Performance Aware Software Development (PASD) Using Execution Time 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

August 01, 2001

Abstract

Performance Aware Software Development (PASD) considers performance issues and information from the earliest stages of development. PASD includes techniques for generating the information, including performance estimates, and techniques for guiding and coordinating the effort for development. The present work uses executions-demand budgets (also known as "real-time budgets") as a coordinating mechanism. Within an early architecture definition based on Use Case Maps (UCMs), demand budgets are allocated to responsibilities and verified by a semi-automated performance analysis using layered queuing models. The responsibility of the individual developer or group is simply to meet this budget, which can be tracked with unit tests of the code using widely available tools such as profilers. Budget changes are developed and verified at the overall architecture level, with the help of the performance model. The paper describes how the model is built and used. The key step is to add "completions" to the model, representing those parts of the system not defined in the software specification (such as COTS components, infrastructure such as middleware, the environment, and competing applications), which could impact the performance.

1. 0 Introduction

Budgets are a familiar approach to controlling the use of resources and to breaking a large problem down into manageable pieces; common examples are financial budgets, and time budgets for projects. This paper describes a budgeting approach to deal with time in software design. The approach

- breaks the problem into parts that correspond to system responsibilities, and to the work of separate developer groups,
- estimates (using a model) the overall performance that will result if all the budgets are met. The model estimates the effects of contention, overheads and latencies, which are outside the designers' control.

What this breakdown achieves is, to give developer targets, which he or she can understand and attempt to achieve, 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. The tools minimize the effort on a single project, since information about components and the execution environment can be stored, and re-used in later work.

The concept of a “balanced” budget is more subtle 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). Simulation techniques can be used to obtain any kind of measures, while analytical models often only give mean values. Preliminary analysis may be based on analytic results, and more careful analysis on simulations.

The analysis can go in a forward direction, from demands to delays, or in a reverse direction from permitted delays back to permitted values for demands. In the forward direction, demand budget values in terms of CPU seconds or network or storage operations are allocated to each activity (each responsibility in the UCM). The model estimates are computed and examined to see if they are consistent with requirements. If not, it may require iteration or search to find suitable budget values. In the reverse direction, a automated search can be carried out to find the budget values which satisfy the requirements.

It is essential to use a model because we are analyzing a system which has not yet been built. The challenge for usable modeling is to reduce the effort and cost of building a model, which is done here by model generation based on Use Case Maps (UCM) [5]. The generation process is mostly automated, apart from steps to define the various kinds of “completions”. These include descriptions of software components that are to be included “as is”, of protocols, middleware, file systems and operating systems overheads, of hardware properties, and of competing workloads that may impact the performance.

A separate challenge is to create the budget values for the demands of individual responsibilities. This process is potentially complex and is not addressed here. However, we believe it can be done in the same way that financial budget values are created under uncertainty, using experience and a reserve for contingencies. To assist the budgeting, prototyping experiments can be done to refine the estimates, 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, it can be accommodated by taking time from other responsibilities, changing the design, or boosting the hardware capability.

In previous research, there has been frequent mention of budgeting of time in software, particularly in designing for hard-real-time deadlines. In these systems, schedulability analysis plays the role of the model; to verify that timing constraints can be met deterministically (see, e.g. [6]). For soft real-time requirements, the discussion has been rather general and solid techniques are scarce.

Previously, the creation of a performance model from a design specification was addressed in [1], based on prototyping in a CASE tool (Objectime), and tracing scenarios executed by the tool. In that work, the environment parameters were measured in advance by calibration experiments on the run-time environment, stored in a repository, and generated into the model automatically to some degree. This work is different in that it starts from designer intentions expressed in the UCMs, and that it addresses budgeting issues in soft-real-time systems.

