Second International Workshop on GNs, Sofia, June 26-27, 2001, 1-12

The use of generalized nets within tasks of software performance engineering

Andreas Schmietendorf[1,2], Evgeni Dimitrov[1], Krassimir Atanassov[3]

1 *T-Nova Deutsche Telekom Innovationsgesellschaft mbH, Entwicklungszentrum Berlin, Wittestraße 30N, D-13476 Berlin, E-Mail: A.Schmietendorf|Evgeni.Dimitrov @telekom.de*

2 *Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik, Institut für verteilte Systeme, Postfach 41 20, D-39016 Magdeburg, E-Mail: schmiete@ivs.cs.uni-magdeburg.de*

3 *Centre of Biomedical Engineering, Bulgarian Academy of Sciences, BG-1113 Sofia, E-Mail: krat@bgcict.acad.bg*

Summary: The technology of Enterprise Java Beans (EJB) provides an architecture for component-based client/server applications. Because this technology is increasingly used for critical corporate applications, the question of performance is gaining importance. This paper concentrates on the possibilities for net-based performance analyses of multi-level client/server applications developed by means of the application server technology (EJB). The analysis uses Generalized Nets (GNs) - including timed Petri net functions - which appear eminently suitable for performance analyses based on simulative solutions. The performance models are derived with the aid of UML diagrams which, for this purpose, were given extended performance aspects, using annotations, stereotypes and constraints. The results achieved are also validated on the basis of a benchmark created with the real application.

Keywords: Performance analysis, network models, multi-level architectures, application servers, generalized nets, Enterprise Java Beans, UML

1 Introduction and motivation

The number of software systems with distributed multi-level architectures based on Web and Internet technologies has increased rapidly in the last few years [Gartner 1999]. Applications in the field of e-business (e.g.. e-commerce) play a major role in this field. Critical factors for the acceptance of such applications include, in addition to the required functionality, performance characteristics such as response times, throughput and the workload of the system resources used.

If the information system to be developed is to exhibit the required performance features, we need appropriate methods and models for carrying out performance analyses. Performance analyses should be embarked on in the early phases of software development, so that an unsatisfactory software design can be abandoned before any possible implementation. Thanks to advances in system design, it is now possible to achieve ever more detailed performance models.

There are a number of different approaches, models and methods available for the modeling and analysis of performance problems and features, such as:

- Queuing networks (extended QN, layered QN,...),

- UML models (extended representation of time aspects)
- Net-based models (Petri nets and their extensions),
- Models based on formal FDT descriptive techniques (SDL, Lotos, Estelle,..),
- Graph models (execution graphs see also [Smith 1990]),
- Tool-based systems (SES/strategizer).

This paper concentrates on both a UML-based procedure for modeling performance aspects and an analysis of the options for a net-based approach based on it, using generalized nets (GN) according to [Atanassov 1991], [Atanassov, 1997] for performance modeling.

Section 2 demonstrates how distributed C/S architectures and their performance aspects can be modeled and interpreted using GN elements. Section 3 describes how the performance models analyzed by means of GNs are derived from UML diagrams with added performance aspects. The mapping rules and conventions used in connection with GNs are applied in Section 4 to the modeling of a specific CoNCCaT C/S system. The results of the simulative analysis of these GNs are validated with the results obtained from an application benchmark using the real system, the model being thus verified (Section 5).

The GN model built is based in this analysis on a genuine prototype, in particular, but with a few minor technical modifications it should be applicable to a broad class of 3-tier C/S applications.

2 Performance modeling with generalized nets

2.1 Models based on Petri nets

There are a number of disadvantages and weaknesses associated with using ordinary Petri nets (PN) [Petri 1962] for the modeling and analysis of complex systems, especially distributed IT systems.

- Rapid growth in size and the associated complexity of the model,
- Arbitrary solution of conflicts in the net model when representing synchronization problems,
- Restriction to non time-related considerations,
- Inability to operate with local and global data.

To overcome these disadvantages, different extensions and modifications of the ordinary PN are proposed:

1. Assignment of special characteristics to places, transitions and arcs:
   - Special places (e.g. decision places)
   - Special transitions (e.g. fork, join and macro transitions)
   - Special arcs: inhibitor or reset arcs, colored arcs
2. Changing the token profiling and adding local and global data

   To reproduce the data flow of the modeled system exactly, the tokens were given attributes (E-nets, PRO-nets) and colors (colored PN). System data and states can be modeled with additional local and global data regardless of the tokens. An example of the use of colored Petri nets for architecture modeling is given in [Xu 1998].

3. Adding of time characteristics to transitions, tokens or arcs as well as of global system clock

   To enable us to make statements about performance, transitions and places were assigned time factors (processing times and delay times). Similar work focusing on the idea of timed Petri nets has been done by [Ramchandani 1974] with the introduction of deterministic switching times for transitions or [Molloy 1982] who introduces time aspects as random variables, which resulted in stochastic Petri Nets (SPN).

4. Changing the firing rules for transitions by evaluating additional logical conditions.

   This means that decisions can be made, conflicts resolved and priorities observed. The time a transition takes can therefore model the time for the execution of a method.

