Performance Engineering with the UML Profile for Schedulability, Performance and Time: a Case Study

Andrew J. Bennett and A. J. Field
Department of Computing
Imperial College London
United Kingdom
ajb@doc.ic.ac.uk

Abstract

We describe the application of a performance engineering methodology based on UML diagrams with annotations taken from the Profile for Schedulability, Performance and Time. The methodology targets the early stages of the development process and works exclusively with system scenarios. These scenarios are mechanically translated into the stochastic process algebra FSP and are analysed using existing tools to study both the behavioural and performance properties of the system. A case study of a mobile telecommunications billing system is used to assess the effectiveness of the approach. The results show that our methodology is effective at detecting, quantifying and locating performance bottlenecks. A set of guidelines for resolving performance problems is devised and used with performance analysis results to drive a series of design changes until the performance requirements of the system have been met.

1. Introduction

The resolution of performance problems found during system testing can result in significant redesign and reworking, causing schedule delays, cost overruns and lost market opportunity. The system architecture may need to be replaced or, in the worst case, it may not be possible to build a system that meets the performance requirements at all. There is clearly a need for tools and techniques that enable performance analysis of designs to be done easily, quickly and reliably throughout the development process, in particular during the early stages of architectural and sub-system design.

This paper is a case study. It explores the application of a UML-based performance engineering methodology to the early design stages of a telecommunications billing system centred on a main-memory (i.e. not disk-resident) relational database accessed by a multiprocessor shared-memory server.

The methodology uses the profile for Schedulability, Performance and Time (SPT) to augment UML sequence diagrams with performance information. The UML diagrams are translated into the stochastic process algebra FSP [8] and subsequently analysed by discrete-event simulation using an established tool called LTSA. In a production environment we would not expect the user to see the FSP code or the LTSA tool – everything would be controlled from the UML interface, including UML/SPT input and the analysis of the simulation output. We therefore do not discuss FSP, LTSA, or the UML to FSP translation scheme in detail in this paper – the interested reader is referred to [3].

The case study is representative of a real system for mobile phone billing and is inspired by [2]. To focus the investigation we consider the top-level architecture of the system which becomes the hardest to change as the system development progresses. It is also where some of the most significant performance bottlenecks are likely to be present. The same methodology can, of course, be applied to the lower-level components in turn. From the engineering perspective, the aim is to produce a design that meets specified performance targets, at least at this top-most level, as early as possible.

The paper has two main objectives. Firstly, we wish to evaluate the UML/SPT methodology with respect to the case study. In particular, we want to show how it can be used to detect, quantify and locate performance problems in the application, and to successively refine the application design so that it meets its performance targets. At the same time, however, we are interested in the factors affecting the performance of systems of this type *per se*. In particular, we are interested in identifying commonly-observed performance "antipatterns" in the application and in identifying solutions to those problems as a set of performance engineering guidelines. We do not expect these guidelines to be new, but it is interesting to extract them in this example as they might be usefully incoporated into future tools aimed at supporting the methodology.

The case study is described after some background on UML and the SPT profile and a brief overview of the underlying process algebra analysis framework. We end with some general discussions and concluding remarks.

2. Performance modelling with UML

The UML diagrams that provide the key information required for performance analysis are those that describe behaviour and resources:

- Sequence or activity diagrams can be used to express those scenarios that have performance requirements.
- Statechart diagrams describe the behaviour of objects, and the time required to respond to stimuli.
- Deployment diagrams define how objects are mapped on to processing resources.

We have used sequence diagrams because they are popular and easily understood. However we believe the approach can be applied to any behavioural specification in UML. For simplicity, it has been assumed that each active object (i.e. process or thread) has been assigned to its own processing resource (i.e. processor), obviating the need for deployment diagrams. Statechart diagrams have also not been used – instead we have taken the approach that the system is defined solely by sequence diagrams, and we mechanically generate state machines for each class, expressing them in FSP. The implications of this approach are assessed in the discussion section.

The idea of constructing performance models from UML is by no means new. Previous work has proposed customised performance annotations and/or extensions to the standard UML. Statecharts, often in conjunction with other UML diagrams, have been used in various performance modelling studies, for example [4], which concerns deriving a PEPA (another stochastic process algebra) model from UML, [9] which has similar objectives but which targets stochastic Petri nets, and [7] which combines statecharts, general time delays and probabilistic choice to build stochastic automata that can be analysed by simulation.

