Performance Evaluation of CORBA Concurrency Control Service Using Stochastic Petri Nets

Roberta A. A. Fagundes[1]

Paulo R. M. Maciel[1]

Nelson S. Rosa[1]

Abstract. The interest in performance evaluation of middleware systems is increasing. Measurement techniques are still predominant among those used to carry out performance evaluation. However, performance models are currently being defined due to their flexibility, precision and facilities to carry out capacity planning activities. This paper presents stochastic Petri net models for performance evaluation of the CORBA Concurrency Control Service (CCS), which mediates concurrent access to objects. In order to validate the proposed models, CCS performance results obtained using those models are then compared against ones obtained through actual measurements.

1 Introduction

Middleware is a class of software especially designed to enable the communication between distributed applications, in such way that it hides, from application developers, the complexity of underlying network mechanisms and the heterogeneity of environments where the application runs [26]. Along with those two basic characteristics, middleware systems also provide services, adding value to the communication. In a typical use of those services, a client/server programmer should focus on business aspects and leave to the middleware services the responsibility of treating with the complexity of security, transactional, concurrency and fault tolerance aspects.

Unlike traditional performance areas such as computer network and database management, performance evaluation of middleware systems is currently on an initial stage. While initiatives such as TPC (Transaction Processing Council) [15] and network simulators are widely known in their respective areas, middleware evaluation is currently defining basic metrics, benchmarks and still almost solely adopting measurement as the unique performance evaluation technique. In practical terms, most performance evaluation of middleware concentrate on measuring and comparing the performance of existing products, and may be summarised in three activities: to build an application that sends thousands of messages using

1 Universidade Federal de Pernambuco, Centro de Informática
Caixa Postal 7851 50740-540 – Recife – PE – Brazil
+55 81 2126 8430
{raaf, prmm, nsr}@cin.ufpe.br

the middleware, to measure the time taken by the middleware to carry the message from sender to receiver, and average the measured time.

While measurement is an important and necessary technique, it is the most time consuming of the three techniques (measurement, simulation, analytical modelling) for performance evaluation, especially along with the case of being able to change configurations and scenarios. In particular, middleware performance evaluation suffers of proper difficulties such as complex instrumentation, reduced number of tools, great diversity of operations and computation models, lack of a global clock, state and control. Furthermore, as current middleware systems are black box (application developers do not see details of the middleware), measurement refers to the execution time of coarse grain middleware mechanisms and everything below it, i.e., the virtual machine (e.g., Java or C#), the operating system (e.g., Windows or Linux) and network layers (e.g., TCP or UDP). Besides, while performance of an existing system can be evaluated by measurements, such approach is not possible during its design phases. In this case, one may resort to model based evaluation techniques.

Many modelling formalisms have been used for performance evaluation of communicating systems. Such modelling formalisms should provide a description of performance aspects related to the modelled systems. Some modelling formalisms provide graphical representation of systems such as Queuing networks and stochastic Petri nets [9, 5, 6, 2, 8, 7]. Petri nets may also be applied to verification and/or qualitative analysis of systems properties, such as deadlock freedom, liveness and boundedness [2].

This paper concentrates on the definition of Petri net models used to performance evaluation of the CORBA Concurrency Control Service (CCS). Stochastic Petri nets are a family of performance models that naturally represents concurrency, synchronization and resource sharing aspects. These models allow steady state and transient evaluation of systems and such evaluation may be carried out either applying numerical methods or based on stochastic simulation [4,5,6].

This paper is organized as follows: Section 2 introduces basic concepts of Petri nets and CCS. Next, Section 3 presents the proposed Petri net models of CCS. Simulation results along with comparison with measurements of OpenORB CCS are then presented in Section 4. Section 5 presents related work to the proposed approach. Finally, Section 6 presents the conclusions future work.

2 Basic Concepts

Prior to present the proposed Petri net models of the CCS, we introduce basic concepts of CCS and both Petri nets. As the service being modelled is the CCS, it is initially presented followed by the main definitions of Generalised Stochastic Petri Nets (GSPN) and Deterministic Stochastic Petri Nets (DSPN).

2.1 CORBA Concurrency Control Service (CCS)

CORBA (Common Object Request Broker Architecture) [17] is an architecture specifically designed to support the development of distributed applications in heterogeneous environments. Essentially, it provides the communication infrastructure that allows objects to make requests to remote ones in a transparent way, i.e., wherever the remote object is executing, no matter which implementation language and operating system is being used.

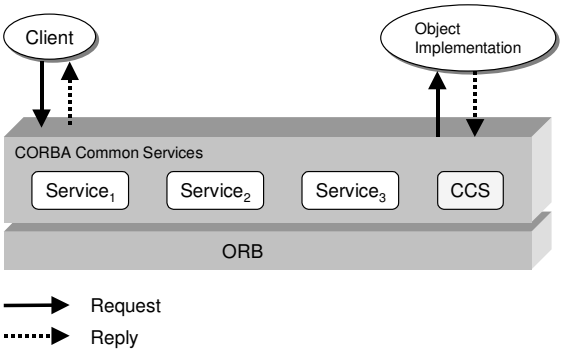


Figure 1 – Simple Overview of CORBA Architecture

Figure 1 shows a simple overview of CORBA and how distributed applications are built on it. A Client makes a request (invocation) to a remote object (Object Implementation) that implements a service. After being processed, the object implementation sends back a response to the client. The request passes through CORBA Common services and is forwarded to the Object Implementation via the ORB (Object Request Broker). The ORB works as a communication bus responsible for the transportation of the request from the client host to the object implementation in a transparent way.

The CORBA Common Services (COS) includes several different services such as concurrency, transaction, fault-tolerance, naming, trader, and so on. These services are usually specified as a set of APIs described in IDL (Interface Description Language). In particular, the Concurrency Control Service (CCS for short) [17] allows multiple clients to coordinate their access to shared resources. Coordinating access to a resource means that when multiple, concurrent clients access a single resource, any conflicting actions by the clients are reconciled so that the resource remains in a consistent state.

CCS does not define what a resource is. It is up to the clients of the Concurrency Control Service to define resources and to properly identify potentially conflicting uses of those resources. In a typical use, an object would be a resource, and the object implementation would use the concurrency control service to coordinate concurrent access to the object by multiple clients [21].

In CCS, the coordination of access to shared resources is carried out through the use of locks, which work as permissions for a client to access a particular resource. The CCS specification defines five lock modes: read, write, upgrade, intention read and intention write. The user of CCS may acquire a lock in two different ways: on behalf of a transaction or on behalf of the current thread. In the first way, the transaction service is responsible for releasing the lock (transactional client), whilst in the second one the current threads (non-transactional clients) do this. A user acquires a lock for a particular resource if the required mode does not conflict with the lock mode of a concurrent user.