2. 0 Budget Analysis Road Map

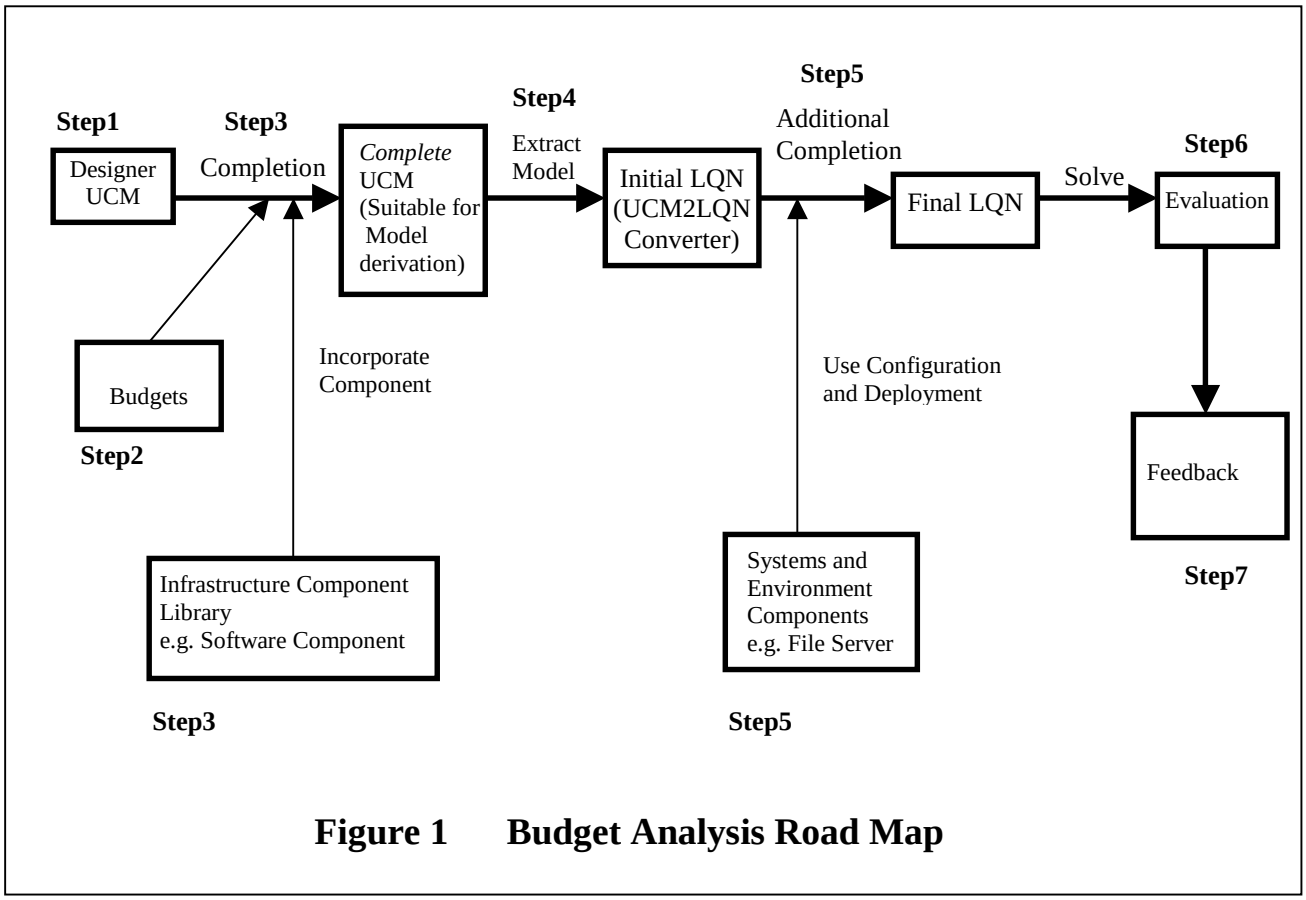

Figure 1 Budget Analysis Road Map

Figure 1 illustrates the process for analyzing budgets. The boxes represent design artifacts or libraries used in the analysis, while the arrows show operations carried out either by the analysis, or by some automated transformation. There are seven steps.

Step1: Designer UCM

The designer UCM is a document which captures the system specification as a set of scenarios, at a suitable level of abstraction, and defines software components and responsibilities. The use of UCMs is described in [5].

A UCM is not the only possible starting point for PASD. Another form of specification which identifies the scenarios, the components, and the activities for which code is to be developed, could be used. For instance an executable state-machine specification in ObjecTime was considered in [8] and (using SDL) in [3]. It might be possible to use a UML specification, and a profile for UML has recently been defined to support such an analysis [9]. 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 [7].

The designer UCM may or may not define the components and infrastructure that will be included in the final system; these are the subjects of the “completions” below.

Step2: 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. The performance engineering analysis is based on assumed budget values, which must be guessed or derived by the design team. There are various possibilities, including deriving the budgets from the performance requirements themselves (dividing up the allowed delay into parts allocated to different responsibilities).

Step3: Completion at the UCM level

The software design normally does not fully define the system to be deployed. There may even be 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. The missing detail can be supplied at the UCM level, if it arises at particular points in the scenario, or at the LQN (performance model) level, if it describes a service or infrastructure operation that is used by the system. Often a given "completion" can be added at either level; there is no hard rule.

UCM "completions" are taken from a library of UCM specifications and are added as stubs to the designer UCM. A stub represents an operation with detail defined in a hidden sub-map. Thus the analyst must explicitly indicate where they are to be added. In future it might be possible to add these stubs automatically guided by rules or preferences input by architecture team.

Step4: Create the Performance Model

The fourth step of road map is LQN model extraction from the Complete UCM. The LQN model used here is a generalization of well-known queuing models, with layered features for describing software behavior.

Layered Queuing Network (LQN) Model

In a Layered Queuing Network (LQN) model, the resources can be the usual hardware devices, tasks that provide services to other tasks, buffers, critical sections, and other resources or modules. These resources are called *"tasks"*. System behavior is represented in terms of requests for service between tasks and the queuing of messages at tasks [4]. An LQN can represent a queuing model as special case. In an LQN, tasks can accept service request messages at an LQN *entry*. An entry describes and models a service provided by an actor, and also the associated resource demand. If a task is a kind of object, an entry is a kind of method. Requests for service are characterize as entry-to-entry interactions sent in a synchronous (i.e. send-wait-reply), or asynchronous (i.e. send-and-continue) fashion. The interaction types affect performance because they affect task blocking, queuing delays at devices and messages queues, parallelism, and resource contention. The unit of concurrency for a LQN model is a scheduled concurrent operating system process or thread, and is referred to as a task.

