. The important capabilities of predictive modeling are retained, including

- the ability to experiment with design changes
- the ability to incorporate available knowledge about performance parameters of existing components.

Thus, PASD is an incremental change in software performance analysis, beginning with a change in the meaning of familiar quantities. A major difference is in the meaning of model validation, for instance. In PASD a model is not validated by comparing it to a real system, but it is valid if it truly represents the requirements, and the intentions of the developers. This paper describes the budgeting process with a case study, and discusses the effects of the change in viewpoint.

PASD combines

- A scenario specification of the "important" operations of the planned system, showing their sequence in the execution of each type of operation, and their binding into planned architectural components. Suitable models can be made with UML sequence, activity or collaboration notations (for instance using the UML performance profile [15]), or with other notations such as Smith's execution diagrams [18][19]; here, we have used a notation called Use Case Maps [3][4].
- identification of "responsibilities" which are the granules of execution within the scenario,
- resource demand budgets for the responsibilities, determined based on guess, intuition, prediction and sensitivity analysis.
- system "completions", as discussed in [21]. These are sub models for path segments, hardware and software components, which are not represented in the software specification, but (a) are expected to be part of the final system and (b) will affect the performance.
- tool support for automatic generation of performance models [16].

The model is the key to understanding the budget. PASD does not divide the specified response time into parts allocated to each responsibility, because that does not provide development groups with practical targets for their work. What is practical for a developer of a particular feature, is a target for its resource demands. In PASD, resource demand budgets for responsibilities are established as sub goals for developers of the features that will implement each responsibility. The performance model determines the consequences of the budget, in terms that can be compared to the performance specification

Therefore, the concept of a "balanced" budget in PASD is subtler than just fitting all the demands into an available total time. Rather, a budget is "balanced" if it gives acceptable predicted overall performance, which we propose to estimate with a model. Typical performance measures are time delays and throughputs or capacities, which may be stated as mean values, mean values with a given variance, or as percentile values (such as 95% of response should be completed within a given delay).

Model solving techniques are not central to PASD. Preliminary analysis may be based on analytic approximations, and more careful analysis on simulations. Use Case Maps are also not essential to PASD. Another form of specification, which identifies the scenarios, the components, and the activities for which code is to be developed, could be used. The advantages of the UCM are that it is a small compact abstract description of the architecture, which can be created before these others and can lead into them [3][4].

An established area for performance budgeting is in real-time systems for command and control, which seek guarantees on response times given guarantees on CPU demands (see e.g. [11]). The notion of demand budgets is precisely the notion used here, as an upper bound on the "real" value, however the models and specifications tend to be much simpler. One view of PASD is that it attempts to apply the methods of [11] to non-deterministic systems, and to approximate analysis based on queues.

The form of model used in PASD is not confined to one paradigm, but this work has used Layered Queuing Networks [6],[7] because they reflect software structure and software resources, they scale up well, and they provide insight into soft bottlenecks [14]. Solution techniques include analytic approximations and simulation. The budgeting strategy described here could be applied equally well with other kinds of models, such as Petri Nets, Queue Nets, or straight simulation.

Once a model is obtained, the analysis in PASD can go in a forward direction, from budgets to performance measures such as throughputs and delays, or in a reverse direction from required values of measures back to budgets.

- The forward direction provides a *test of feasibility*. Demand budget values in terms of CPU seconds or network or storage operations are allocated to each responsibility, and the model estimates are computed and examined to see if they are consistent with requirements.
- The reverse path can in principle generate feasible budgets automatically, using an iteration or search to find budget values, which satisfy the requirements. How to do this is an important open problem, which could involve many additional constraints and heuristics. If it could be done, it could either be used from the beginning to generate a plan, or invoked to deal with an infeasible starting budget.

To go in the forward direction, one must create the initial budget values for the demands on individual responsibilities. It can be done in the same way that financial budget values are created under uncertainty, using experience and a reserve for contingencies [12]. To assist the budgeting, prototyping experiments can be done to verify feasibility of individual budgets, and the demand value can be tracked during development. Model experiments can be done to estimates the effects of different kinds of overshoots. If a particular responsibility needs more time, taking time from other responsibilities, changing the design, or boosting the hardware capability can accommodate it.

To be effective, the analysis should be carried forward into the development of the system. As the specification is refined into more detailed forms, PASD can use models created from those forms too. Other work in [5][10][23] describes a suitable technique for building layered models, based on traces created from the specification, or even from an execution of a prototype. In [5] and [20] it was applied with commercial CASE tools that specify behavior by communicating state machines.

2. Budget Analysis Road Map

Figure 1 illustrates the process in PASD for analyzing budgets. The boxes represent design artifacts or libraries used in the analysis, while the arrows show operations. There are seven steps.

Step 1: Designer Specification as a UCM

The designer specification captures the system behavior as a set of scenarios, at a suitable (high or low) level of abstraction, and defines software components and responsibilities in the scenarios. Use Case Maps UCMs [3][4] are a suitable form, which we have found to be flexible and capable for very early specification. They have an advantage over Message Sequence Charts, Sequence Diagrams, and other competing models in representing complex scenarios, and in reasoning about scenarios (for changing the location of a responsibility, for instance [17]). Also, they can represent multiple scenarios in a single model.