CCS is defined through four main interfaces: LockCoordinator, LockSet, TransactionalLockSet and LockSetFactory. In particular, we are mainly interested in the `LockSet` that is explicitly used by nontransactional clients and implicitly by transactional ones. This interface provides five operations, as defined in the following OMG IDL specification:

```
Interface LockSet{
     void lock(in lock_mode mode);
     boolean try_lock(in lock_mode mode);
     void unlock(in lock_mode mode) raises(LockNotHeld);
     void change_mode(in lock_mode held_mode, in
                                        lock_mode new_mode)
raises(LockNotHeld);
     LockCoordinator get_coordinator(in CosTransactions::Coordinator
which);};
```

The operation `lock` acquires a lock in specified mode. In the case that the lock mode is incompatible with one of the concurrent clients the operation will block the current thread until the lock is acquired. The operation `try_lock` is used to acquire a lock and return "`false`" if the required lock mode is incompatible with one hold by a concurrent client.

2.2 Generalised Stochastic Petri Nets (GSPN) and Deterministic Stochastic Petri Nets (DSPN)

Modern computer, production and communication systems process complex workloads with random service demands. The most direct method for performance evaluation is based on actual measurement of the system under study. Nevertheless, during design, the system is not yet available and performance still needs to be predicted [1,2,3]. Most performance evaluation approaches are based on discrete-event simulation system models. Analytical models can be broadly classified into non-state and state space models, where the most commonly used state space models are Markov chains. In order to determine steady-state probabilities of a finite Markov chain, at least three different approaches for solution of the linear system (Interval Inter Quartile) IIQ = 0 are commonly considered: direct and iterative numerical methods, and a technique that yields closed form results. When real world problems are studied, Markov chains tend to become very large. Therefore it is attractive to be able to specify such systems in a compact way avoiding error-prone and tedious creation of models, and allowing designers to focus more on the system being modelled than on low-level modelling details. Hence,

separation of high-level model description and low-level computational models and its automatic generation are of great importance. GSPN (Generalised Stochastic Petri Net) is a prominent member of such generation models.

Petri nets are a family of formal specification techniques that allow for a graphical, mathematical representation and have powerful methods, which enable designers to perform qualitative and quantitative analysis [3]. Place/transition Petri nets are used to model a logical point of view of the systems, however no formal attention is given to temporal relations and constraints [1,2,3,4,5]. The idea of associating random time delay was first explored by Natkin [6] and Molloy [4] and this was the start of the emergence of Stochastic Petri Nets and their extensions. GSPN [1] is one of the most extensively used classes of stochastic Petri nets. GSPN are obtained by allowing transitions to belong to two different classes: immediate and exponential transitions.

Many reasons can be given for modelling a system, such as: existing systems are modelled for a better understanding, and analysis of deficiencies, such as bottlenecks, or for upgrading studies; and models are built during design of future systems in order to check if requirements are met [3,5]. For each case, different levels of details are considered. When designing a system, few details are known in advance, models tend to be more abstract and then analytical methods are more appropriate.

High-level models of systems are the first step to be accomplished. Such models are built considering information already available in this stage of design or based on previous modelling knowledge. Conceptual validation of such models is commonly accomplished as an iterative process based on step-wise refinement. After gaining confidence in the high-level model, low-level models are generated and metrics are calculated. Generating low-level models directly is often out of question due to their size and error-prone process of creating them by hand. High-level models allow a compact representation of low-level models to be analysed. Several high-level specification techniques have been suggested, among them GSPN is one of the most prominent. It is assumed that the reader is familiar with Petri net notation and graphical representation, nevertheless the structure as well as its firing semantics are informally described in the following paragraphs.

A GSPN is defined by a set of places, a set of transitions, relations describing pre-conditions, post-conditions, and inhibition conditions; and a mapping from the set of places to the natural numbers describing the model's state. The set of places represents the set of resources, local states and system variables. The set of transitions represents the set of actions. This set is divided into two subsets: the set of immediate transitions that depicts a set of irrelevant actions under the performance point of view; and the subset of timed transitions. Besides, two other functions are taken into account for representing timing and priorities. The timing function associates to each timed transition a non-negative real number, depicting the respective exponential transition delay (or rate). The priority function associates to each immediate transition a natural number that represents the respective transition priority level. Transitions are fired under interleaving firing semantics, a common semantics adopted even in

the untimed place/transition model. However, as defined, immediate transitions have higher priority than those timed transitions.

From a given bounded, live and reversible GSPN, a reachability graph is generated containing vanishing and tangible markings. This reachability graph is a semi-markovian process and can be analysed identifying an embedded Markov chain that describes transitions from state to state of the process. Steady-state and transient analysis are carried out on such models. Another possibility is to simulate the GSPN model in order to obtain steady state or transient measures.

In many modelling situations the adoption of exponential distribution may be adequate. Nevertheless, this assumption may be too restrictive for many real life cases. Therefore, over the last years, significant efforts have been devoted to studying analysis techniques for non-markovian stochastic Petri nets in which the exponential assumptions are partially not considered [9]. Among those considered methods for treating with timed transitions with more general distribution functions, methods based on phase approximation have been considered [11, 10]. One of the most commonly used phase approximation class is based on a mixture of Erlang and hyper-exponential distributions. Such an approach has been applied giving good fits to some commonly occurring distributions such as Weibul, deterministic, lognormal and uniform as well as empirical distributions [1,2, 29,30]. For instance, an Erlangian distribution of γ order and parameter λ consists of γ exponential transitions in series each one with parameter λ.

Deterministic and stochastic Petri nets (DSPN) [3] is an extension GSPN. DSPN allow the association of a timed transition either with a deterministic or an exponentially distributed firing delay. Hence, DSPN are well suited to represent system features such as time-outs, propagation delays or processor rebooting times which are naturally associated with constant delays. Early applications of DSPN include performance modelling of an Ethernet bus local area network [7] and a fault tolerant clock synchronization system [1].

3 Petri Net Models of CCS

This section describes the stochastic Petri Net model of the CCS. Actually, three models have been derived from a general model. Each model takes into account specific statistics obtained from data measured considering specific basic components (lock and unlock operations). These models are quite similar in terms of their structures. Therefore, a first model is described in details and the other two ones are presented where they differ from the first.

In order to define and evaluate the proposed Petri net models, we have adopted a methodology composed by ten phases that goes from system understanding to the presentation of the evaluation results. Hereafter, this methodology is briefly depicted:

Phase 1: this phase consists of understanding the problem (system) to be evaluated, its interface with other systems, the respective criteria that should represent the system performance, and highlighting the system components where the respective metrics should be obtained as well as performance of the representative components.

Phase 2: an abstract performance model is generated in this phase. In this model, timing information related to system components are not yet expressed, that is, those respective timed transitions do not define any specific time distribution – they are "generic" timed transitions.

Phase 3: the measuring process related to basic system components should be carried out according to a specified protocol of the measuring process [29].

Phase 4: the set of data collected in the previous phase is considered for obtaining a set of statistics (mean, standard deviation, variance, coefficient of variation, etc.) where outliers may or may not be discarded.