The LQN analytical model gives throughput and response times. Tools, which exist to make the analysis easier, are the LQNS solver for LQNs, described in [4], and the SPEX program [2] for sequencing repeated runs over ranges of parameter values.

An automated generative tool UCM2LQN Converter for creating performance models from UCMs is used [7]. The resulting LQN model is normally incomplete from the point of view of the expected execution environment.

Step5: Completion of the LQN Model

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

LQN Completion Filter

Distributed and communication software involve different kinds of network components, with their own latency and delay characteristics, and a structure which must be added to the LQN model. Such a component may include multiple tasks, which can be single or multiple, local or remote, single or multi threaded, depending on the communication types and systems. To insert network components that handle a call between tasks, we use the *LQN Completion Filter* tool. The inserted elements are bound to the source and destination tasks of the call, and either to existing processors or to new devices of their own.

The LQN Completion Filter has three variations

- Single Network Component Task (SNCT), which inserts a single network component to handle a call,
- Multiple Network Components Tasks (MNCT), which inserts a subsystem, and
- Infinitely Threaded Server Task (ITST), which just inserts a latency (as an infinitely threaded pseudo-task to represent the delay).

Step6: 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.

Step7: Feedback

Feedback depends on the evaluation results. If they are not satisfactory then some changes are required. If predictions show inadequate performance, on 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.

3.0 Example: Distributed Hand-Off Protocol UCM

Step1: Designer UCM

Figure 2 shows a Designer UCM for a distributed hand-off protocol, based on a tutorial example by Gunter Mussbacher at Mitel Networks which illustrates a particular style of drawing UCMs, to teach designers the UCM notation. The original UCM example has eight paths, one of which has been simplified and is shown in Figure 2.

The protocol described in this specification is used to coordinate the execution of some arbitrary duty by two tasks, Process_A and Process_B, in a distributed environment.

In Figure 2, Party_A initiates the execution of the duty. which is forwarded to Process_A via Device_A and Main_Controller, with a confirmation to Party_A. Later, Party_A initiates a request to hand off the duty to Process_B, which goes via Device_A and the Main_Controller to Process_A, and then to Process_B. Process_B interrogates Party_B. Party_B forwards a confirmation via Device_B and Main_Controller to Process_B, which then begins to execute the duty. While doing so it sends a synchronous storage request to Process_Backup, which carries out the request and replies back to Process_B.

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 tasks together, it needs to be refined to a point where it is executed by one component, before continuing.

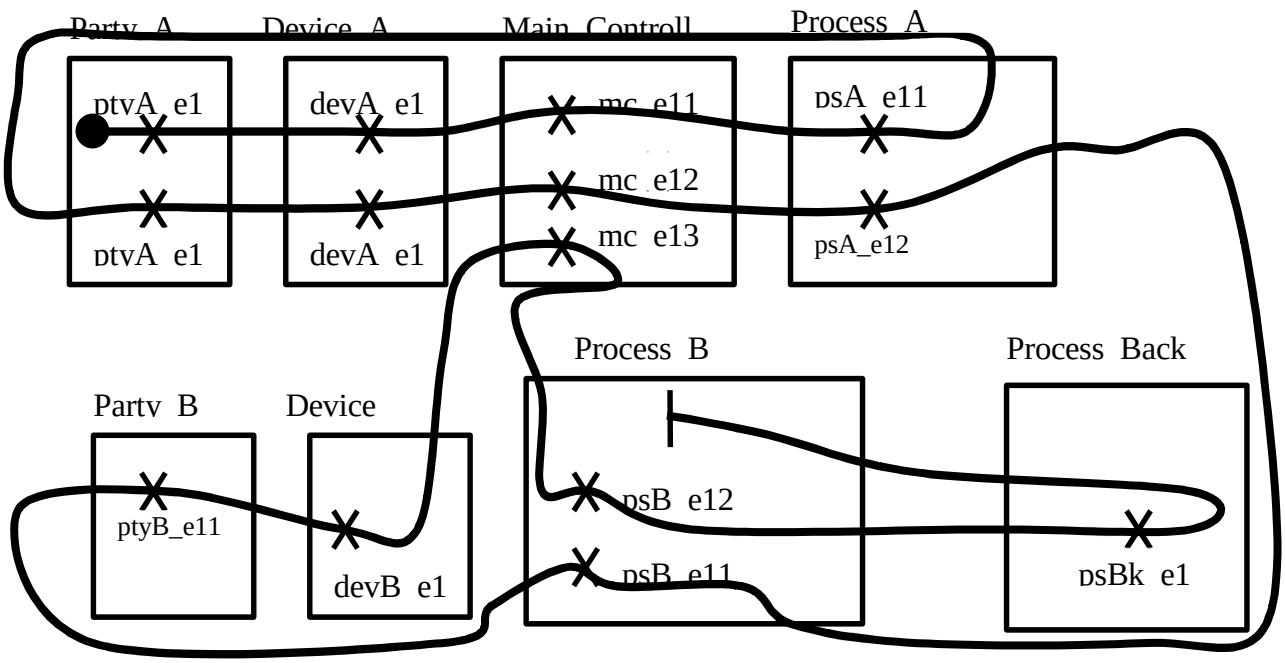

Figure 2 Distributed Hand-Off Protocol UCM (Designer UCM)

Step 2: Budgets

Because the details of budgeting are beyond the scope of this paper, the CPU time demand of each responsibility is chosen as 1.0. This is sufficient to illustrate the steps. Since components used in this example are new and unfamiliar types, therefore, sensitivity of system response time and throughput are considered as targets to budget CPU time demand for each responsibility.