The advantages of scenario-based formalisms (such as sequence diagrams and activity diagrams) from the user's perspective have been highlighted by various authors, e.g. [12]. Sequence diagrams have been explored extensively for describing functional properties of systems, e.g. [13], and more recently for the extraction of performance models by translating performance-annotated sequence diagrams into layered queueing network (LQN) models – LQN is an approximate hierarchical queueing model that can be solved either analytically or by discrete-event simulation [11].

2.1. The SPT profile

The SPT Profile [10] allows UML diagrams to be annotated with performance information. It consists of three sub-profiles, but only the performance analysis sub-profile is used in this work.

A simple example of a sequence diagram with SPT profile performance annotations is shown in Figure 1. The `<<PAcontext>>` stereotype indicates that this diagram is a scenario involving some resources (software objects in this case) driven by a workload. The objects are a server and a mutex lock. The heavy box of the server object indicates that it is an active object that can execute concurrently with any other active objects, although none exist in this example. The annotation on the lifeline of the user object has a `<<PAopenLoad>>` stereotype indicating that it is a workload, i.e. it defines the intensity of the demand made on the system by the users of this scenario. The text following the stereotype consists of pairs of tag definitions and values – the meaning of each is defined by the standard. In this example, the interval between requests is defined by an exponential distribution with a mean value of 40 ms. A requirement that the mean response time is 100 ms is given, along with a placeholder variable `$Resp` for the predicted value that will be determined by solution of the performance model. The request operation made to the server has an open arrowhead, indicating that it is an asynchronous request, and there is therefore an implicit queue of requests at the server object; the server repeatedly takes an item from the head of the queue and processes it. Each request requires the mutex to be locked and unlocked – each mutex operation takes 10 ms on average, also exponentially distributed. Some local processing is done between the two lock operations, taking a mean of 20 ms.

The first study of a performance engineering methodology based on UML, the SPT profile and an LQN model generator and solver was presented in [14]. Our approach is inspired by this work, although we remark that the process algebra framework offers some additional advantages. For example, it can simultaneously be used to allow functional properties, such as freedom from deadlock and livelock, to be verified. This is not a significant issue in this paper – we focus here exclusively on performance – although it could well be in the wider context of the case study. The reader is referred to [3] for further details.

3. FSP and the LTSA

It is not necessary to understand FSP for the purposes of this paper, however we remark that each object in a sequence diagram can be translated into an FSP process (effectively a description of a state machine). Each object is

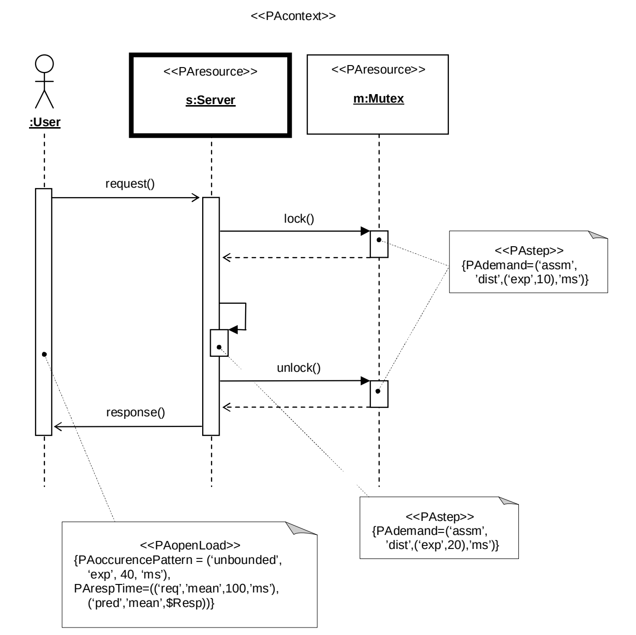


Figure 1. A simple scenario.

translated individually into an FSP process, and the resulting processes are *composed* at the topmost level.

In the example, the user object represents an open workload, so there is implicitly a queue of user jobs requiring access to the server – this queue must be explicitly represented in FSP. The mutex can be thought of as a library class with unspecified internal behaviour but with an interface comprising `lock` and `unlock` calls. It is quite feasible to specify the lock behaviour itself in UML, but this is not addressed here.

The translation, which is mechanical but not currently automated, inserts a system-oriented performance measure for each resource (objects with `PAresource` annotations and implicit queues in the sequence diagrams). A performance timer process is similarly added to each workload process to record response time distributions. Measure and timer values are output at the end of simulation runs, and are used to aid the understanding of the performance of the system, as will be seen later.

For more details of the FSP language the reader is referred to [8]. The stochastic extensions are described in [1], and an example of UML to FSP translation is given in [3].