Phase 5: in particular, the first and second moment matching process has been adopted for representing empiric distributions according to the respective values related to coefficient of variations [8].

Phase 6: after moment matching, "generic" timed transitions are transformed into subnets representing hyper-exponential, hypo-exponential and Erlang distributions or single exponential or deterministic transitions. In this phase, the set of performance criteria specified in the initial phase should be stated in terms of probabilistic formulas representing throughputs, capacities, average times, availabilities etc. The products of this phase are, therefore, refined models.

Phase 7: a refined model should be validated by taking into account a set of specific scenarios in which real measures are obtained from the system under evaluation and compared with the respective results provided by the model evaluation.

Phase 8: in order to keep complexity under control, depending on the potential state space size and model structure, the refined model might be disaggregated into smaller sub-models, individually evaluated, and then aggregated back into a higher-level model that probably generates a state space size smaller than the original refined model [12]. However, this task should only be considered if the evaluation team, for practical reasons, faces problems in terms of memory space or evaluation time.

Phase 9: after generating the refined models, the evaluation method should be chosen (steady state or transient analysis/simulation) according to the performance criteria/metrics to be evaluated and the model's structure. The choice of these metrics is made according to specific criteria that represent system's performance. In order to generating the refined model, the abstract model is refined according to the specific values of the statistics obtained previously (defining the distribution type to be considered). The refined models are those taken into account in the evaluation phase.

Phase 10: after carrying out the evaluation method of choice, the respective results should be carefully analyzed, interpreted, highlighting bottlenecks, and actions might be recommended for the system's tuning.

3.1 Refined Models

Considering the phases defined in the preview section, a set of the middleware components and their interactions were obtained for generation of the abstract model. In order to derive the models (generated from the abstract model), statistics (mean and standard deviation) related to measurements associated with `lock` and `unlock` operations had to be obtained. For measuring the basic data that are considered for obtaining the refined models, HRTimer implemented for ITimer interface of JAVA has been adopted, since it provides time information milliseconds. Four methods have been used for carrying out the measuring of `Lock` and `Unlock` operations. The method `start`() starts counting the time for a new time interval and the `stop`() method ends this counting. The method `getduration`() returns the duration of the time intervals that elapsed between the last `start`() and `stop`() calls. Finally, the method `reset`() can be called from any state and will reset the timer instance back to the 'ready ' state.

Considering the phases defined in the preview section, the middleware components and their interactions were obtained for generation of the abstract model. In order to derive the models (generated from the abstract model), statistics (mean and standard deviation) related to measurements associated with `lock` and `unlock` operations (see Section 2.1) have to be obtained.

In order to measure the specific performance information related to the basic system's operations, a controlled environment has been considered. This environment consists of: Pentium III 800 MHz, with HD of 20 GB (5400 RPM Ultra-ATA/100), and 190 MB of RAM; Windows 2000 Professional (version 5.00.2195 - Service Pack 4); Development environment: Java 2 SDK, Standard Edition, version 1.4.2_02; and OpenORB, version 1.4.0 BETA2 Release [8].

The experiments were carried out using CORBA/OpenORB [8] concurrency server, the name server and the clients applications were all executing in the platform described previously. We measured the time of solicitation and answer of the operations `lock` and `unlock` in the read and write modes. We used as unit of time the millisecond (ms) and a sample of size 12.000, but the 2000 first samples were discarded. With the data, we computed the measures of central trend (mean, median, mode) and measures of dispersion (variation coefficient, standard deviation, variance). However, since this work considers first and second moment matching for deriving the refined models, only mean and standard deviation are presented.

These basic measurements registered the respective time information for `lock` and `unlock` operations. For the lock creation, the mean and the standard deviation obtained were

0.012080 ms and 0.024392 ms, respectively. For the `unlock` operation, the mean and standard deviation were 0.011185 ms and 0.045813 ms, respectively. If outliers are not taken into account the mean and the respective standard deviation, for the lock operation, were 0.011676 ms and 0.000806 ms, and, for `unlock` operation, 0.010131 ms and 0.000767 ms.

Considering these statistics, the refine models were generated. The first refined model takes into account only the mean values obtained for `lock` and `unlock` operations; the standard deviations are not considered. The other two models, on the other hand, consider both the mean and the standard deviation. The statistics generated from rate measured for lock and unlock operations consists of 10.000 considered measurements. However, for moment matching, outliers have been removed, hence 1140 measures have not been considered for the `lock` operation and 1123 for the `unlock` operation. The second model that those transitions representing lock and unlock operations are depicted by subnets describing an Erlang distribution, since the statistics obtained show smaller variability than exponential distribution, and according to the first and second moment matching, the Erlang distribution, is a better approximation. The third models represents the lock and unlock operations by deterministic transitions, since the very small variability (standard deviation) suggest it as an interesting representation.