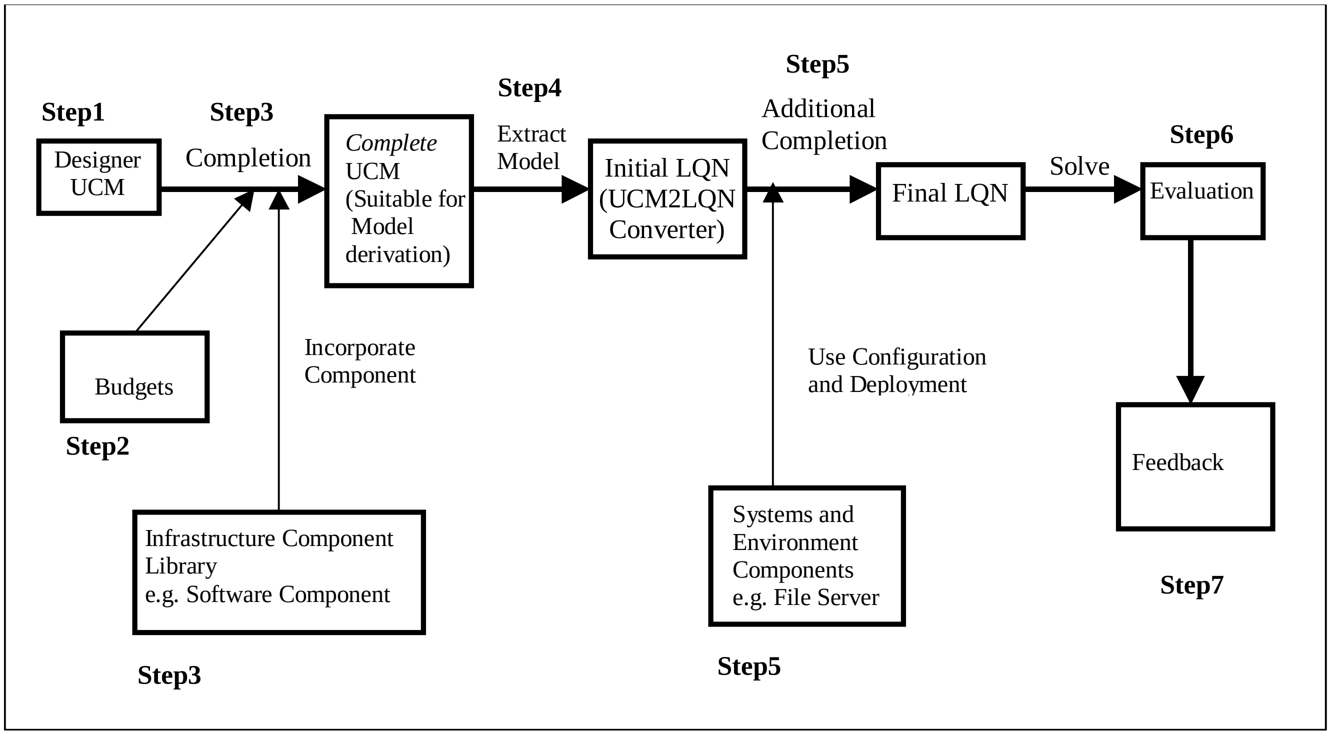
Step1
Designer UCM
Step3
Completion
Complete UCM (Suitable for Model derivation)
Step4
Extract Model
Initial LQN (UCM2LQN Converter)
Step5
Additional Completion
Final LQN
Solve
Step6
Evaluation
Budgets
Step2
Incorporate Component
Use Configuration and Deployment
Feedback
Infrastructure Component Library e.g. Software Component
Step3
Systems and Environment Components e.g. File Server
Step5
Step7

Figure 1: Budget Analysis Road Map

Step 2: Demand Budgets

The budgets are values for workload demands by the responsibilities in the UCM. Demands may include CPU operations and storage and communications operations, and may be in units of operation counts, or CPU-seconds on a known processor. These budgets are to be mapped to development of particular features or sub-components in the system, so the specification granularity should be fine enough to identify these parts. Budgets may be derived in a forward process from proposals for each responsibility, or backwards from the performance requirements, using the following management patterns.

...Demand Budget Management Patterns

Patterns for the process of developing the initial budget values are:

- *Assumed budgets:* if a software system consists of components which are new, unfamiliar types or used first time in the budget analysis then CPU demand budgets of such components are assumed or guessed by the design team.

- *Derived budgets:* if the system components are new but of familiar types then the budgeted demand values can be derived from the performance requirements themselves (dividing up the allowed delay into parts allocated to different responsibilities). Or the data for the CPU time demands of the system component is not available then the sensitivities of the performance metrics can be considered as targets to budget the CPU demand for each responsibility.
- *Reused budgets:* existing components can be given budgets based on experience (these components may have the role of completions).

- *Budget revision:* if the desired performance targets are not met then it is required to refine the budgets. It can be done make by making adjustments in the CPU demand of one operation by reducing the CPU demand time and by allowing more time to other operations. Since every operation is time bounded that’s why time reduction must be within allowable limit. The example of such systems is mission critical or real time system where the operation CPU demands are time restricted and the operations finish their jobs within the specified time constraint. Therefore, if the budget revisions are not possible in such system then the architecture adjustment can be an alternate option for achieving desired performance goal. Time allowance from one operation to the other operation is acceptable in non-real time systems for the budget revision and refinement where one operation time is reduced and the other operation is allocated more time to finish its job for the desired performance.

Step 3: Completion at the UCM level

The software specification normally does not fully define the system. The earlier the specification, the more abstract is the description, and this is sometimes taken as a reason to delay evaluation. However, it is often true that the missing elements are left out because they are understood by all and *need not* be stated. These implied elements can be inserted into the model and have been called completions. A thorough discussion is given in [21]. Some missing elements require more thought, for instance the planning may imply multiple deployments in different environments, and with different components. For performance analysis some of these deployments must be defined at least approximately; the amount of detail to be included is a matter of judgment. These are also completions.

The missing detail can be added to the specification (at the UCM level), or at the performance model (here, LQN) level. If its operation is very context-dependent the former may be more suitable, but there is no hard rule. PASD includes a library of UCM “completions” which can be added as *stubs* to the designer UCM. A stub represents an operation with detail defined in a hidden sub-map. At present the analyst must explicitly indicate where they are to be added, but it appears possible to add these stubs semi-automatically, guided by rules or preferences input by architecture team.

Step 4: Create the Performance Model

The process of making the performance model is a serious barrier in many organizations to early analysis, which is perhaps why it has received so much attention (e.g. in [18][19]). To make a model by hand from the specification requires a modeling expert. On the other hand the core of the information governing the model is already in the specification. If it could be extracted automatically then a major bottleneck to the analysis could be removed, although modeling expertise is still required to interpret the result.

Automatic extraction of performance models from software specifications has been described for UML Activity models [15] and for UCMs [16]. Here, the first version of the performance model is an LQN generated directly from the scenario specification, in this case using a program called UCM2LQN, which is described in [16]. The resulting LQN model is normally still somewhat incomplete, from the point of view of the expected execution environment.

Step 5: Completion of the LQN Model

The LQN model is completed by adding details of the hardware and software aspects of the execution environment. Examples of “completions” include file servers, protocol stacks, web servers, network latencies and processor speeds. These may be defined at this step if it was not done earlier.

Step 6: Evaluation

The performance model is solved and the results are compared to the required values. One modality is to use the performance results to verify that, if the budgets are met, the performance will be adequate. A second modality is to derive equipment capacities (such as processor speeds) that, given the budgeted demand values, will allow the performance goals to be met.

Step 7: Feedback

Feedback depends on the evaluation results. If they are not satisfactory then some changes are required. If predictions show inadequate performance, one approach is to tighten the budgets until the predictions are in the green zone. Alternatively one could adjust the design in the UCM domain, or the implementation options in terms of “completions” and the environment.