Step 3: UCM Completion and Infrastructure Components

The Designer UCM shown in Figure 2 does not describe the communication infrastructure between different communicating processes. Communication infrastructure might consist of protocol stacks, middleware, etc. Here, it was decided to use a tuple space [10] communication mechanism. Other communications such as protocol stacks are needed but will not be shown here, for simplicity of explanation. The tuple space communications is provided by a server task, which handles and stores messages.

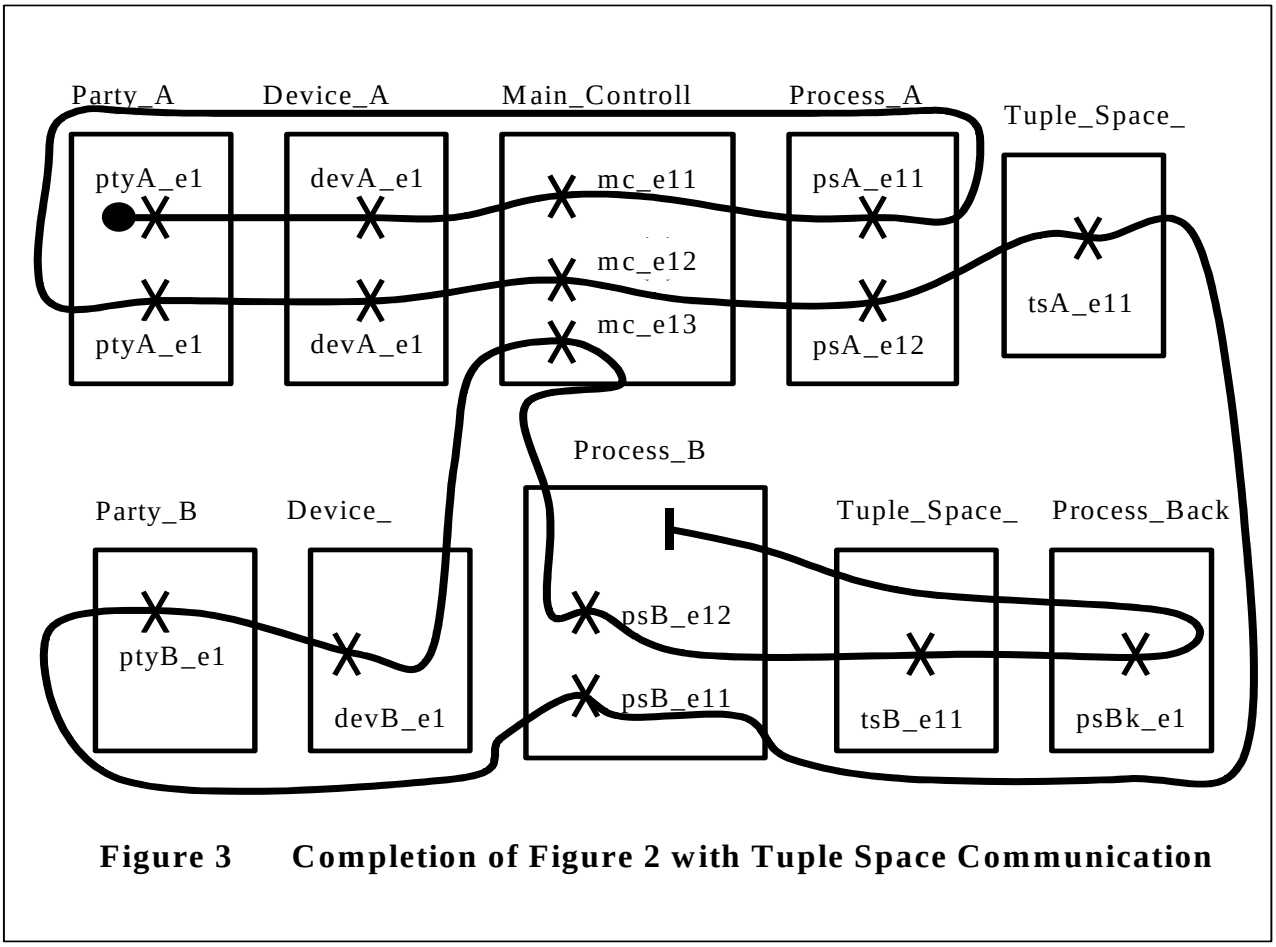

Figure 3 Completion of Figure 2 with Tuple Space Communication

Tuple Spaces

A tuple space provides an anonymous communication among the different processes, based on "tuples" which are ordered sequences of data or "messages". Tuple space communication provides flexibility and scalability. Flexibility does not restrict format of tuples nor types of data contained. Scalability provides facility of anonymity of tuple operations. A basic tuple space provides four operations: out, in, rd and eval. Many implementations support two new operators, "inp" and "rdp", which are non-blocking versions of "in" and "rd". In this example, it is assumed that communication between Process_A and Process_B, and Process_B and Process_Backup is anonymous via the tuple space, while all other the processes are using address based socket communications.