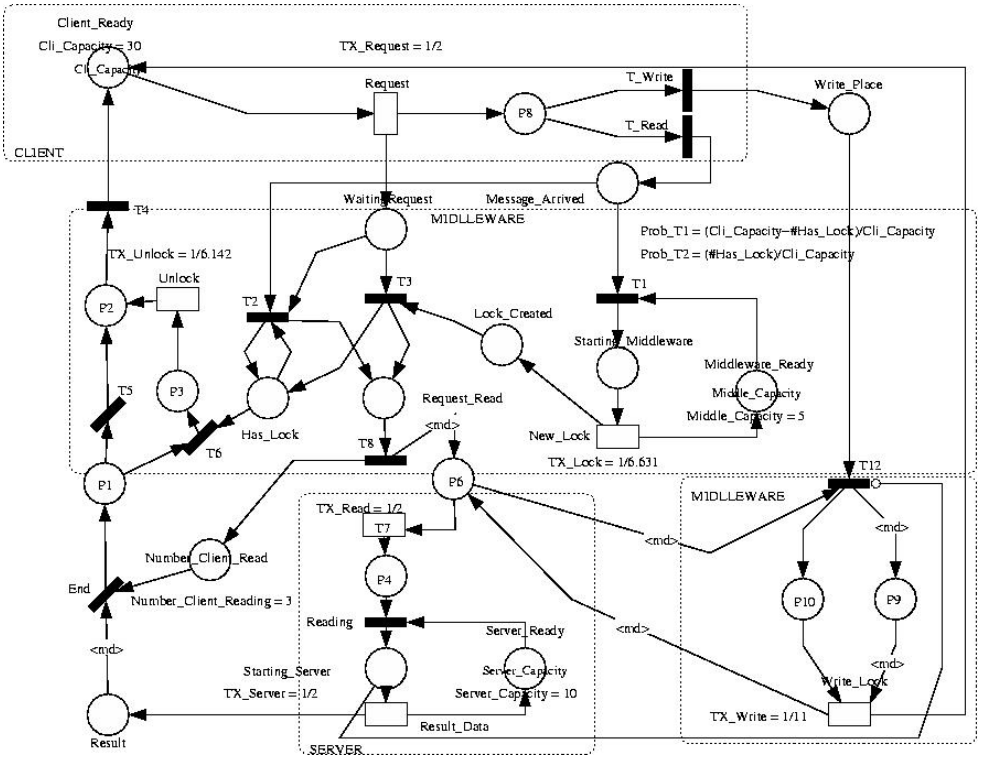


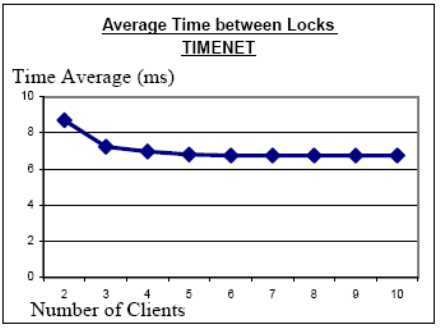
Figure 2 – Model Exponential Transition

Figure 2 represents the CCS specifying the operations of `lock` (`New_Lock`) and `unlock` (`Unlock`) in the reading mode. For a reading operation to occur, one must make a reading request and have it granted, to be able to read. In the described modelling, the client makes a request to a variable rate (`TX_Request`), this request solicitation can be to read (`T_Read`) or write(`T_Write`), where a choice probability exists that can vary from 0% to 100%. The transitions `T1` (`Prob_T1`) and `T2` (`Prob_T2`) can be associated to probabilistic association to each one depending on the initial marking and the amount of clients who can lock. After getting the lock, the client is qualified to execute this operation in the server at a rate of reading request variable (`TX_Read`) and will make a certain number of consecutive readings (`Number_Client_Reading`) predefined initially as three. In order to perform these readings, there is a variable rate to serve (`TX_Server`), the client is served at the variable rate.

Finally, the operation of `unlock` (`Unlock`) has a probability varying from 0% to 100%. If the option is to continue with the lock, a new reading operation is made without

taking into account the lock creation. In this model, an exponential transition is adopted to represent the operations for creating and releasing the lock.

Comparing their respective results with those obtained by measuring the real system has validated the proposed models. In the scenario considered, the average time between locks was calculated varying the amount of clients, and client requests set at a predefined rate of 1/6.7333 per second, the reading mode probability set to 1 (which means that only reading will occur), the lock creation rate set to 1/0.0117469, the server service rate set as 0.5 per millisecond, reading request rate specified as 0, since we will not read any data, but only pass through the server and then release the lock (Unlock) at a rate of 1/0.00951843 per millisecond.



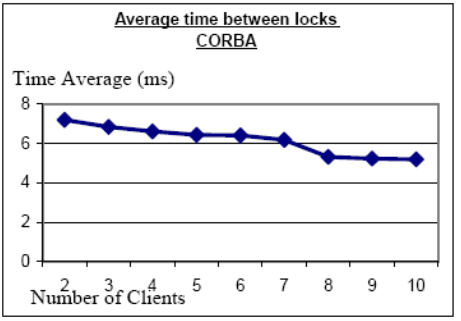


Figure 2.a- Average Time between Locks (TimeNET) Figure 2.b- Average Time between Locks (CORBA)

Considering 10 clients, the average time between locks obtained through the model with deterministic transition taking into account stationary analysis in Figure 2.a, was around 6.8 milliseconds. And measuring the CORBA application, using the OpenORB in Figure 2.b, it was around 5.2 milliseconds. It is observed that the average time between locks decrease to the measure that we increase the number of client, arriving at one determined point that tends to stabilize, in Figure 2.a in 6.8 ms and Figure 2.b in 5.2 ms. With this, it is perceived that the bottlenecks of this scenario is the request rate, therefore, if to increase this rate, we perceive certain variability in the average time between locks. With these measures, it is verified that the reproduced model presented quite similar results.

The initial statistics indicate that the generic transition should be substituted by an Erlang distribution of 257 phases for the `New_Lock` transition and 157 phases for the Unlock transition. The large number of phases is associated with the coefficient of variation related to time duration of these operations. Such large number of phases, depending on the models' structure, makes the analysis process (state space generation) hard to be carried out. Therefore, in order to validate the proposed models, we now faced to alternatives: validate it by simulating the proposed GSPN model or by analyzing (numerical analysis) the respective

model taking into account a smaller number of phase than those recommended by the moment matching process, if obviously, it does not significantly affects the considered metrics. We have adopted the numerical analysis for validating the model. The proposed model has been analyze considering Erlang subnets with 4, 6 and 8 phases, since it has been observed that after that number of phase (8), the respective results were no significantly affected. The average time between requests of locks was around 6.8 milliseconds, presented in Figure 3.

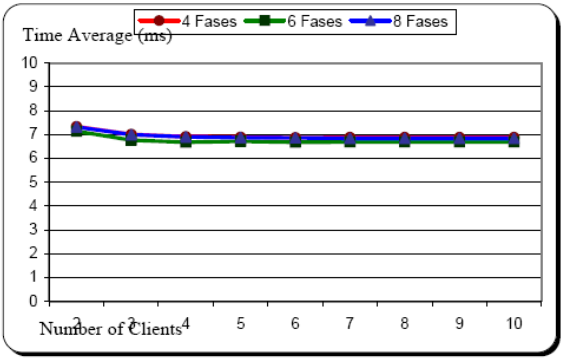


Figure 3 – Average Time between lock (Erlang)

The first refined model (obtained from the abstract model) describes each operation by exponential transition (first moment matching). This models as well the followings (abstract and other refined models) are composed by three subnets: Client, Middleware and server. Figure 2 depicts these models, where each subnet has been delimited by dashed-lines.

The subnet Client is composed by a place named `Client_Ready`, where its initial marking represents number of clients to be considered for evaluation. The transition Request represents client's request rate for a given service. The two immediate transitions, `T_Write` and `T_Read`, are considered for specifying reading and writing operations, associated to a given load. The weights attached to these transitions allow defining several load scenarios.

The subnet representing the middleware's concurrency control service has place called `Starting_Middleware` that represents the number of clients waiting to get a lock. Whenever a client request a reading operation, a token is stored in place `Waiting_Request`. If the client already has a lock (place `Has_Lock` marked), the transition `T2` is immediately fired as soon as a token arrives in place `Message_Arrived`. However, if the client does not have a lock, one is created. The transition `New_Lock` depicts the lock creation (the time associated to this transition represents time demanded for creating a lock) and the place `Lock_Created` represents the lock creation. Therefore, a token in the place `Waiting_Request` and another in the place `Lock_Created` allows the transition `T3` firing. The transition `Unlock` models the reading (the time associated to this transition depicts the respective releasing time).