3. Example: Analyzing a Video Server Specification

This example has substantial complexity and a variety of performance aspects. It represents a system during its early specification, so that no implementation exists. As a result, model validation only arises to the effect that the model fairly represents the software specification and the intentions for deployment.

The system supplies on-demand video clips in a corporate environment, through a web server interface. The user chooses a clip by selecting it on a web page, and the request to play it goes to a specialized video server, with a high-performance video store. The video server creates a connection to video player software in the user’s workstation, initializes it and then plays the video. The specification used here leaves out some controls, such as a path to abort the clip, or to control the presentation (pause, fast forward, etc.) for simplicity and because they are judged to be secondary in the present performance evaluation.

Step 1: Designer UCM

Figure 2 shows a video server accessed through a web server. The user, with a web browser, examines web pages, indicated by a loop through the web-server script and the database of video clips, until a satisfactory video is selected. The server confirms the operation and sends the request to ‘VideoServer’.

The UCM notation [3][4] describes a scenario by a path, starting from a filled circle an ending in one or more bars. The path crosses components indicated as boxes, and have responsibilities indicated as X’s. The path may fork at a bar with multiple paths coming out, and join at a bar with multiple paths entering it, as indicated in the Video Server. It may follow alternative sub paths (not indicated in the Figure) and loops. Where a path crosses from one component to another, the mechanism of transfer is not specified. Details can be filled in using a symbol for a nested diagram, called a stub, with one or more plug-ins defined by other diagrams.

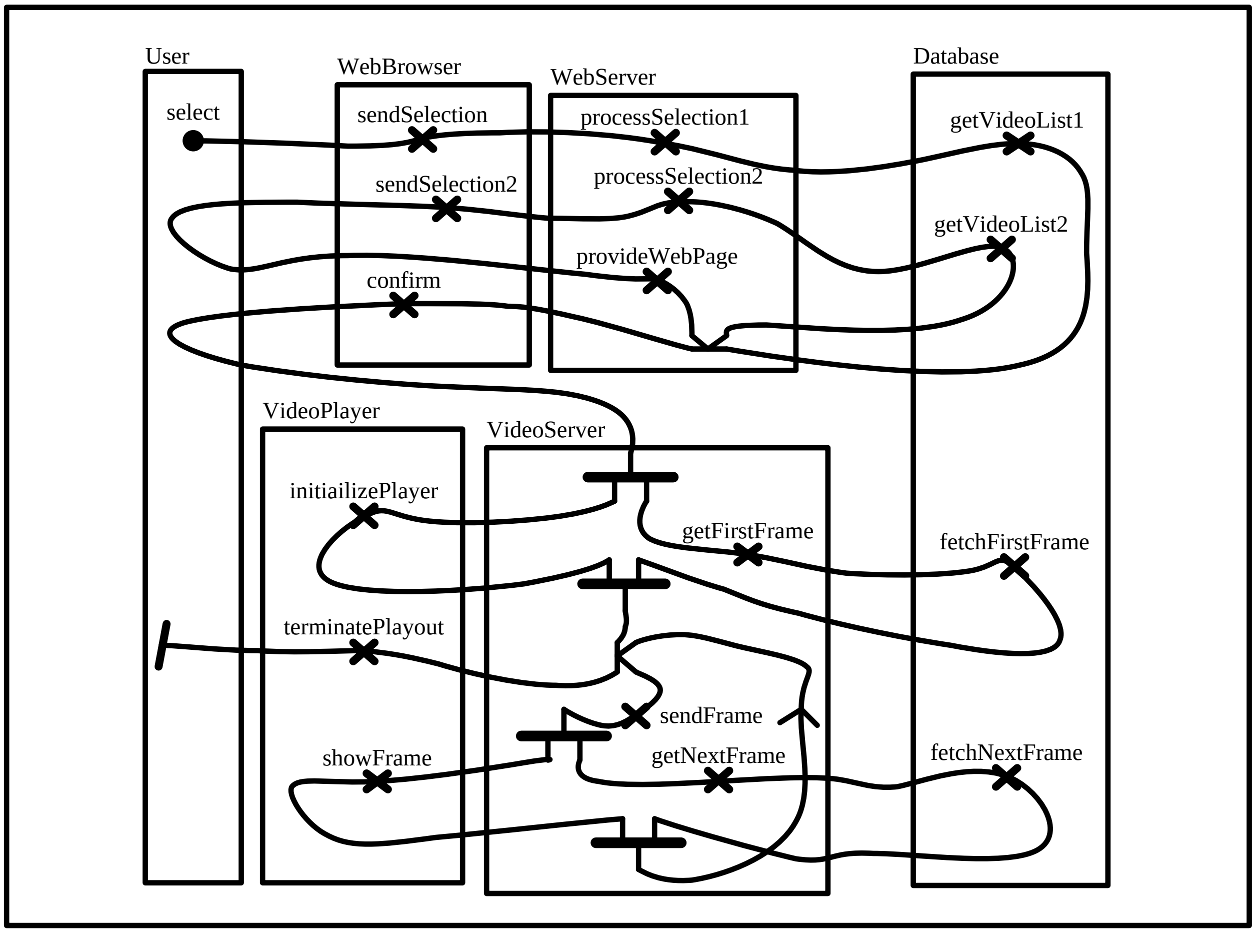

Figure 2: The Designer UCM

The ‘VideoServer’ goes into a parallel operation, which initiates ‘VideoPlayer’ in the user workstation, and (in parallel) fetches the first video frame from the ‘Database’. Then it loops through the remaining frames, doing another parallel operation in each pass through the loop, to send the previously fetched frame to ‘VideoPlayer’ and fetch the next frame (in parallel). Each frame is received by the ‘VideoPlayer’, and confirmed to the ‘VideoServer’ (a representation of flow control to send one frame at a time), and displayed. The last frame is followed by a termination indication.

Responsibilities in the UCM are interpreted for performance analysis, as operations. If a responsibility in the UCM is really an abstract representation of work done by several components, it needs to be refined to a point where each part is executed by one component, before continuing.

Step 2: Budgets

For an initial set of values, we used:

- one millisecond of CPU time for all the “small” responsibilities in the server system, including web server and video server operations, with the exception of...
- 10 ms for the database accesses,

- 4 ms for the ‘VideoServer’ to operate on a frame before sending it,
- the video player was budgeted at 5 ms to receive a frame and acknowledge it, and 10 ms to uncompress it and format it for the display. The display processing comes *after* the reply shown in Figure 3.

Step 3: UCM Completion and Infrastructure Components

The Designer UCM shown in Figure 2 does not describe the deployment or the communication infrastructure between different communicating processes. The deployment involves the following processing nodes:

- users at PCs which run the browser and the video player,
- a web site running the web server,
- a separate database node at the web site,
- a video server node, and a video store node. The video store is a specialized file server.