Functional and performance analyses can be applied to FSP specifications by using the LTSA tool [8]. Functional analysis involves analysis of the underlying labeled transition system (LTS) for the FSP program. This describes the set of states that the system can be in and the transitions between those states. In this sense the LTS is very similar to a UML statechart for the entire system. Importantly, here this is *derived* from the UML sequence diagrams rather than being supplied in addition to them. The LTSA tool provides built-in model checking that can determine whether deadlock and livelock can occur in any possible execution of the system. Various performance analysis tools are available – in this paper the performance models are solved by LTSA using discrete-event simulation, although other solution methods can be used.

4. Case study: debit-based billing

The problem studied is how to provide a billing system for a debit-based (i.e. "pay as you go") mobile phone service. It is inspired by the system outlined in [2].

Customers top up their accounts using vouchers that are bought for cash. When a call is made, the base station nearest the user contacts the billing system to check that the user is valid and to determine how much credit is held in the user's account. The call is allowed to proceed if the account is in credit. The base station maintains the cost of the call, terminating the call if the cost exceeds the available credit. When the call ends, either by a user hanging up or by forced termination, the base station instructs the billing system to reduce the balance in the user's account by the final cost of the call.

Strict performance requirements exist so that the billing system does not slow down the process of making and terminating calls. The time taken to authenticate a user and retrieve their balance must not exceed 70 ms, and the time taken to update the account balance must not exceed 90 ms.

Some loss of data during recovery from failures (such as loss of power or a software fault) is acceptable.

Calls are made via the base station every 20 ms on average. It is assumed that it takes 10 ms to read or write data records from memory and 100 ms to read or write a block of data from disk. For the purposes of this exercise all times are assumed to be exponentially distributed although a variety of distributions are supported by FSP and LTSA.

4.1. System overview

The proposed solution centres on a database comprising a single table that holds information on each customer, including their current balance. A conventional relational database management system holds data on disk, but this is too slow for this application – disk access time is 100 ms, greater than the balance retrieval requirement of 70 ms. The solution adopted is to use a main-memory database that stores all table data in memory for fast retrieval and modification [5]. Such a database can also be accessed concurrently on a shared-memory multiprocessor machine, allowing several table reads to be performed at the same time. However, writes must be serialised to ensure consistency of the stored data. A read-write lock is used to control access to the table because it allows multiple readers or a single writer to hold the lock. The use of a multiprocessor machine with up to four processors will be explored in the performance analysis later on.

The key disadvantage of a main-memory database when compared to a conventional database is resilience – the contents of memory will be lost if the host on which the database is running crashes, or if the database program itself crashes due to a software defect. Such data loss would result in a loss of revenue to the mobile network operator. This problem is addressed by periodically writing changes to the table out to disk since disk-based data will not be lost in the event of a crash (disks can also fail, but such failures are much more rare and are ignored here). Particular care must be taken to minimise disk usage since disk accesses are so much slower than memory accesses – an order of magnitude slower in this case study. This issue is addressed in detail in the performance analysis later in this paper.

4.2. Design 1: the basic system

The first attempt at a design for the system consists of three classes of object:

- Application: the billing server.
- RWLock: the read-write lock to protect the database, allowing concurrent read accesses.
- Disk: the passive disk store.

The database itself is not represented by a class because it is embedded in each instance of the application (it is a shared-memory segment that is mapped into the address space of each application process). There are two scenarios with performance requirements: open call authenticates the user and retrieves their current balance, and close call updates the user's balance.

4.2.1. Sequence diagrams for design 1. The sequence diagram for the open call scenario is shown in Figure 2. The application waits for the next open call request from a base station; it then acquires the lock, accesses the data from memory (shown by the `<<PAStep>>`), releases the lock, and returns control to the user. The workload performance annotation shows the required response time of the operation, with a placeholder for the predicted result that will be found by simulation. The number of instances of the application class is shown by the `PAcapacity` annotation – it is a variable that will be assigned several different values during performance analysis. From our earlier assumptions about process mapping the `PAcapacity` is therefore synonymous with the number of physical processors at the server.

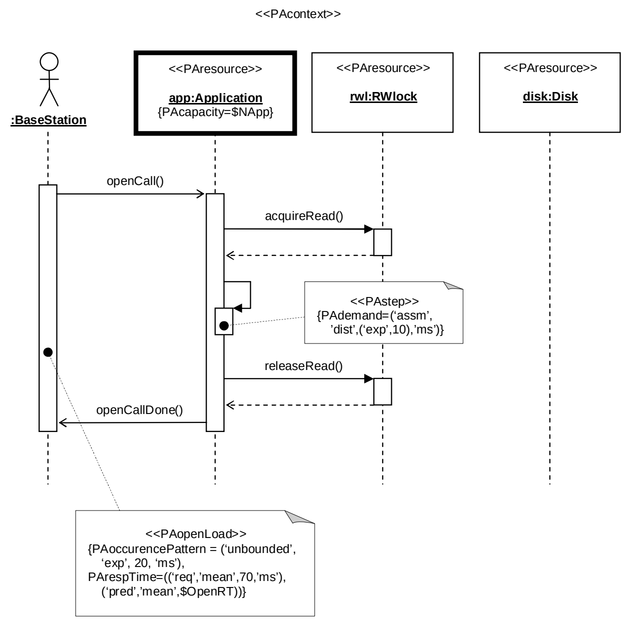


Figure 2. Open call scenario for design 1.