Since, this work focus on reading locks, the writing operation has been coarsely represented, although precisely expressing the concurrency mechanism between reading and

writing operations. The time related to the whole writing operation is attached to transition `Write_Lock`. The weights of the arcs connecting the place `Write_Lock` and place `P6` assures mutual exclusion between reading and writing operations. The subnet Server is composed by the `Server_Ready` place in which the associated the amount of clients that can be simultaneously served. The exponential transition `Result_Data` represents the service execution (the time attached to this transition represents the service time). The `Starting_Server` place represents the number of clients being served.

Figure 2, the rate associated to `T1` is `Prob_T1 = {(Cli_Capacity - #Has_Lock) / Cli_Capacity}`, thus the probability of firing `T1` depends on the number of the tokens in places `Client_Ready` and `Has_Lock`. Considering this weight expression, the probability o firing transition `T1` (that is part of the lock creation) increases as number of the tokens in place `Has_Lock` decreases. On the other hand, the probability firing transition `T2` increases as the number of tokens in place `Has_Lock` increases. This is represented by the probability expression attached to transition `T2` (`Prob_T2 = {#Has_Lock / Cli_Capcity}`). When every client has a lock, it does not make reuse creating a new lock, this is correctly expressed by the probability expression attached to transition `T1`(it is evaluated to 0). This causes a confusing situation, since a structural choice is represented, but no probabilistic choice is allowed. Therefore, in order to avoid such a scenario, a guard function has been attached to the transition T1 `(#Has_Lock < Cli_Capacity)`. This guard function disables `T1`, whenever the number of locks matches the number of clients.

It is important to stress that the weights associate to transitions `T_Write` and `T_Read` allow expressing several scenario thing into account alternative loads in terms of reading and writing operations. The time associated to transitions `New_Lock` and `Unlock` have been measured according to the process described in the beginning of this section. Several extended conflict sets (ECS) have been defined in order to avoid confusion [6]. The priorities associated to the immediate transitions are depicted in Table 1.

Table 1. Extend Conflict Set - (ECS)

ECCs	Priority	ECCs	Priority
T_Write, T_Read	5	T8	8
T1, T2	4	Reading	3
T3	2	End	10
T12, T4	9	T5 ,T6	0

The Figure 4 shows the second derived model, in which the lock and unlock operations are rep-resented by Erlang sub-nets, that is the transitions `New_Lock` and `Unlock` been substituted by respective Erlang nets. The time associated to transitions `New_Lock` and `Unlock` are 0.011740533 ms, 0.009517518 ms, respectively. Besides, these ECSs described earlier, anew ECS is considered: ECS={r1,r2,r1_un,r2_un}, priority=1.

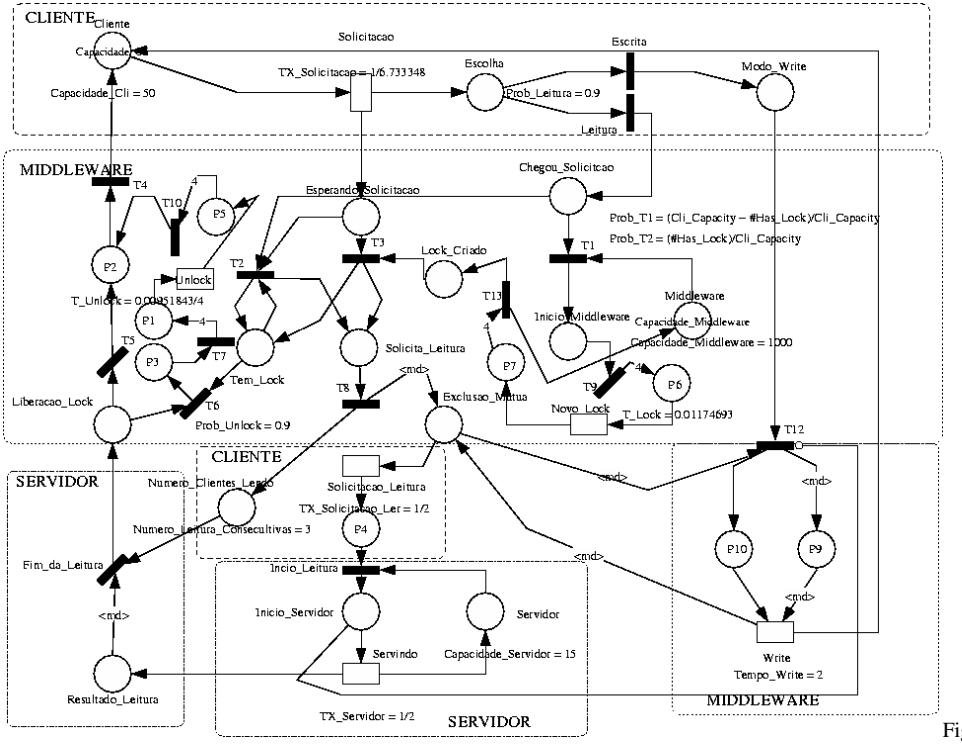


Figure 4 – Model Subnet Erlang

Due to space restriction, the third model in which lock and model operations are represented by deterministic transitions, is not presented in this work, since it has the same structure of the net depicted in Figure 2. The only difference is that the transition `New_Lock` and `Unlock` are specified as deterministic transitions the time associated to these transitions are 0.011676 ms and 0.0100131, respectively.

4 Evaluation

In order to evaluate the adequacy of applying a given evaluation technique (stationary simulation and analysis) to the model, a parameter (number of clients) was varied to verify the possibility of using each respective techniques. It is important to say that stationary analysis can become inadequate, due to the model's structure and availed the memory of machine in use.

The structure of the obtained model does not disable the use of the analysis. However, the number of clients can be a restriction for the application of this technique, given the available memory of the used machine. The configuration of the machine used for evaluation was: processor Athlon XP 2.0 MHz, HD of 40 GB and 528 MB of RAM. With the operational system Linux Gnome distribution version 2.6, the Timenet tool version 3.0[10] and the Microsoft Excel of the package of Office 2000. It was observed that, for seven or more clients, the memory of the machine was insufficient for the storage of the infinitesimal generating matrix.

Therefore, a stationary simulation was adopted as the chosen technique in this work. Using a confidence interval of 95%, the minimum number of firings for each 50 transitions and a relative error of 10%. In the presented experiments, the constant and variable parameters referring to each scenario have been defined. Finally, we present the interpretation of the results.

Next, we define two scenarios which have been simulated in the TimeNET [10]. In these scenarios some values are constant and have been obtained by measurement as described earlier. These values are the service request rate (1/6.733348 ms-1), associated to the transition Request, the lock creation rate (1/0.0117469 ms-1), attached to the transition `New_Lock`, and the lock release arte, associated the transition `Unlock` (1/0.00951843 ms-1).

Scenario 1 concerns the evaluating the middleware and system throughputs according to the number of clients. Some assumptions have been made in order to evaluating the models. It has been assumed that the respective load executes 90% of reading operations. This was set (in the model) by assigning the respective transition weights to transitions `TX_Reading` and `TX_Writing`. It is adopted that each client makes three consecutive readings. These readings are represented by the `Number_Client_Reading` variable, set as 3, which defines the weight of the arcs of exit from the `T8` transition to place `P6`, from the place `Result` to transition `End`.