The communication infrastructure is planned as:

- use of an ORB and a local network between the web server and the database or video server,
- a local high-speed link between the video server and the video store (which is essentially a file server),
- an internet connection from the user terminal (the PC with the browser and the video player) and the web server or video server

Object Request Broker (ORB)

The ORB is middleware following the Common Object Request Broker (CORBA) standard [24], which provides transparent access to services registered with it. The ORB interface can connect processes based on different standards, languages, and operating systems.

There are several ways of deploying an ORB, with different performance implication [1]. Here, the client (which is the web server component) communicates with a local agent on the same node, which makes requests to the ORB components running on a central ORB site (these could be replicated but are not in this completion). It first determines the server to use, with the “trader”, then determines the server address with the name service, then forwards the request to the database server.

ORB-based communications is rather expensive, since it includes the overheads of Remote Procedure Calls. The client calls a stub, which marshals the data into a standard ASCII format and sends the request. Eventually the message arrives at a server stub, which unmarshals it into the local data format before the server can process it. The reply is handled the same way in reverse. Various possible ORB optimizations are not represented in the UCM, for example the agent can cache the results of its lookups, and the marshaling and unmarshaling can be bypassed if the data representation is guaranteed to be the same. In this sense the completion sub-model is conservative.

Besides structural completions, some decisions about the case to be analyzed can be made at the UCM level. An example is the length of a video. We will consider here that videos are for in-house corporate presentations and they are short (2 minutes, as an example). At 30 frames per second, the scenario includes 3600 frames, inserted as the repetition count $nframes of the loop at the bottom of Figure 3.

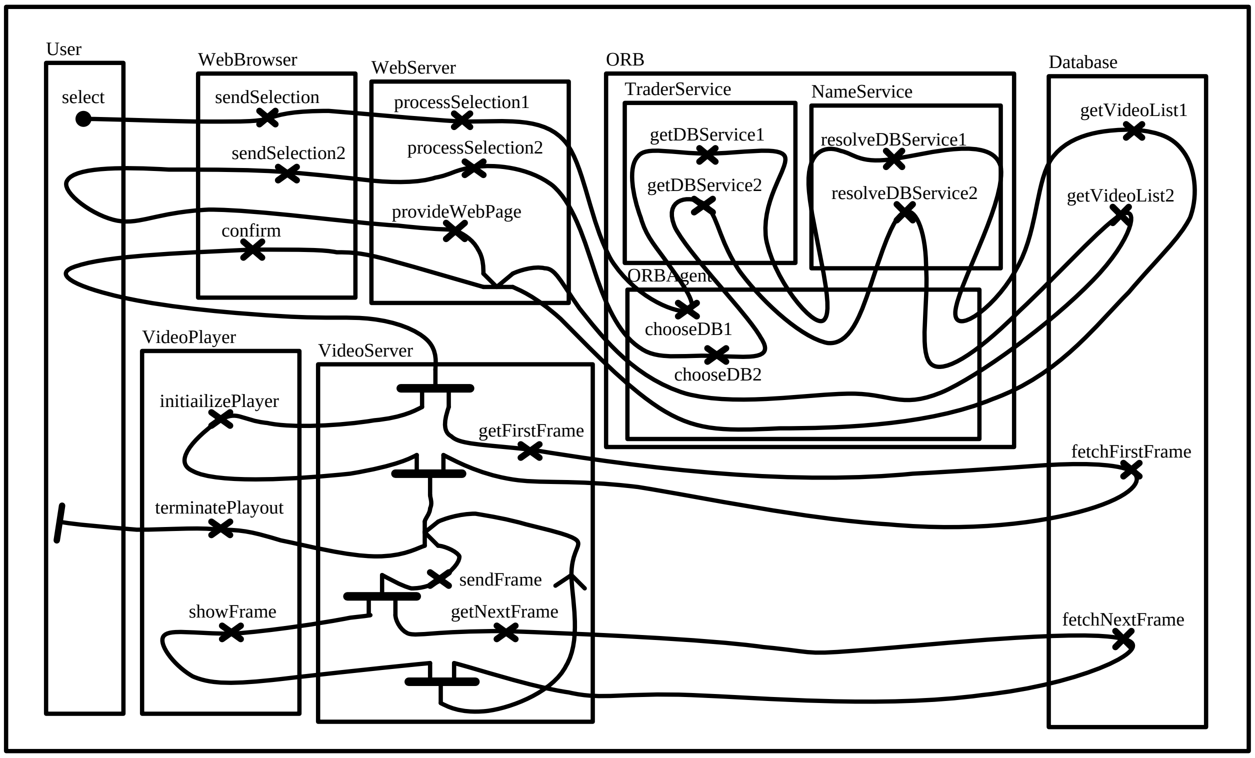

Figure 3: The UCM with the ORB completion inserted

Step 4: Make the performance model from the UCM

The model in Figure 4 is a hybrid of hand and automatic model building, which has simplified the model for presentation. Some extraneous operations were removed, and others aggregated, such as the two loops through the database being aggregated in the browse operation.

A Layered Queuing Network (LQN) model generalizes a queuing model, to include layered servers, which can make requests to other servers while they operate. These layered servers are used to represent software entities and resources such as server threads, buffers, critical sections, and other resources or modules, as well as hardware devices. The layered servers are called *"tasks"*, and the system behavior is represented in terms of requests for service between tasks and the queuing of request messages at tasks [11], which accept these messages at an LQN *entry*. An entry models one service provided by a task, with the associated resource demand (if a task is a kind of object, an entry is a kind of method). The task is the unit of concurrency and is treated as a scheduled concurrent operating system process or thread, which runs when it has a request.

In Figure 4 the tasks are the long horizontal boxes, with the task name attached to a shaded area at the right-hand end, and sub-boxes representing entries arrayed to its left. The task at the top representing the set of 'Users' and their 'Browsers' (shown with a multiplicity of $nusers) is assumed to cycle forever, and make requests to the web server entries 'browse' and 'watch'. Requests are shown by arrows from entry to entry, with filled heads for synchronous requests (that block for a reply) and open heads for asynchronous messages (no reply). The number of requests can be labeled on the arrow if it is other than one; for example, one user session is assumed to generate an average of 4 browse requests for one video to watch. The interaction types affect performance because they affect task blocking, queuing delays at devices and messages queues, parallelism, and resource contention.