The sequence diagram for the close call scenario is shown in Figure 3. On receiving a close call request, the application requests and acquires the lock for writing – a fair read-write lock requires both a `requestWrite` and `acquireWrite` operation [8]. The data is then modified in memory, and the modified data is also written to disk to ensure that the update is not lost in the event of a failure. Finally, the lock can be released, and the user notified of completion. The exact form of the data written to disk is not clear from the diagram – the disk file envisaged is an exact copy of the table held in memory and the write updates the file so that it remains a faithful copy of the memory-based table. The file will be loaded into memory when the system needs to recover after a failure.

4.2.2. Performance analysis of design 1. The translation scheme can be used to produce an FSP representation of the billing system. The first task is to test for functional problems (i.e. deadlock and livelock) – none are present. Results of a set of simulation experiments conducted using LTSA to determine the response times of the two scenarios and object utilisations for a varying number of instances of the application class are shown in Table 1. Confidence intervals are not shown but in all cases in this paper the half width of the 90% confidence interval is less than 5% of the mean. Where resources are replicated the reported utilisations are averaged over all instances.

Each sequence diagram has one performance requirement, and these results show that neither is met regardless of the number of application instances used. Studying the ob-

	Mean response times (ms)		Utilisations			Mean queue lengths	
NApp	open call	close call	app	disk	rwlock	open call	close call
1	493.50	601.03	0.16	0.84	1.00	3.89	3.99
2	499.10	625.67	0.17	0.85	1.00	3.87	3.96
3	569.58	687.96	0.17	0.85	1.00	3.91	3.94
4	614.27	778.21	0.17	0.86	1.00	3.90	3.93

Table 1. Simulated performance of design 1.

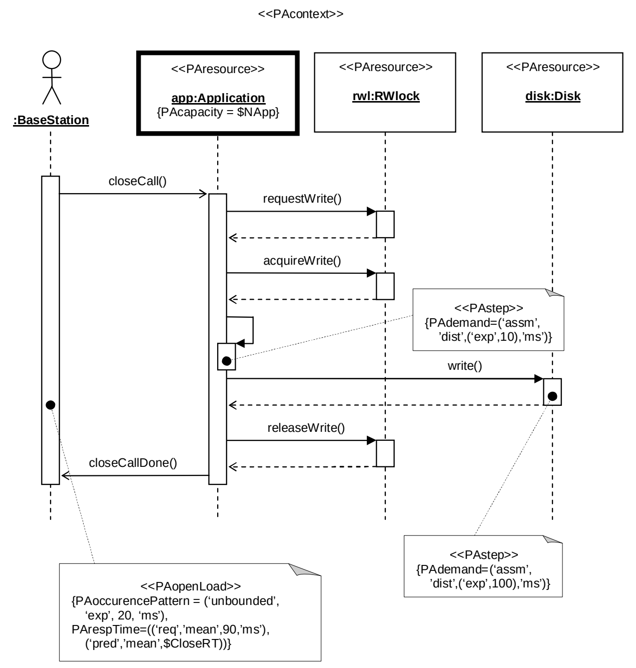


Figure 3. Close call scenario for design 1.

ject utilisation figures with a single processor (i.e. NApp = 1) indicates that the lock is saturated. The capacities of the workload queues were set to 4 for these experiments, and the mean lengths found are close to this, indicating that the system is backing up, preventing new jobs from entering the system. Increasing the number of instances of the application will not improve performance as it is not the bottleneck. The application is mostly idle (typically about 17% utilisation) whereas the disk is very busy (typically about 85%).