Figure 3 shows a distributed tuple space server with two server processes. The process Tuple_Space_A provides communications between Process_A and Process_B, Tuple_Space_B process provides communication between Process_B and Process_Backup. The scenario illustrates that Process_A performs an "out" operation and places data in Tuple_Space_A, and Process_B performs an "in" operation which removes it. Process_B and Party_B forward this request to Device_B. Device_B replies back to Main_Controller, which sends asynchronous message to Process_B. Process_B performs "out" operation and places tuple in Tuple_Space_B. Process_Backup removes tuple from Tuple_Space_B by performing "in" operation, it performs a backup operation and replies back to Process_B through Tuple_Space_B.

Measurements have been done on a commercially available tuple space called J-Spaces. J-Spaces Technologies Limited releases J-Spaces as The J-Spaces Platform 1.0. The J-Spaces Platform 1.0 is the first commercial implementation of the JavaSpaces specification. JavaSpaces is a powerful Jini service specification. The time measured for "out" and "in" operations are about 47 milliseconds (47 cpu tick) each.

Step 4: LQN Model of Distributed Hand-off Protocol

The LQN model of Distributed Hand-off Protocol Complete UCM was generated by the LQN2UCM converter. This converter maps every task of UCM to an LQN task, every crossing to an entry, and every responsibility to an LQN activity. Figure 4 shows the model, with many activities suppressed. In Figure 4, every rectangle represents a task, with a gray color box showing the task name. White boxes besides this are the entries of the task. Boxes shown in a second row of the task are activities. The entry which contains these activities, is just above the first activity from the left. The messaging arrows with a filled head represent synchronous service calls while the other arrows represent asynchronous service calls. The dashed line arrows with a filled head represent forwarded service calls.

The Figure includes two new tasks, Party_ A_User and tsCriticalSection, which have been introduced as "additional completions" which will be described in next step.

Figure 4 shows that the scenario begins with:

1. the Party_A_User task initiates the scenario by making a synchronous request through entry ptyA_User to entry ptyA_tae of the Party_A task.
2. The activity PtyA_e11 handles this request and makes a synchronous request to entry dev_e11 of the Device_A task.

3. This request is forwarded by mc_e11 entry of Main_Controller to psA_e11 entry of Process_A, which replies back to ptyA_tae entry. Eventually, Party_A task replies back to Party_A_User task.

The scenario continues as Party_A task makes an asynchronous request to Device_A through ptyA_e12 activity. devA_e12 makes an asynchronous request to mc_tae entry of Main_Controller. This request is handled by mc_e12 activity of mc_tae entry. It makes a synchronous request to psA_e12 entry of Process_A task. Tuple_Space_A, Process_B, and Party_B forward this request to Device_B, which replies back to Main_Controller.

Finally Main_Controller initiates an asynchronous request through mc_e13 activity to psB_e12 entry of Process_B task, which requests Process_Backup for back up operation through Tuple_Space_B task. Process_Backup performs a backup operation and replies back to Prcoess_B.

Step 5: Additional Completions, Environment and Assumptions

"Additional completions" are introduced in the form of environment components i.e. Party_ A_User and tsCriticalSection tasks. Both tasks are added as to achieve suitable performance model. Party_A_User task, which is a reference task, will initiate the scenario. tsCriticalSection task, which is acting as centralized server for distributed tuple space servers, will provide critical section for Tuple_Space_A and Tuple_Space_B. It will synchronize both tuple space servers.

- To follow the analysis a little further, the performance model was created with workload values, and the following assumptions:
- The CPU demands were considered as unity for all entries and activities in the beginning
- The probabilities for the path traversing were considered as 100 % (i.e. value '1' from one entry of a task to entry of another task).
- All the requests for in and out operations of tuple space are handled by one entry.
- All the tasks are on separate processors.

Step 6: Experimental Results Evaluation

Figure 5 shows the model predictions for the delay of the system, giving the sensitivity of response time vs. different number of users for the "full demand" or base case (labeled FDMC, with the value '1' for all demands) and for other budget values. The number of threads of the Main Controller, denoted as m, was also varied with values 1, 2, 6 and 10. Thus, for instance, FDMC(m=1) is a base case with single Main_Controller thread and CPU demands of 1.0 for all the entries and activities.