The large box for the entry playout of the ‘VideoServer’ task encloses a detailed definition of the execution of the entry in terms of activities with precedence (precedence is indicated by arrows with small closed heads). The activities include two sections, which interact with the ‘VideoPlayer’ and ‘VideoStore’ in parallel, first to initialize the interaction and then to play all the frames. Each frame is fetched from the ‘VideoStore’ while the previous one is sent to the player. The activity, which sends the frame, waits for a reply, to avoid over-running the buffering in the PC.

The dashed arrows in Figure 4 indicate *forwarding* of the playout request from the ‘WebServer’ to the ‘VideoServer’ and then, when the playout is finished, to the ‘VideoPlayer’. The semantics of the forwarding interaction in layered queuing is a handoff of a request to another server, with an eventual reply going from it to the original requester. The User remains blocked throughout the playing of the clip. There is also forwarding of the ‘WebServer’ request to the ‘Database’ through the ‘ORB’.

Step 5: Additional Completions, Environment and Assumptions

The deployment of the system, if not specified earlier, can now be filled in with processor resources for each node, storage resources such as disks to carry out the storage operations specified in the budgets, and any missing connectivity. The parameters involve known resource properties (such as network speeds and disk access times), assumptions (for parameters such as cache efficiencies) and possibly even additional budgets, if a component has to be purchased and contributes to the workload. That is, a hardware component purchase, or a software component purchase decision in a COTS project, might be governed by an execution time budget for the operation of the component.
In this system we should consider as completions:

- networks linking the nodes. We will assume that the back-end networks linking the ‘WebServer’ to the database, and the ‘VideoServer’ to the ‘VideoStore’, have ample capacity and introduce negligible additional delay. The Internet link between the user and the system however has a latency, which could affect the time to handle a frame in the parallel path that delivers frames. This path shows that each frame must be acknowledged before the server will deliver the next one. Since it might impact the throughput, this latency will be included.
- the ‘Database’ and ‘WebServer’ disk subsystem should be included, but instead, an allowance for retrieval is built into the service time of the task. This approximation is justified by the small number of accesses in a scenario, and by knowledge that these system elements will not saturate.
- the ‘VideoServer’ storage path. Since this is an area under high stress during the delivery of videos, it will be modeled in more detail. The ‘VideoStore’ task in Figure 4 is a single threaded task with an execution demand of 6 ms per frame, which replaces the ‘Database’ in Figure 4 for video frame access. It is modified to have execution demand of 1 ms per frame, multiple threads, and a demand for 5 disk operations which each require on average 1 ms access time. For a single frame request this gives the same delay, but it allows for multiple disks and concurrent access by several connections. The base configuration has one disk.
- operating system and protocol execution overhead should be rolled into the responsibilities of the tasks.

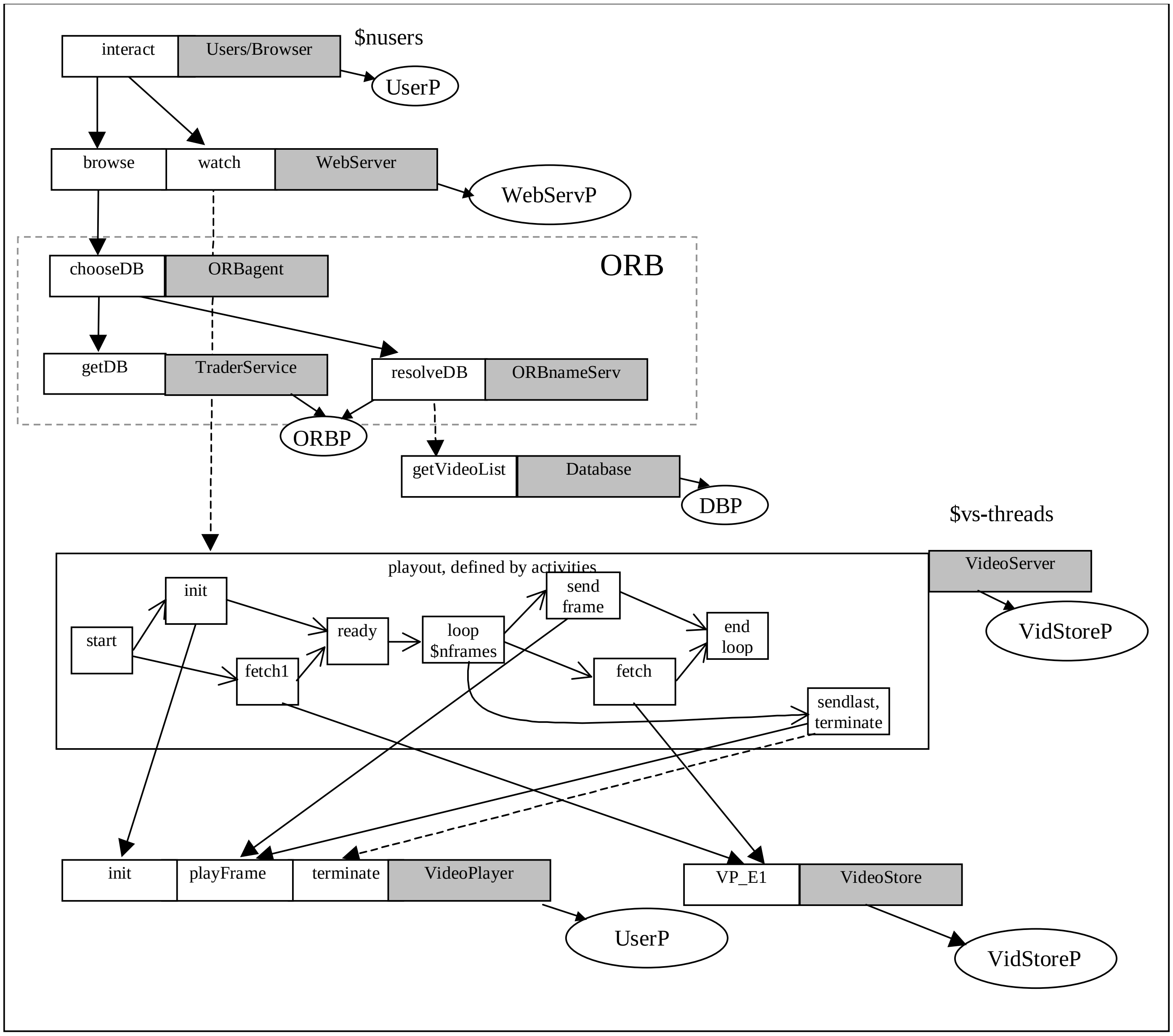

Figure 4: Video Server LQN Model with ORB Completion

LQN Communications Completion Filter

To assist with insertion of completions for communications paths, a filter has been developed which adds components to an LQN model. The *LQN Completion Filter* tool [26] can insert, in place of a message transfer between two entries,

- a latency or pure delay (point to point),
- a single network component (point to point),
- a subsystem of arbitrary complexity (point to point),
- a multicast network, (point to multi-point, to handle multiple messages in parallel).