4.3. Design 2: the buffered system

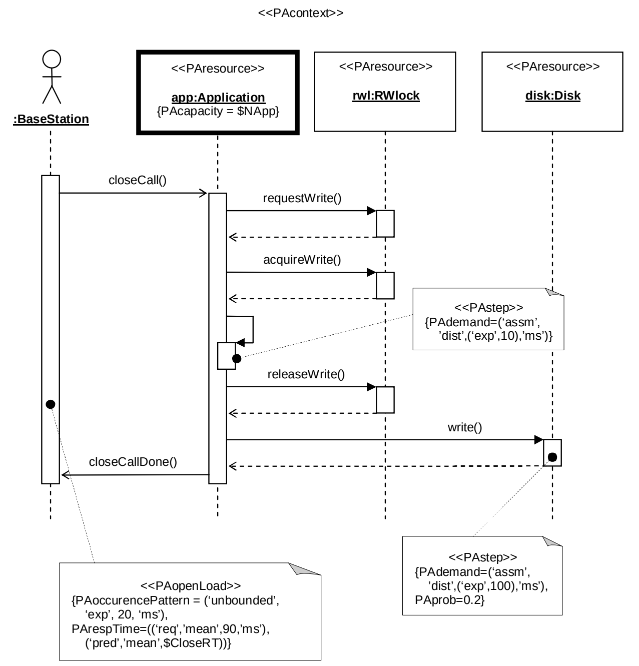
A design change is needed to overcome the problem of the saturated lock. Lock utilisation can be reduced by removing work from the critical section in each scenario so that the lock is held for shorter periods of time. The database accesses must be made when the lock is held in order to guarantee correctness, and so the removal of the disk access from the critical section of the close call scenario must be investigated. However, the disk already shows high utilisation, so it too could easily become a bottleneck.)Disk utilisation could be reduced if multiple instances of it were used – this potential solution has not been investigated here because it has been assumed that the file resides on a single disk.)

Instead of writing to the disk in the critical section, a record of the data change made in the close call scenario is written to an in-memory buffer; the contents of the buffer are written to disk after the lock has been released. Due to the high disk utilisation observed in design 1, data changes from several consecutive invocations of the close call scenario are buffered together and the buffer contents are only written to disk when the buffer becomes full.

It is now assumed that the buffer contains descriptions of the database records that have been changed, and that these are appended to a log file on disk. In this way, a set of change records can be written with a single disk write. Entries in the log are "replayed" when the system recovers after a failure in order to rebuild the state of the database to that immediately before the failure. Such a log file would grow in size until it fills the disk – for this reason a "checkpoint" of the table is made periodically, in which a copy of the entire table in memory is written out to another file on the disk. Recovery now involves loading the most recent checkpoint file and replaying the log entries made since that checkpoint. This allows the log file to be deleted periodically, preventing it from filling the disk, albeit at the expense of another process writing to the disk that is already potentially a performance bottleneck in the system. Recovery in main-memory databases is described in detail in [6].

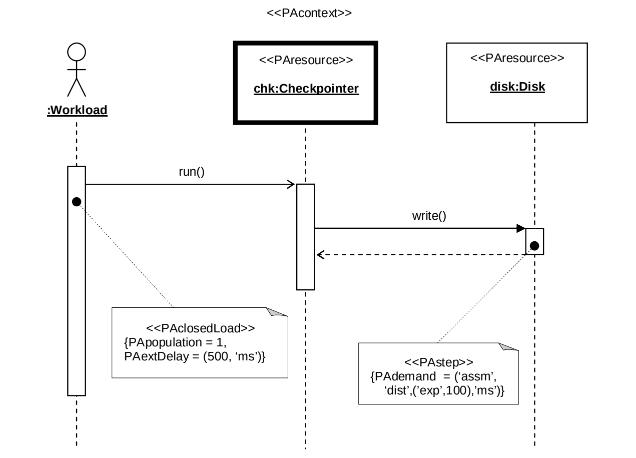
This design change has altered the data integrity properties of the system: the buffer entries will be lost if the system fails, resulting in a loss of revenue to the network operator. The potential loss would need to be assessed in a real system. A more sophisticated buffering scheme could be adopted in which the buffer is written to disk if the monetary value of buffered entries exceeds some threshold, but this is not considered here.

4.3.1. Sequence diagrams for design 2. The revised sequence diagram for the close call scenario is shown in Fig-



<<PAcontext>>
:BaseStation
<<PAresource>> app:Application {PAcapacity = $NApp}
<<PAresource>> rwl:RWlock
<<PAresource>> disk:Disk
closeCall()
requestWrite()
acquireWrite()
<<PAstep>> {PAdemand=('assm', 'dist',('exp',10),'ms')}
releaseWrite()
write()
closeCallDone()
<<PAstep>> {PAdemand=('assm', 'dist',('exp',100),'ms'), PAprob=0.2}
<<PAopenLoad>> {PAoccurencePattern = ('unbounded', 'exp', 20, 'ms'), PArespTime=(('req','mean',90,'ms'), ('pred','mean',$CloseRT))}

Figure 4. Close call scenario for design 2.