Other values of demands were modeled with 10 threads (m = 10):

- the MC(0.05, 0.15, 0.15) curve has demands 0.05 for entry mc_e11, 0.15 for activity mc_e12, and 0.15 for activity mc_e13,
- the DVD(0.1, 0.1) curve is for both devices having demands 0.1,
- the PABD (0.1, 0.1) curve shows the improvement obtainable by reducing the demands for Process_A and Process_B entries to 0.1.

the PAB (m = user) curve is for multiple threads in Process_A and Process_B, equal to the number of users)

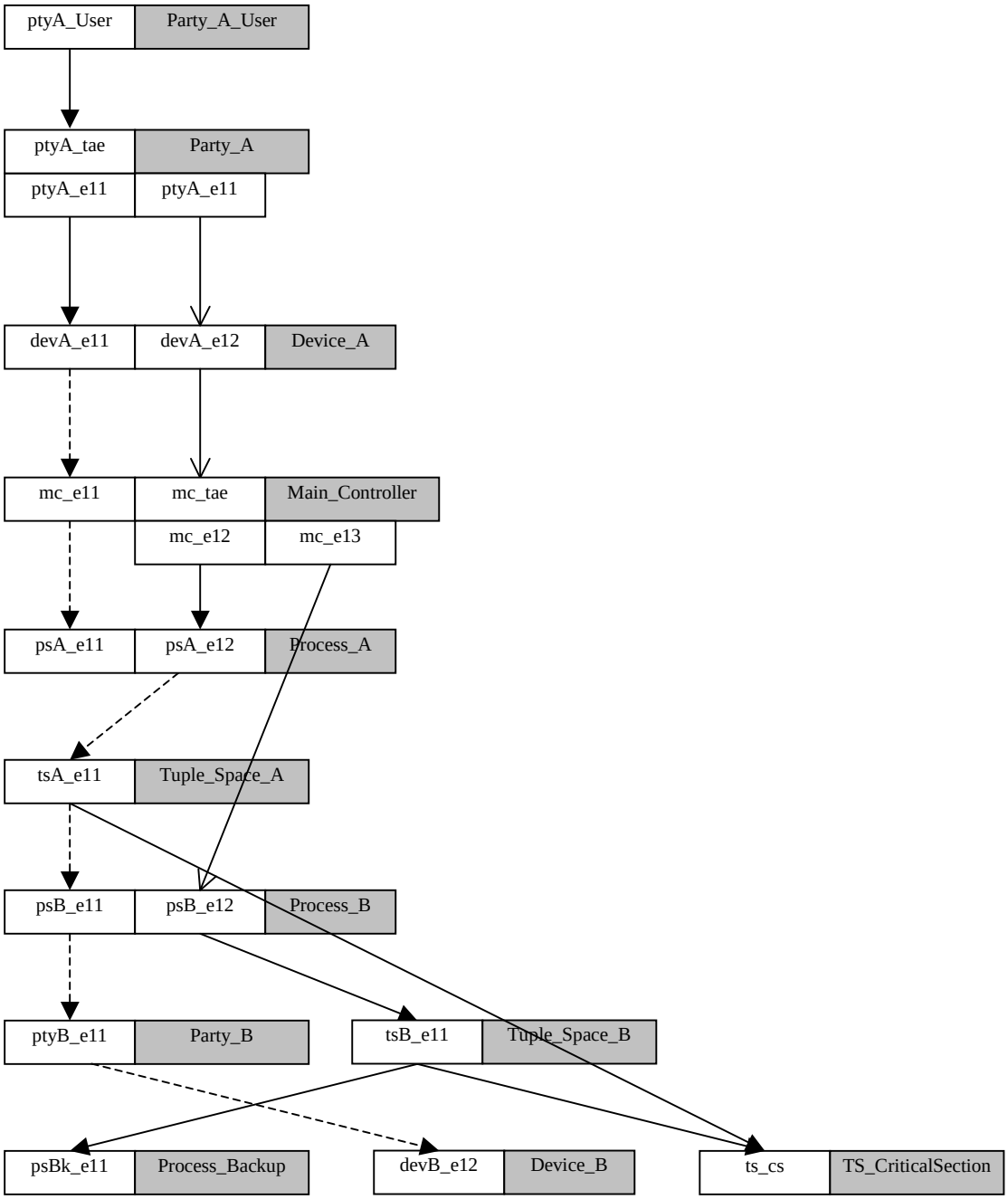

Figure 4 Distributed Hand-off Protocol Layered Queuing Network (LQN) Model

From the graphs we can see that the performance starts to degrade around 90 simultaneously active users, for m = 1, but that with 10 threads in the Main Controller this is improved to about 120 users. Detailed inspection of the model outputs confirm that the Main Controller is a software bottleneck, which requires multi-threading, but the improvement is slight beyond m=10. Main controller demand reduction has only slight effect; device demand reduction has a modest effect; Process A and B demand reductions have a large effect, and Process A and B multithreading has a very large effect, roughly doubling the capacity when m=10 for the Main Controller.

The sensitivity of the throughput was also analyzed and shows the effect on system capacity of changes in numbers of users, in Figure 6. We can see that throughput saturates at the point where response times begin to rise, for each curve. The same data is shown in numerical form in Tables 1 and 2.