It is assumed that the middleware can treat with 1000 simultaneous clients at most. In fact, a greater number of clients could be used. The number adopted in this scenario does not present a restriction for the performance of the system. The maximum number of clients is represented by the marking of the `Middlware_Ready` place (defined as 1000, indicating that at most 1000 locks can be simultaneously created). The probability of choice between releasing and holding the lock is assigned to the transitions `T5` and `T6`. `T5` represents the lock, defined as 10% and `T6` the release of the lock, defined as 90%.

Still in this scenario, it is adopted that the read request rate, assigned to transition `T7` is equal to the rate of the server, attached to the `TX_Server` transition and defined as ($0.5\ ms^{-1}$). This means that, at each 2 millisecond a new reading request takes place and 2 milliseconds later the operation is finished.

The server has a capacity of service, assigned to the place `Server_Ready`, defined as 10, indicating that only 10 clients can be simultaneously served. It is assumed that the average writing time is 2 ms, following an exponential distribution function (attributed to the `Write_Lock`). The values of creation and releasing of the lock are accumulated in a unique transition (exponential).

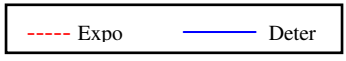
With this, the system of throughput (P{#Starting_Server>0}*TX_Server) and the middleware throughput (P{#Starting_Middleware>0}*TX_Lock) are calculated, where the first indicates the number of readings completed per time unit and the second indicates the number of locks created per time unit.

In the second scenario, this experiment consists in the evaluation system throughput and middleware throughput varying the reading probability from 10% to 90%. The scenario described below presents only the values modified in relation to the previously defined scenario. In this scenario some values are constant and have been obtained in the measurement phase. The number of clients, assigned to the place `Client_Ready`, is defined as 50. The server has a capacity of service, attached to the place `Server_ Capacity`, defined as 15, indicating that only 15 clients can simultaneously be served.

6.1 Results

The experiments were carried out simulation with the TimeNET[10] tool. The preceding section presented two GSPN models and one DSPN model for the CORBA concurrency service. The first GSPN model (Figure 2) uses exponential transitions and the second one uses an Erlang subnet (Figure 4). The DSPN model uses deterministic transitions.

The experiments were carried out using these models for analyzing the performance behaviour of CCS CORBA under different conditions and components parameterization. The chosen evaluation method was stationary simulation (steady-state). The metrics calculated were the system throughput and the middleware throughput.



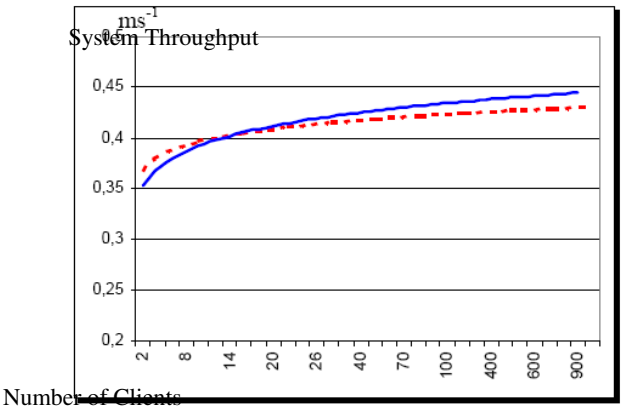
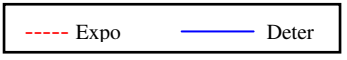


Figure 6 – System throughput versus Number of the Clients

The experiment provides performance results for the first scenario defined. The first evaluated measure is the system throughput varying the number of clients. The adopted models were the exponential model (Figure 4, first model) and the deterministic transition (third model).

The results obtained related to the deterministic model is plotted as continuous line as well as the results related to the exponential model are represented by dashed lines. The obtained results are presented in Figure 6, and one should observe that the system throughput increases as the number of clients increases. Considering 7 clients the throughput of the system is around 0.4 per millisecond, i.e., at each 2.5 ms a reading is completed.

The curves, presented in Figure 6, show that the system throughput increases as the number of clients also increases. The results obtained from deterministic transition model vary from 0.356881(2 clients) to 0.443739(1000 clients). Considering the exponential transition model, the system throughput varies from 0.368352 (2 clients) to 0.427738 (1000 clients). It has been noticed that the system throughput increases and that it tends to stabilize, hence there might bottlenecks such as reading re-quest rate and/or the capacity sever.



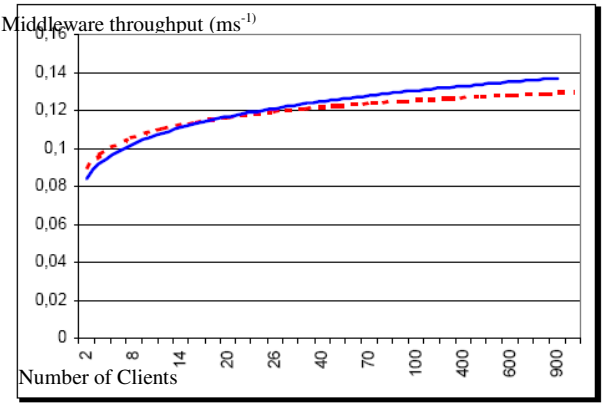


Figure 7 – Middleware throughput versus Number of the Clients

The second evaluated measure is the middleware throughput varying the number of clients. The adopted models were the exponential model (Figure 4, first model) and the deterministic transition (third model).

The obtained results are presented in Figure 7, and one should observe that the middleware throughput increases as the number of clients increases. Considering 6 clients the throughput of the system is around 0.1 per millisecond, i.e., at each 10 ms a lock is created.

The curves, presented in Figure 7, show that the middleware throughput increases as the number of clients also increases. The results obtained from deterministic transition model vary from 0.0863889(2 clients) to 0.144604 (1000 clients). Considering the exponential transition model, the middleware throughput varies from 0.0803402(2 clients) to 0.127753(1000 clients). Therefore, is has also been noticed that the middleware throughput approaches a steady value.

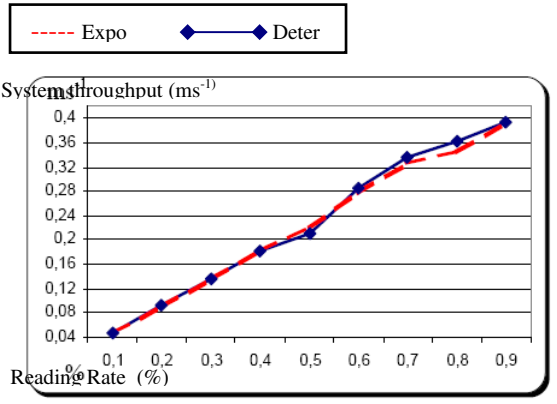


Figure 8 – System Throughput versus Reading Rate

This experiment provides the performance results, considering the second proposed scenario. In Figure 8 it is observed that the system throughput increase with rate reading. The number of clients requesting a reading and the number of clients who perform a complete reading also increase.