<<PAcontext>>
:Workload
<<PAresource>> chk:Checkpointer
<<PAresource>> disk:Disk
run()
write()
<<PAclosedLoad>> {PApopulation = 1, PAextDelay = (500, 'ms')}
<<PAstep>> {PAdemand = ('assm', 'dist',('exp',100),'ms')}

Figure 5. Checkpoint scenario for design 2.

ure 4. It is not necessary to show the buffers because they are not allocated and freed dynamically. It has been assumed that the buffer can hold five update entries, and therefore the disk write operation has an annotation to indicate that it is only invoked with a probability of 20%. The creation of the checkpoint file is represented by the checkpoint scenario – the sequence diagram for this is shown in Figure 5. It uses a closed workload to represent the finite amount of work that must be done in each time period. It is assumed that one disk write is made every 500 ms, meaning that checkpointing utilises 20% of the disk device. The open call sequence diagram is unchanged from design 1 and is not shown again.

4.3.2. Performance analysis of design 2. The simulation results for design 2 are shown in Table 2. These results show better mean response times than design 1, but still neither response time requirement is met. Studying the results for a single instance of the application (i.e. NApp = 1) shows that the lock is no longer a bottleneck. The disk is also much less busy than in design 1, although the effect of the reduced number of disk accesses made in each execution of the close call scenario is largely offset by the increased rate at which the scenario is executed, indicated by higher application utilisation. In addition, disk accesses are made by the checkpointer. Two of the workload queues are close to their capacities, indicating that the system is starting to back up.

It is natural that increasing the number of application instances leads to improving response times now that the lock is no longer a bottleneck – this is demonstrated by the simulated open call response time with 4 processors being less than half that of the 1 processor value.

The response times are still not good enough to meet their requirements, despite the significant improvements over design 1.

4.4. Design 3: the decoupled system

The disk is the most heavily used object in design 2: it is accessed by the checkpointer as well as the application. It is inevitable that an application will sometimes have to wait for the disk to become available in the close call scenario, thereby reducing application utilisation and system throughput.

This problem can be addressed by a second design change: when the buffer is full it can be passed to another active object, the logger, that is responsible for writing the contents of the buffer to the log file on disk. This allows the application to finish handling the close call request before the disk write has been made. Extra concurrency is introduced into the system by decoupling the action of writing the buffer to disk from the close call operation.

4.4.1. Sequence diagrams for design 3. The sequence diagram for the revised close call scenario is shown in Figure 6. The open call and checkpoint sequence diagrams are unchanged from design 2 and are not shown again. The system is configured with a finite number of buffers that are allocated and freed on behalf of clients by a buffer manager. The buffer manager must be modelled in this design because buffers are finite resources that are allocated and freed dynamically by the pool of application instances. An active object is used to represent the logger – it receives buffers

NApp	Mean response times (ms)		Utilisations			Mean queue lengths		
	open call	close call	app	disk	rwlock	open call	close call	checkpoint
1	156.58	180.53	0.44	0.66	0.44	3.28	3.34	0.15
2	105.51	146.54	0.58	0.79	0.55	2.72	2.79	0.26
3	90.67	144.55	0.66	0.86	0.61	2.45	2.43	0.43
4	74.01	141.42	0.72	0.90	0.66	2.10	2.13	0.56

Table 2. Simulated performance of design 2.