A general tool for adding a server subsystem into a model has also been prototyped, based on an analogy to component-based systems development tools.

Step 6: Experimental Results Evaluation

The LQN model can be solved by analytic approximations for mean throughput and response times; the simulator gives more detailed measures. The LQNS solver is described in [6], and there is a utility SPEX for sequencing repeated runs over ranges of parameter values [22].

The first results obtained from the model provide insight into both the system and the budgeted resource demand values. They give:

- a feasibility check on the initial budgets, which is the main goal,
- an idea of where the resource bottlenecks are likely to be.
    - Hardware bottlenecks are due to saturated devices
    - Software bottlenecks are due to inadequate thread resources, or exclusive access to software resources
- sensitivity results for budget values, and value assumptions.

The base case provides a variety of measures; we will focus on the ‘playout’ time, represented by a variable $watchtime, which is the average predicted time to play the nominal 2-minute video. The model does not include a timer to pace the frames, so in the model the frames are played as fast as possible... if the predicted mean duration $watchtime is under 2 minutes (120000 ms, as the model is based on a ms time unit), we will say the requirement is satisfied. This interprets the requirement as permitting some variation in the pace, around an average of 30 frames a second.

For the evaluation, the number of simultaneous users of the system was varied from one to 30, and the model showed a playout time given in the first column of Table 1. One user can be served in about half the required time; two users (not shown) in about the required 120000 ms, and beyond that the system is too slow. So, the base system with the given budgets can handle two users. In the base system also, the five ‘VideoServer’ threads and its processor all saturate at five users, but the ‘VideoStore’ only reaches about 60% utilization.

4. Sensitivity Analysis

Most of the interpretation of the model rests on its sensitivity to changes in the budgeted values, which in turn is tied to resource saturation. The saturation of the ‘VideoServer’ indicates that its workload is more important to the system performance than the workload of other components. One implication of this is that budget errors or overruns on the ‘VideoServer’ are more important. The degree of importance can be checked with sensitivity calculations, for example to compare the impact of changes in

- the CPU demand of the ‘getFrame’ entry, which is the Video Store operation. It is nominally budgeted at 1 ms. Because the ‘VideoStore’ is not saturated, this is not expected to be very important, even though it is in the main loop of fetching and sending a video frame.
- the CPU demand of the ‘fetchFrame’ activity in the ‘VideoServer’, which is nominally budgeted at 4 ms. Because the ‘VideoServer’ is saturated, this is expected to have a stronger influence on the total delay.

Table 1 shows that the effect of changing the demand of ‘getFrame’ is negligible. Column 3 shows the playout time calculated for a smaller demand by a factor of 2 at 0.5 ms; column 4, for a demand which is doubled to 2 ms. The values do not even increase or decrease systematically, the changes being within the scale of the numerical approximation errors.

$nusers	**$watchtime for different CPU demands of get Frame**		
	1 ms	**0.5 ms**	**2 ms**
1	67611.1	66726.5	71588
5	161419	161713	158562
10	334976	337232	324602
15	486701	489050	478132
20	641449	642612	636427
30	959064	959423	959639

Table 1 Response Time (msec) for different numbers of users for different budgeted demand values of getFrame (showing small impact)

Table 2 shows the same data for a change in the demand of ‘fetchFrame’, (from the base value of 4 msec., down to 2 and up to 8) with a very different effect. For fetchFrame the impact is quite strong, and is larger for larger values of $nusers, at which the system is saturated. The plot in Figure 5 shows the same numbers, and emphasizes how the effect of an increase is greater (in proportion) than that of a decrease (the difference is more than twice as large), because of the non-linear impact of saturation.

$nusers	**$watchtime for budgeted demands of fetchFrame**		
	4 msec	**2 msec**	**8 msec**
1	64309	57418.7	79439.1
5	124425	114779	173078
10	258721	238915	362144
15	379395	350552	528554

Table 2 Playout Time (msec) for different budgeted demands of fetchFrame

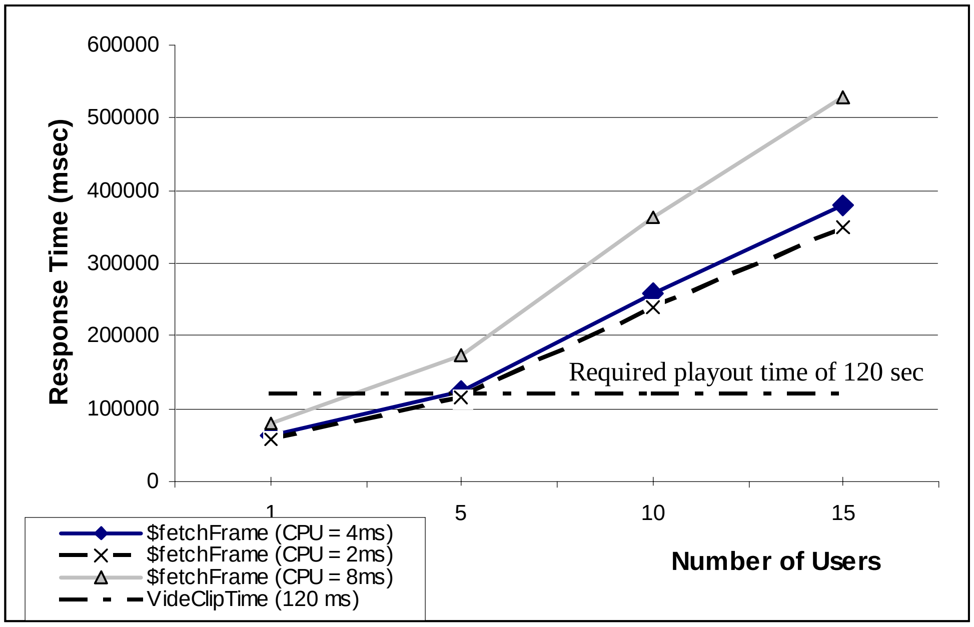

Figure 5: Playout time (msec) for different budgeted demands of fetchFrame

5. Budget Adjustment

Like any other budget, PASD budgets need to be adjusted to cope with changes in circumstances, including changes in requirements, and to better knowledge of the actual parameters. Consider the scenario:

- the budgeted demands are feasible, until some development group finds it cannot implement its features within the demands. They propose a new figure, which is not feasible. Some increment of demand must be taken from another responsibility, to restore feasibility.