4.0 Summary

A general process for budgeting CPU times for software operations has been described and illustrated. The results show how a performance model can be generated automatically from a high-level architectural view, and how the results can be interpreted to identify sensitive points in the design.

It is a small step to generalize the process described here to also budget the number of requests made to a system service, as well as CPU demands.

The example of a distributed hand-off protocol also illustrated the concepts of model completion at the UCM level and the LQN level, with a tuple space server inserted in the UCM, and a critical section controller inserted in the LQN. In many cases it is a matter of convenience, at which level a completion is provided. The example did not describe details of creating the initial budget values, and this is an important area for further development. Another interesting issue, not investigated here, is the process of adjusting the budget values when they are unsatisfactory.

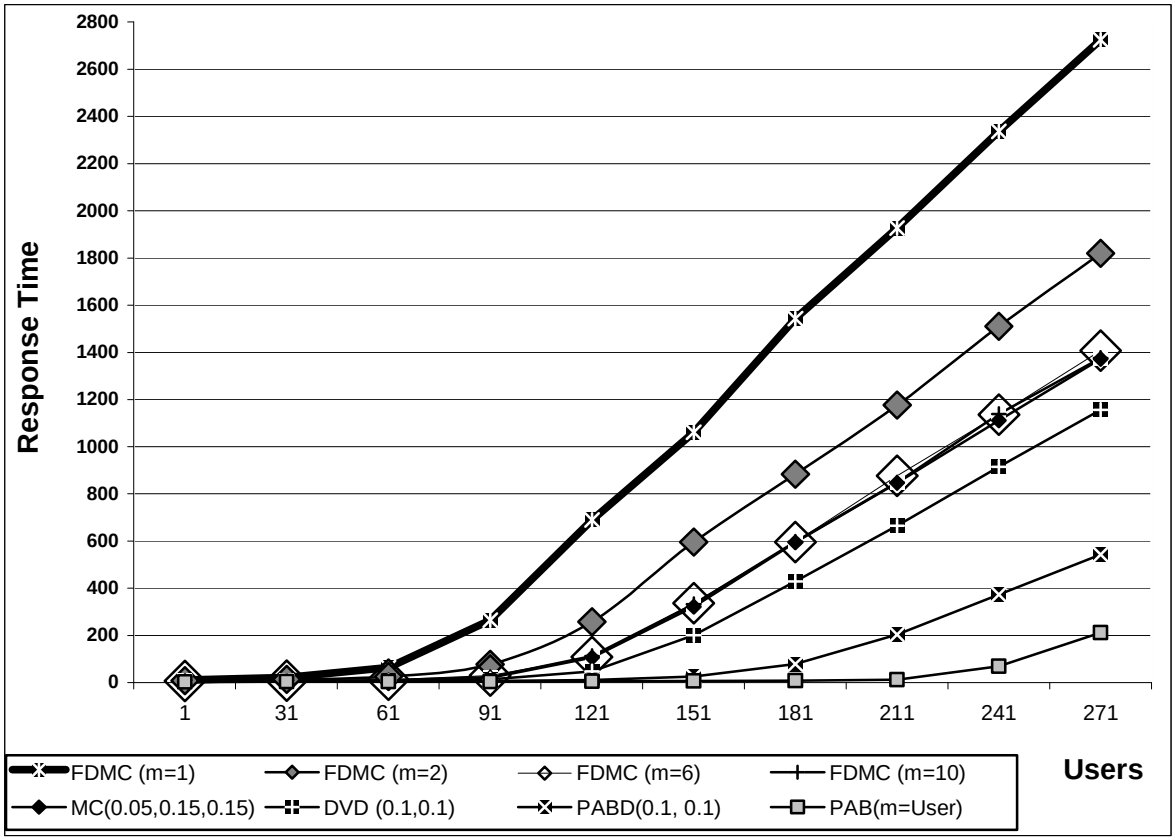

Figure 5 Sensitivity of Response Time vs. No of Users

User	FDMC (m=2)	FDMC (m=6)	FDMC (m=10)	MCD (0.05,0.15,0.15)	DVD (0.1,0.1)	PABD (0.1, 0.1)	PAB (m=User)
1	10.518	7.936	6.825	6.388	5.610	4.417	3.122
31	21.951	12.976	8.021	7.090	5.993	4.928	3.650
61	64.308	25.569	12.600	9.670	8.805	6.948	4.314
91	263.789	77.915	29.326	22.374	19.993	13.359	6.144
121	689.314	258.030	108.101	112.250	105.530	47.920	11.100
151	1061.060	594.883	336.250	331.773	321.029	200.679	25.578
181	1542.470	882.566	594.886	597.021	593.844	428.956	78.589
211	1927.360	1177.000	875.489	847.206	846.439	664.890	202.736
241	2336.310	1509.870	1136.060	1138.340	1112.520	914.523	372.236