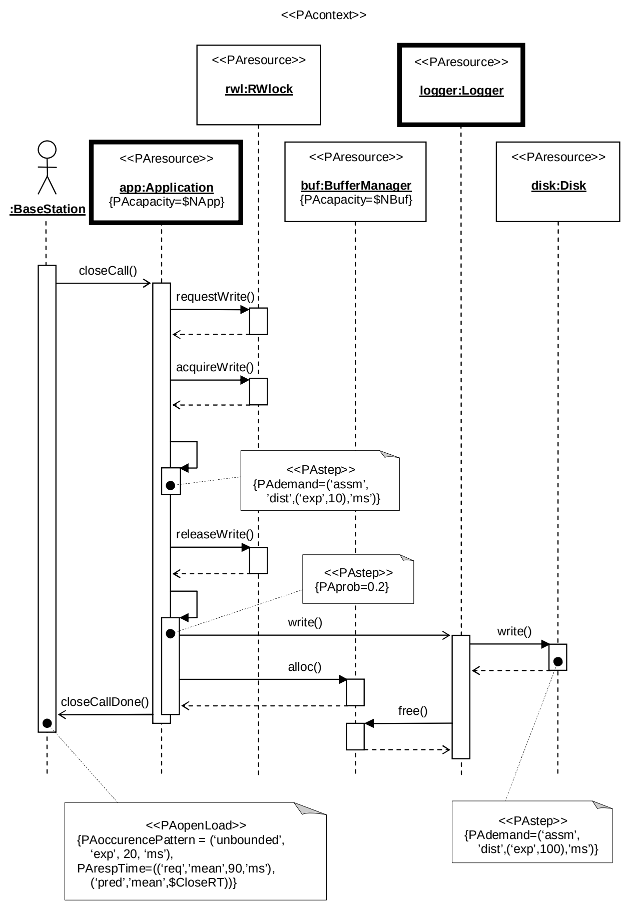


Figure 6. Close call scenario for design 3.

from application instances by means of asynchronous calls, and writes their contents to disk before freeing them.

This formulation of the system does not accurately represent the state of the system at start-up when each instance of the application class needs to allocate a buffer. As a workaround the FSP process for the application class can be modified manually to include an initialisation phase that allocates the first buffer. The required functionality could instead be specified using a statechart for the application with a boolean value that represents whether a buffer has been allocated – we return to this issue in the discussion section.

4.4.2. Performance analysis of design 3. The results of simulation runs of design 3 are shown in Table 3. The objective here is to determine whether it is possible to meet the open and close call response time requirements, and so the number of buffers is set to a value large enough that the buffer manager never becomes a bottleneck, i.e. there is always at least one available buffer throughout the simulation. A value of 10 was found to be adequate.

The open call response time requirement is achieved with two or more application instances, whereas the close call response time requirement is achieved with both one and two application instances, but no more. That is, the only configuration that achieves both requirements is two applications. Removing the disk write from the application allows it to handle open and close call requests at a faster rate, thereby increasing the utilisation of the application and the lock. However, as the number of application instances increases the close call response time worsens. This is the only design that has shown this behaviour. It is caused by the read-write lock allowing multiple concurrent readers, which in turn causes write requests to wait a longer time on average for the set of readers to release the lock.

5. Guidelines for performance improvement

In applying the methodology we have systematically employed the following rules of thumb for improving performance:

1. If a resource is heavily used, add more instances of the resource. (This was considered for the disk in design 2, but was rejected as inappropriate in that case.)
2. If a resource is heavily used, reduce the demand that the system makes on it. (Multiple changes are buffered together to reduce the number of disk accesses in design 2.)
3. Remove work from critical sections to reduce the time busy locks are held – this will tend to reduce serialisation and increase utilisation. (The disk write was moved out of the critical section in design 2.)

	Mean response times (ms)		Utilisations				Mean queue lengths			
NApp	open call	close call	app	disk	rwlock	buffer	open call	close call	checkpoint	logger
1	77.82	83.45	0.73	0.92	0.73	4.29	2.40	2.44	0.16	2.39
2	67.32	88.38	0.74	0.94	0.68	5.72	2.00	2.04	0.22	2.80
3	66.73	101.96	0.77	0.95	0.69	7.03	1.90	1.91	0.18	3.09
4	64.23	116.02	0.76	0.97	0.69	8.17	1.79	1.81	0.17	3.21

Table 3. Simulated performance of design 3.

4. If a server uses a shared resource that suffers from high utilisation, add another active object to access the shared resource, allowing the server to finish the request and start a new one earlier. (The full buffer is passed from the application to a new active object, the logger, in design 3.)

These guidelines are widely applicable although they are by no means new. A suggestion very similar to 1, and another somewhat like 4 are set out in [14]. Guidelines 2 and 3 appear obvious, but using them with simulation results allows design changes to be made in a targeted way. This raises the possibility that the guidelines could be implemented in a modelling tool so that suggestions for design changes are automatically presented to the modeller. Engineering judgement would still be required to assess whether changes are appropriate to the system, and to assess their impact on other properties of the system such as the cost of data loss in the event of failure introduced in design 2.

6. Discussion

We have successfully used FSP models that are mechanically derived from UML sequence diagrams to detect performance problems in a case study system. The case study did, however, expose a shortcoming in the translation scheme: one FSP process had to be manually augmented with a call to allocate the first buffer when it initialises. Adding statechart diagrams to the translation scheme would resolve this issue, albeit at the expense of greater complexity. An interesting issue is how this extra complexity could be shielded from the modeller. A UML tool could provide the ability to add attributes to objects in sequence diagrams, and operation invocations could be guarded by expressions over those attributes, thereby preventing the modeller from having to express the state of objects using statecharts. A related issue is how statecharts could be included in the translation later in the development process when they are available – this appears to be feasible since FSP processes are effectively state machines. This emphasises that UML diagrams form the only model that needs to be maintained during system development – the performance model can be rebuilt whenever a UML diagram is changed, and consistency problems caused by having to manually maintain both performance and functional models are eliminated.