In this graph the system throughput provided by using deterministic model varies from 0.0471776 (10% clients) to 0.392984(90%) and using exponential model varies from 0.0473628 (10%) to 0.390049 (90%). With these results, it is noticed that in the two models the throughput of the system linearly increases. If show that, for there considered values, the server capacity does not impere and performance constraint.

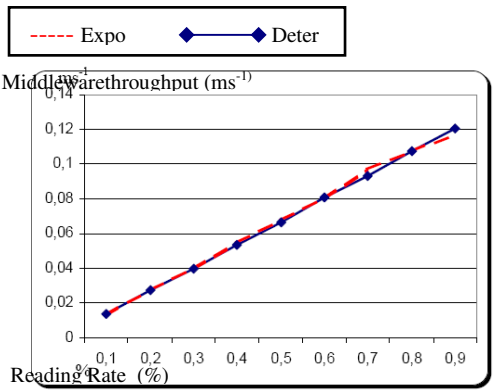


Figure 9 – Middleware Throughput versus Reading Rate

In Figure 9 it is observed that the middleware throughput tends to increase with the growth of the probability of the reading rate. Therefore, when the reading probability is low, fewer requests are made to the middleware. As a consequence, less lock is created.

In this graph the calculation of the middleware throughput deterministic model varies from 0.0134788 (10%) to 0.120651 (90%) and the result related to the exponential model varies from 0.0131042 (10%) to 0.116607 (90%). With these results, it is observed that in the two used models, the throughput of middleware increases. Hence, according to the observed results, one may concludes that for the adopted TX_Request (obtained from the considered application) does not flood the middleware.

5 Related Work

Related works on modelling and analysis of distributed systems have received considerable attention during the last two decades. A review of technologies involved in Transaction Processing (TP) with a variety of performance models for major building blocks of a TP system were given in [4]. A number of product-form Queuing Network (QN) models for delay analysis were presented in [11] for application servers, disk subsystems, and networks. Computationally efficient algorithms were proposed for product form QN models using Mean Value Analysis. However, these models presented deficiencies in handling the interaction of various processes such as simultaneous possession of more than one resource. In [25] the QN are used for describing client/server system using CORBA as a middleware and the response times estimated from the model are compared to the measured response times for a growing number of clients, in order to assess the accuracy of the model and the values of the parameters in the model. This model can then be used for designing a distributed application, before the entire system is installed or even fully implemented.

Related works on demultiplexing focuse largely on the lower layers of the protocol stack (i.e., the transport layer and below) as opposed to the CORBA middleware. For instance, [1, 2, 3] study demultiplexing issues in communication systems and show how layered demultiplexing is not suitable for applications that require real-time quality of service guarantees. In [12] presents two contributions to the study of demultiplexing for real-time CORBA end systems. First, an empirical study of four CORBA request demultiplexing strategies (linear search, perfect hashing, dynamic hashing, and active demultiplexing) is presented for a range of target objects and operations. Second, describe how are using the perfect hashing and active demultiplexing strategies is described aiming at developing a high performance, real-time ORB called TAO [18]. An important class of applications (such as avionics, virtual reality, and telecommunication systems) require scalable, low latency communication. Our earlier work [5, 7] shows why latency sensitive applications that utilize many servants and large interfaces are not yet supported efficiently by contemporary CORBA implementations due to demultiplexing overhead, presentation layer conversions, data copying, and many layers of virtual method calls. On low speed networks this overhead is often masked. On high-speed networks, this overhead becomes a significant factor limiting communication performance and ultimately limiting adoption of CORBA by developers. The results presented in this paper compare four demultiplexing strategies. The object and operation lookup used by these strategies are based on linear-search, perfect hashing, dynamic hashing, and de-layered active demultiplexing. Results reveal that the de-layered active demultiplexing strategy provides the lowest latency and most predictability amongst the three strategies studied. We have currently implemented the de-layered active demultiplexing strategy in TAO, which is our high performance, real-time ORB [18].

Related work on modelling and analysis using Petri nets can be used for the event service and notification service of the CORBA. The Event service is the earliest CORBA solution to the message queue model of communication in distributed systems. Typical implementations however suffer from the lack of event delivery guarantees. The loss of

messages is aggravated in the presence of burstiness in the input to the Event service, and occurrences of isolated bursts of traffic could also have serious effects. In this

paper we develop stochastic reward net (SRN) models that can aid in the study and configuration of the Event service to conform to design

specifications. To capture burstiness in the input, Markov modulated Poisson process (MMPP) is used as the input source. Erlang distributed event consumption times are used in the models to accommodate more general distributions and a wider range of variances. The models also take into consideration the FIFO discard policy adopted in many Event service implementations. The applicability of the models to the CORBA Notification service is also briefly discussed. In [9] studied the most common implementation of the CORBA COS Event service using stochastic reward net (SRN) models. The models can aid in configuring the Event service by

determining operating ranges that satisfy design specifications. The models were used to study the effect of Poisson arrivals as well as bursty and correlated arrivals by means of an MMPP event source model. We also demonstrated the use of the models to determine the effect of an isolated burst in input. A model to study the scheme of state resynchronization between suppliers and consumers was also developed. The Event service specification does not enforce event delivery guarantees and does not specify any QoS settings.

In [22] the Notification service provides standards based QoS settings to messaging in CORBA applications. Our models can be used to configure a system for minimum event loss as in the case of the Event service, as well as guide the choice of the quality of service as in the case of the Notification service. The Notification Service provides several configurable quality of service (QoS) and administrative settings that deal with issues such as reliability, event (message) delivery order and discard policies. Unlike in conventional queuing systems, some Notification Service QoS configurations can lead to discards from within the internal queues, requiring careful analysis and configuration if such discards are to be avoided or minimized. This paper presents stochastic models (based on continuous time Markov chains and queuing theory) for analyzing the Notification Service delivery and discard policies in detail. The CORBA Notification Service aims to provide a very flexible messaging framework for CORBA applications. The Notification Service QoS properties that can be chosen include event delivery and discard policies, and the maximum queue lengths. Apart from these choices the effect of a particular event arrival rate needs to be determined in order to regulate the suppliers, if necessary. In this work we developed stochastic models based on continuous time Markov chains (CTMC) and queuing theory to analyze the performance of the CORBA Notification Service under various combinations of these policies. The utility of the models was illustrated by studying the effect of event arrival rate on the measures of interest for delivery and discard policies based on FIFO, LIFO, and priority that the Notification Service allows. The applicability of the models to evaluate deadline violations was also outlined. The models can also aid in the assignment of priorities to important events when priority based delivery and/or discard policies are used. The models were also shown to be useful to study

the effect of varying the maximum queue length on the computed measures. Thus the models can be considered to verify that design specifications are met, as well guide the choice of quality of service settings.