Table 1: Mean Delays

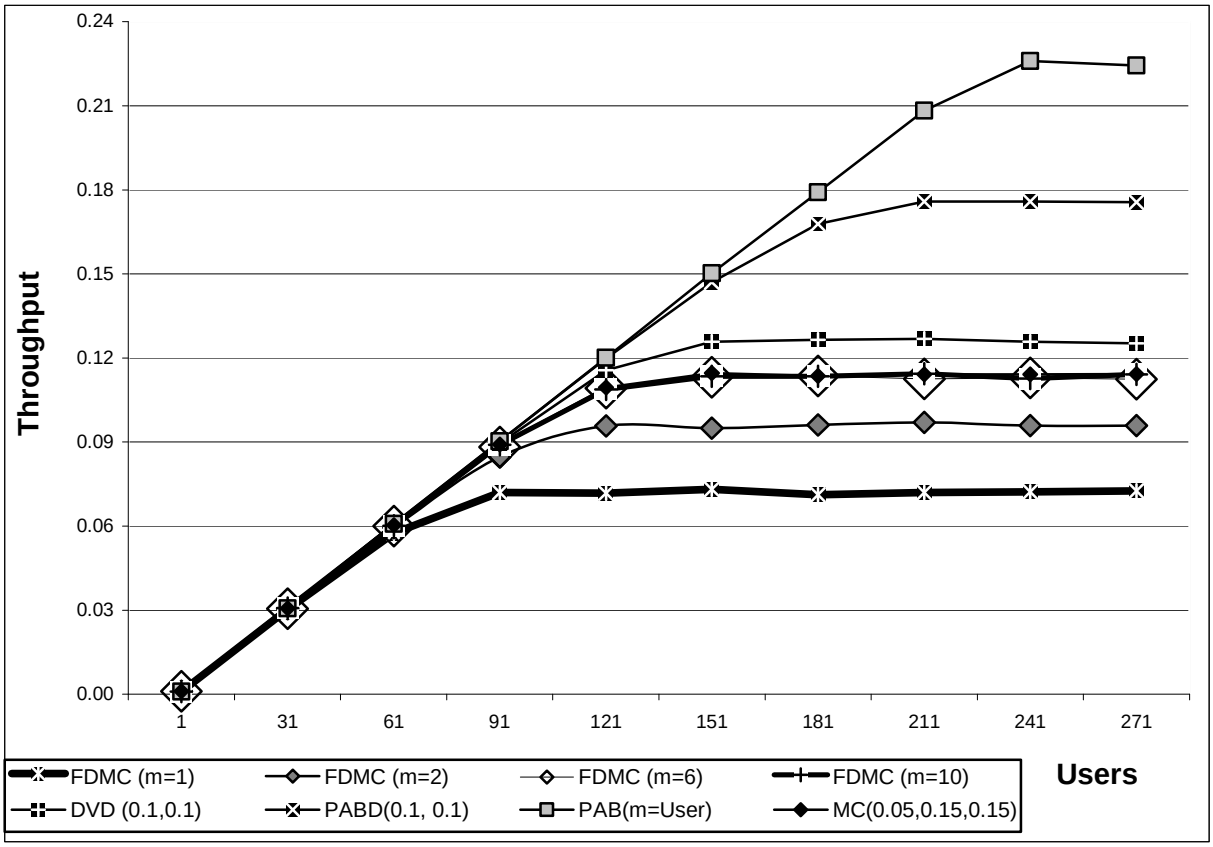

Figure 6 Sensitivity of Throughputs vs. No of Users

User	FDMC (m=2)	FDMC (m=6)	FDMC (m=10)	MCD (0.05,0.15,0.15)	DVD (0.1,0.1)	PABD (0.1, 0.1)	PAB (m=User)
1	0.0010	0.0010	0.0010	0.0009	0.0010	0.0010	0.0010
31	0.0304	0.0305	0.0306	0.0308	0.0307	0.0307	0.0308
61	0.0575	0.0595	0.0601	0.0602	0.0604	0.0606	0.0610
91	0.0720	0.0847	0.0883	0.0890	0.0891	0.0896	0.0903
121	0.0718	0.0957	0.1092	0.1088	0.1094	0.1156	0.1198
151	0.0732	0.0949	0.1130	0.1134	0.1146	0.1258	0.1470
181	0.0712	0.0960	0.1135	0.1135	0.1136	0.1265	0.1678
211	0.0721	0.0970	0.1126	0.1143	0.1141	0.1268	0.1759
241	0.0722	0.0959	0.1129	0.1127	0.1141	0.1258	0.1759

Table 2: Throughputs

References

[1] 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.
[2] 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.
[3] 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.
[4] 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.
[5] R.J.A. Buhr and R.S. Casselman, “Use Case Maps for Object-Oriented Systems”, Prentice Hall, Inc., 1996.
[6] W. Halang and A. Stoyenko, Constructing Predictable Real-Time Systems. Amsterdam: Kluwer Academic Publishers, 1991.
[7] D. B. Pertiu, “Layered Software Performance Models Constructed from Use Case Map Specifications”, M. Eng. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2001.
[8] 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.
[9] OMG document "UML Profile for Schedulability, Performance, and Time", Revised submission, June 2001, available from OMG at www.omg.org.
[10] S. Kawai and S. Matsuoka., “Using Tuple Space Communication in Distributed Object Oriented Languages”, Proc. of the OOPSLA '88, pages 276-283, 1988.