Deployment diagrams are not used in the translation scheme for simplicity, but again it appears straightforward to include them. A processing node, for example, could be modelled as a set of processors – a processor is merely a resource that is acquired and released, and is easily represented in FSP. We are particularly interested in including these diagrams in our translation scheme because changing the deployment of objects in a distributed system is another approach to resolving performance problems – the approaches used in the case study set out above were changing configuration (e.g. adding more instances of the application) and changing design (e.g. adding the logger to access the busy shared disk).

7. Summary, conclusions and further work

We have studied the effectiveness of using performance models expressed in the stochastic process algebra FSP that are mechanically generated from UML sequence diagrams to detect, quantify and locate performance problems in a case study system. A debit-based mobile phone billing system was used as the case study, with sequence diagrams produced from a free-text description of the system. One FSP process required manual editing due to a limitation in the translation scheme, but otherwise all FSP programs showed no functional (i.e. deadlock and livelock) issues.

Results for the first design showed that the system did not meet its performance requirements. FSP utilisation measures generated for each object were used to locate the root cause of the performance problem, allowing a sequence of targeted design changes to be made until the performance requirements were met.

This work has highlighted several areas that we wish to pursue in the future. First, we are interested in automating the UML to FSP translation – this would enable us to study larger systems more easily. Secondly, although our approach was successful for the case study, its effectiveness needs to be assessed for other systems. Finally, the guidelines for performance improvement raise some intriguing issues, such as whether they are effective for other, more complex systems, whether it is possible to mechanise them, and how to present design modification suggestions to the modeller in a clear way.

Acknowledgments

This work was partially funded by the U.K. Engineering and Physical Sciences Research Council through grant GR/R02238/01. The authors would like to thank Murray Woodside for many fruitful discussions on the case study and subsequent performance analysis.

References

[1] T. Ayles, A. J. Field, J. Magee, and A. J. Bennett. Adding performance evaluation to the LTSA tool. Technical report, Department of Computing, Imperial College London, Mar. 2003.

[2] G. D. Baulier, S. M. Blott, H. F. Korth, and A. Silberschatz. Sunrise: a real-time event-processing system. *Bell Labs Technical Journal*, 3(1):3–18, 1998.

[3] A. J. Bennett, A. J. Field, and C. M. Woodside. Experimental evaluation of the UML profile for schedulability, performance and time. In *UML 2004, Lisbon, Portugal*, 2004.

[4] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. In *18th UK Performance Engineering Workshop, Glasgow, Scotland*, 2002.

[5] H. Garcia-Molina and K. Salem. Main memory database systems: an overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, 1992.

[6] H. V. Jagadish, A. Silberschatz, and S. Sudarshan. Recovering from main-memory lapses. In *Proceedings of the 19th International Conference on Very Large Data Bases, Dublin, Ireland*, pages 391–404. Morgan Kaufmann, 1993.

[7] D. N. Jansen, H. Hermanns, and J.-P. Katoen. A QoS-oriented extension of UML statecharts. In *UML 2003: The Unified Modeling Language, Modeling Languages and Applications, Lecture Notes in Computer Science, volume 2863*, pages 76–91. Springer-Verlag, Berlin, 2003.

[8] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, Chichester, England, 1999.

[9] J. Merseguer, J. Campos, S. Bernardi, and S. Donatelli. A compositional semantics for UML state machines aimed at performance evaluation. In *6th International Workshop on Discrete Event Systems*, pages 295–302, 2002.

[10] Object Management Group. UML profile for schedulability, performance and time specification, Mar. 2002.

[11] D. Petriu and M. Woodside. Software performance models from system scenarios in use case maps. In *Computer Performance Evaluation, Modelling Techniques and Tools: Twelfth International Conference (TOOLS 2002), Lecture Notes in Computer Science, volume 2324*, pages 141–158. Springer-Verlag, Berlin, 2002.

[12] C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, Reading MA, 1990.

[13] S. Uchitel, R. Chatley, J. Kramer, and J. Magee. LTSA-MSC: tool support for behaviour model elaboration using implied scenarios. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Warsaw*, 2003.

[14] J. Xu, M. Woodside, and D. Petriu. Performance analysis of a software design using the UML profile for schedulability, performance and time. In *Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003), Lecture Notes in Computer Science, volume 2794*, pages 291–310. Springer-Verlag, Berlin, 2003.