6 Conclusion and Future Work

Middleware has a large diversity of features such as price, complexity and performance. For each target application, different aspects are more important than others. The process of choosing the correct middleware technology is directly related to deciding which qualities have more significance for the applications. However, a common requirement for almost all applications is the performance behaviour. Although other techniques may be applied to evaluate middleware's performance, it is mainly based on measurements that consider test environments. Performance of an existing system can obviously be evaluated by measurements, but such an approach is not possible during its design phases or even if one intends to evaluate different scenarios in order to tune systems performance. In such cases, one may resort to model-based evaluation techniques. Thus, the search for simulation and analytical models that allow complex system performance behaviour reproduction has received significant attention by the distributed systems scientific community.

This paper proposed two GSPN models and one DSPN model for CORBA CCS. Performance evaluation of these models was carried out through steady-state simulation and the models accuracy has been confirmed by comparisons with measurements performed in a real test measurement environment. The GSPN model demonstrated to be very accurate and succeeded in catching the main performance characteristics of a real CCS. Moreover, some performance analysis could be extended to verify the behaviour of some metrics while changing different factors and their respective levels. In general, results presented the ability of modelling in evaluating CORBA Concurrency Service using GSPN and DSPN.

The CCS GSPN-based model is suitable to possible extensions, some of which are currently in progress, for instance, the introduction of operation upgrade, change_mode in the sender or receiver side as well as performance evaluation of CORBA Concurrency Service taking into account phase approximation distributions. Other work in progress also considers evaluating specific CORBA Transaction Service.

References

[1] Marsan, G Chiola. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. Advances in Petri Nets, vol 266, Lecture Notes in Computer Science, Springer Verlag, Edited by G.Rozenberg, pp 132-145. 1987.

[2] T. Murata. Petri Nets: Properties, Analysis and Applications. Proceeding of The IEEE, 1989.

[3] C. Lindemann. Performance Modelling with Deterministic and Stochastic Petri Nets. John Wiley and Sons, 1998.

[4] M. K. Molloy. On the Integration of Delay and Throughput Measures in Distributed Processing Models. PhD. Thesis, UCLA, Los Angeles, CA, 1981.

[5] R. German. Performance Analysis of Communicating Systems - Modeling with Non-Markovian Stochastic Petri Nets. John Wiley and Sons, 2000.

[6] G. Balbo. Introduction to Stochastic Petri Nets. Lectures on Formal Methods and Performance Analysis, Springer Verlag, 2001.

[7] G Bolch, S. Greiner, H. de Meer, K. Trivedi. Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications. John Wiley and Sons, 1998.

[8] A.A.Desrochers, R.Y.Al-Jaar Applications of Petri Nets in Manufacturing Systems. IEEE Press. 1995.

[9] S. Ramani, K. Trivedi, B. Dasrathy Performance Analysis of the CORBA Event Service Using Stochastic Reward Nets. 19th IEEE Symposium on Reliable Distributed Systems, Nurberg, Germany, October, 2000.

[10] Aniruddha Gokhale and Douglas Schimdt. The Performance of the CORBA Dynamic Invocation Interface and Dynamic Skeleton Interface over High-Speed ATM Networks. In GLOBECOM, volume 1, pages 18-22, November 1996.

[11] H. Ammar, S. Islam. Time Scale Decomposition of Class of Generalized Stochastic Petri Net Models. IEEE Transaction on Software Engineering. Vol 15, no. 6, June, 1989.

[12] Aniruddha Gokhale and Douglas Schimdt. Evaluating the Performance of Demultiplexing Strategies for Real-time CORBA. In GLOBECOM'97, volume 3, pages 1729-1734, Phoenix, Arizona, USA, November, 1997.

[13] Sun Microsystems Inc. JavaTM Transaction Service Speci_cation, December 1999.

[14] Matjaz B. Juric, Ivan Rozman, and Simom Nash. Java 2 Distributed Object Middleware Performance Analysis and Optimization. ACM SIGPLAN Notices, 2000.

[15] Te-Kai Liu, Amir Behroozi, and Santoshi Kumaran. A Performance Model for a Business Process Integration Mid-dleware. In IEEE International Conference on E-Commerce, pages 191-198, June 2003.

[16] Te-Kai Liu, Alan Feket, and Ian Gorton. Predicting the Performance of Middleware-based Application at the Design Level. In WOSP, pages 166-170, January 2004.

[17] OMG. Common Object Request Broker Architecture: Core Specification (CORBA 3.0), 2002.

[18] Michael Pang and P. Maheshwari. Benchmarking Message-Oriented Middleware TIB/RV vs. SoniqMQ. In Work-shop on Foundations of Middleware Technologies, California, USA, October 2002.

[19] Dorina Petriu, Hoda Amer, Shikharesh Majumdar, and Istabrak Abdul-Fatah. Using Analytic Model for Predicting Middleware Performance. In Second International Workshop on Software Performance, pages 189-194, Ottawa, Canada, September 2000.

[20] Srinivasan Ramani, Katerina Goseva-Popstojanova, and B Trivedi, S Kishorhy. A Framework Performability Model-ling of Messaging Services in Distributed Systems. In 8th IEEE International Conference on Engineering of Complex Computer Systems, pages 25-34, December 2002.

[21] Srinivasan Ramani, S Kishor Trivedi, and Balakrishnan Dasarathy. Performance Analysis of the CORBA Event Ser-vice Using Stochastic Reward. In 19th IEEE Symposium on Reliable Distributed Systems (SRDS), pages 238-247, October 2000.

[22] Srinivasan Ramani, S Kishor Trivedi, and Balakrishnan Dasarathy. Performance Analysis of the CORBA Notification Service. In 20th IEEE Symposium on Reliable Distributed Systems (SRDS), pages 227-236,October 2001.

[23] Weili Tao and Shikharesh Majumdar. Application Level Performance Optimizations for CORBA-Based Systems. In Third International Workshop on Software Performance, pages 95-103, Rome, Italy, 2002.

[24] Phong Tran and Paul Green_eld. Behaviour and Performance of Message-Oriented Middleware Systems. In 22nd International Conference on Distributed Systems Workshops (ICDSW'02), pages 1-6, 2002.

[25] T. Verdickt, B Dhoedt, F. Gielen, and P. Demeester. Modelling the Performance of CORBA Using Layered

[26] Steve Vinoski. Where is Middleware? IEEE Internet Computing, 2002.

[27] M. Ajmone-Marsan, G. Conte, and G. Balbo. A class of Generalised Stochastic Petri Nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems, 2:93–122, 1984.

[28] J.L. Peterson. Petri Nets and the Modeling of Systems. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[29] Joseph M. Juran, A. Balnton God Frey .Juran.'s quality handbook, 1998, 5 th edtion.