PASD budget adjustments can take advantage of the sensitivity information that can be provided by the model, and of the identification of saturated resources. For each saturated resource, there is a set of budgeted demands that impact it, and a change in one of these must be offset by changes in the others. For hardware resources, the total demand per response can be expressed as $D = \Sigma\ Wi\ Bi$, where *Bi* is the *i*th budgeted demand on the saturated device and *Wi* is a weight reflecting how often that demand hits the device on average, in each response.

For example, suppose that the budget for ‘fetchFrame’ cannot be met, and the operation needs another half millisecond of budget (making 4.5 ms instead of 4). This can most readily be offset by a compensating change within the ‘VideoServer’; other components are not critical. ‘fetchFrame’ is executed 120000 times during one clip; it is impossible to find this time by reducing the initialization or loop overhead, which is only executed once. The only possible place is from ‘sendFrame’, which might be reduced to 0.5 ms from 1 ms.

To trade off a change in ‘fetchFrame’ with a change in some other component, we would use the relative sensitivity of the two budget values, to find a change with an equal impact.

6. Design Adjustment

If a way cannot be found to shift demand from one budget to another in the critical resources, then a more structural change in the design may be found. For example, we could try to reduce the delay waiting for an acknowledgement after sending each frame. This in effect establishes flow control over the connection. If an output buffer were added to the video server, so the server only waits when the buffer is full, then the server would have to wait less often and it might run faster, without reducing its demand. As a bonus, it might be more resistant to increases in the network latency, so delivery of video clips to remote sites would be possible. In the model, the buffer is simply placed in the path of sending the messages to the Players, so that some number of frames can be buffered.

$nusers	$watchtime for different values of delay		
	3ms	10ms	30ms
1	67611.1	107840	231898
5	161419	285676	638553
10	334976	615757	1243670
15	486701	875547	1924750
20	641449	1144890	2573830
30	959064	1691770	3869880

Table 3 Playout Time (msec) for different values of delay

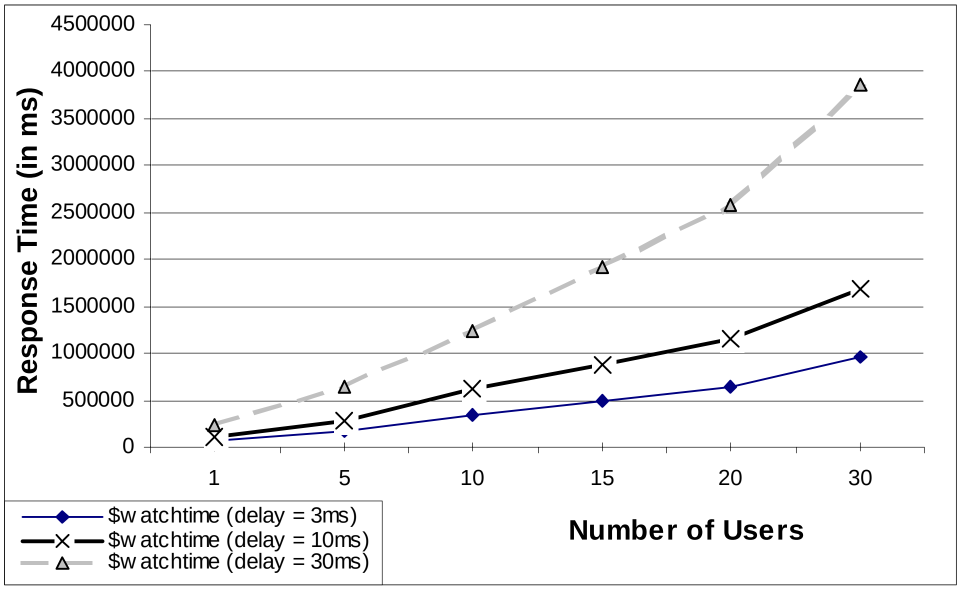

Figure 6: Playout Time (msec) for different values of delay

The results in Table 3 and Figure 6 show the effect of introducing the buffer, for the base case of a short 3 ms network latency, and for increased latencies of 10 ms. and 30 ms. The increased latencies greatly affect the playout time, with 30 ms delay making the design infeasible even for only one user. The reason is not difficult to find... it is the wait for an acknowledgement after each frame, in the basic specification. With some additional buffering, this waiting can be eliminated. The model was modified to include five output buffers in the 'VideoServer', so the server blocks in the send only if the buffers are full (on average, one buffer is provided for each thread in the 'VideoServer'). The results are dramatic, as shown in Table 4 and Figure 7. The 10 msec latency is masked right up to 15 users; the performance is as good as for 3 msec. latency. Thus the design change compensates for the added latency up to 10 msec. And for a single user, the buffered system is better with 30 msec latency than the unbuffered system with 3 msec latency.

$nusers	No Buffers			5 Buffers		
	3 msec	10 msec	30 msec	3 msec	10 msec	30 msec
1	64309	107840	231898	53052.8	53708.2	56480.7
5	124425	285676	638553	123651	124421	153494
10	258721	615757	1243670	257823	259515	317524
15	379395	875547	1924750	376282	378392	469064

Table 4 The Playout Time (msec) to the number of users with network latency values of 3, 10 and 30 msec., and with no buffer and with 5 buffers

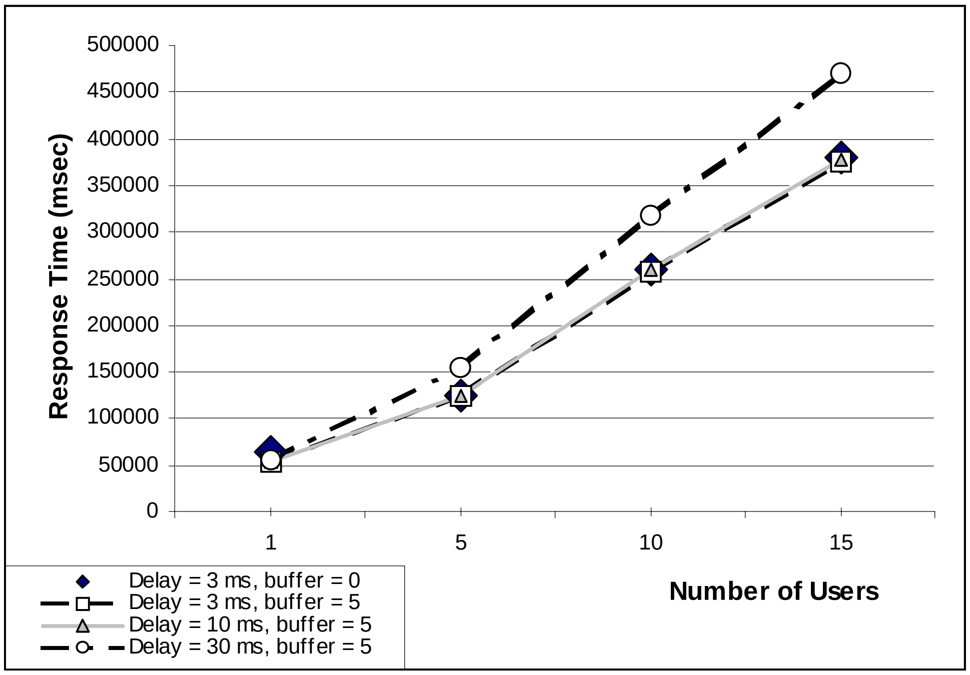

Figure 7: The Playout Time (msec) to the number of users with network latency values of 3 with no buffer and 3, 10 and 30 msec with 5 buffers

7. Conclusions

There are many benefits to the shift in viewpoint from prediction to budgets. Particularly, the choice of resource demand parameters is less difficult. In predictive modeling the step of estimating the demand for a particular operation is often very stressful for everyone involved. In budgeting the difficulty is shifted from the performance analyst, to the developer who must make the commitment. This is a clearer problem, and one which software developers feel more comfortable with.

The developer's goal after budgeting is not only focussed on just part of the system, it is also stated in terms (in terms of resource demands) which can be evaluated for one part in isolation, by measuring it in a test environment. This is critical to the separation of performance concerns.

The role of completions provides full information needed for modeling, while keeping the actual specification separate from the clutter of detail about the environment, re-used components, etc. It is important that the specifier does not have to fill the specification with this detail. The possibility of automation, and libraries of re-usable completions has been described, with filters for inserting completions for communications.

The case study illustrates the sequence of steps in the PASD process, and discusses the use of model sensitivity results to determine budget trade-offs, and the use of design adjustment to improve feasibility and compensate for sensitivity to an uncontrolled source of variation (the network latency).

Acknowledgement

This research was supported by CITO (Communications and Information Technology Ontario), NSERC (the Natural Sciences and Engineering Research Council of Canada), Mitel Networks Ltd. and Nortel Networks Ltd.

References

[1] I. Al-Fatah and S. Majumdar, "Performance Comparisons of Architectures for Client-Server Interactions in CORBA," in Proc. ICDCS '98 (Int. Conf on Distributed Computing Systems), May 1998, pp. 1-11

[2] I. Ahmad and S. Majumdar, "Achieving High Performance on CORBA-Based Systems with Limited Heterogenity," in *Proc. Int. Symp. on Real-time Object-Oriented Computing (ISORC2001),* Magdeburg, Germany, 2001.

[3] Buhr, R.J.A. and Casselman, R.S.: *Use Case Maps for Object-Oriented System*s, Prentice-Hall, USA, 1995.

[4] Buhr, R.J.A, "Use Case Maps as Architectural Entities for Complex Systems," *In: Transactions on Software Engineering, IEEE,* Vol. 24, No. 12, pp. 1131-1155, December 1998.

[5] H. El-Sayed, D. Cameron, C. M. Woodside, "Automated Performance Modeling from Scenarios and SDL Designs of Telecom Systems", in Proc. of the Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE98), Kyoto, April 1998.

[6] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, C.M. Woodside. "Performance Analysis of Distributed Server Systems". In Proceedings of the 6th International Conference on Software Quality, pp. 15-26, Ottawa, Canada, October 1996.

[7] G. Franks, A. Hubbard, S. Majumdar, J. Neilson, D.C. Petriu, J.A. Rolia and C.M. Woodside, "A Toolset for Performance Engineering and Software Design of Client-Server Systems", Performance Evaluation, October 1995.

[8] W. Halang and A. Stoyenko, Constructing Predictable Real-Time Systems. Amsterdam: Kluwer Academic Publishers, 1991.

[9] S. S. Hissam, G. A. Moreno, J. Stafford, and K. C. Walna, "Packaging Predictable Assembly with Prediction-Enabled Component Technology," Software Engineering Institute, Pittsburgh, CMU/SEI-2001-TR-024, 2001.

[10] C. Hrischuk, J. Rolia and C.M. Woodside, "Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype", Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 399-409, Durham, NC, January 1995.

[11] Jane W.S. Liu, Real-Time Systems, Prentice Hall, 2000.

[12] C. McNamara, Business Report on "Basic Guide to Non-Profit Financial Management", 1999

[13] OMG document "UML Profile for Schedulability, Performance, and Time", Revised submission, June 2001, available from OMG at www.omg.org.

[14] D. C. Petriu, J. E. Neilson, C. M. Woodside, and S. Majumdar, "Software Bottlenecking in Client-Server Systems and Rendezvous Networks," *IEEE Transactions on Software Engineering*, vol. 21, no. 9 pp. 776-782, September 1995.

[15] D. C. Petriu and H. Shen, "Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specifications," in Proc. 12th Int. Conf. on Modelling Tools and Techniques (TOOLS 2002), London, England, April 2002.

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

[17] 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.

[18] Smith, C.U., “Performance Engineering of Software Systems”, Addison-Wesley, 1990.

[19] Smith, C. U. & Williams, L. G., “Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software”, Addison-Wesley Publishing Company, 2001.

[20] C.M. Woodside, C. Hrischuk, B. Selic, S. Bayarov, "A Wideband Approach to Intergrating Performance Prediction into a Software Design Environment", Proc. First Int. Workshop on Software and Performance (WOSP98), pp. 31-41, October 1998.

[21] M. Woodside, D.B. Petriu, K. H. Siddiqui, "Performance-related Completions for Software Specifications", to appear in Proc. Int. Conf. on Software Engineering, Buenos Aires, May 2002.

[22] A. Hubbard, "SPEX: The Spftware Performance Experiment Driver," Real-time and Distributed Systems Lab, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, internal report, 1997.

[23] C. E. Hrischuk, C. M. Woodside, and J. A. Rolia, "Trace-Based Load Characterization for Generating Software Performance Models," *IEEE Trans. on Software Engineering*, vol. v 25, n 1, pp. 122-135, Jan 1999.

[24] OMG, “The Common Object Request Broker:Architecture and Specification”, edition 2.3, Dec. 1998.

[25] R. Jain, The Art of Computer Systems Performance Analysis. John Wiley & Sons Inc., 1991.

[26] K.H. Siddiqui, “Time/Performance Budgeting of Software Design”, M. Eng thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, Nov. 2001.