2.2 Generalized nets: overview

With *Generalized Nets* we are dealing with a generalization of the existing modified and timed Petri net, in particular with regard to the various modeling requirements. In general, a GN is described by a 4-tuple with the following groups of elements:

$$E = ([A, \pi_A, \pi_L, c, f, \theta_1, \theta_2], [K, \pi_K, \theta_K], [T, t^0, t^*], [X, \Phi, b])$$

1. A tuple used to describe the ***net structure:*** functions (θ_1, θ_2) for the firing rules and execution time of transitions A and conditions f for evaluating net structure elements with priorities (π_A, π_L) and capacities c of the places.
2. A tuple used to describe the static properties of ***tokens***: a permissible quantity of all tokens K, assigned priorities π_K and entrance points θ_K in the GN.
3. A tuple for depicting ***time characteristics***: global time scale T for executing the net model, definition of an elementary time step t^0 for the progress of model execution and the observation interval t^*.
4. A tuple for depicting the ***changes in token characteristics***. With the quantity of possible characteristics X, the quantity Φ of possible changes during the execution of the model and the maximum number b of characteristics a token can have.

Depending on the requirements of the GN model, specific elements of individual tuples can be eliminated using an appropriate projection function. This results in special classes of GNs so most of the Petri net extensions can be covered.

2.3 Building of GN models

To construct a efficient GN performance model the following steps must be carried out:

1. Representation of the ***static structure*** of the modelled process or system
   - *Mapping Events ←→ Activities:* a specific activity, which is a part of the whole process, should be assigned to each particular event.
   - *Modeling the events by GN-transitions:* a GN transition is assigned to each event of the modeled process in the frame of the GN-model

- *Modeling the firing conditions:* The conditions for the completion of an event in the GN-model are represented by: the presence of tokens in the corresponding input places of the transition modeling the event (i.e., complying with the transition type), the presence of predicates in the condition of the transition which have a truth value *"true"*, and the presence of enough room in the output places of the transition

2. Description of the ***dynamics*** of the modeled process or system
   - Defining the sub-processes to be executed either simultaneously or in parallel
   - Defining the initial characteristics of the tokens representing the sub-processes in the net model: priorities, time data (entrance points), and other token characteristics that model the key process data.
3. Determination of the ***time aspects*** of the modeled processes or system
   - The location of the modeled process within some absolute temporal scale is determined by the three global temporal components of GN
   - Which process starts functioning first?
   - Which process terminates functioning first?
   - What happens to the two processes at a given moment of the absolute temporal scale?
   - If some event takes place in the first process at a given time moment, what event does take place in the second process?, etc.
4. Adding of relevant ***data*** of the process to be modeled. This data is divided into data relating to
   - transitions
   - places or
   - tokens.

2.4 Evaluating the model parameters

Even as the model is designed, appropriate values for the main parameters are defined (CPU time used, time for disk accesses and operations, the time for the necessary net services) and incorporated into the model. These are often estimated values or findings obtained from other models or expert knowledge.

The following standard options exist for obtaining up-to-date values for these parameters:

- Measurements of software elements already implemented, e.g. basic services, existing SW components or prototypes of a proposed new software system
- Results of compiler runs with the existing code
- Estimates of requirements for CPU, I/O etc. based on the experiences of software and hardware designers and systems engineers
- Performance requirements for the software to be produced.

In addition, GNs support work with *uncertain* parameter values (for *fuzzy* and *intuitionistic fuzzy* GNs, see also [Atanassov 1991], [Atanassov, 1999]). These classes of net and standard GNs with token parameters having fuzzy or intuitionistic fuzzy values enable us to use fuzzy or intuitionistic fuzzy estimates of various model parameters.

2.5 Solutions for GN models and the interpretation of results

A number of special classes of GN models can be solved by set formulae using mathematical calculations. In general, however, simulation is the preferred method of solving complex GN models. So, for example, bottlenecks caused by arc and place capacities in the net or by the values of token characteristics can be calculated by running several simulations. The combined values of token characteristics from several simulations can be used for statistical calculations. A brief description of the key features of the simulator used here is given by [Schmietendorf 2000].

Performance values (system response times, total system throughput and workload) are calculated on the basis of a given model workload and data for the processing times of individual transitions in the GN representing the duration of the activities of system components.

With the aid of GN models it is not difficult to analyze and test design alternatives before expensive changes are made in the real system. Furthermore, the effect of new demands on the architecture of the future system can be depicted without much trouble. To diagnose the necessary system changes, we have to analyze precisely the causes of unsatisfactory delays and throughput results.

3 The CoNCCaT prototype: Structure and basic use cases

The CoNCCaT (**C**omponent **o**riented **N**etwork **C**entric **C**omputing **A**rchitecture **T**elekom) prototype was developed by T-Nova/DTAG for the chief purpose of collating experiences of new software technologies like Enterprise Java Beans (EJB) and the J2EE platforms.

The practical purpose of the CoNCCaT prototype was to obtain a consistent view of so-called core information objects (CIO) forming part of various Deutsche Telekom information systems. The customer object in particular was the focus of attention. From the technical point of view, the application is a client/server environment connected on the one hand to a mainframe database system and also accessing data from the SAP system. It enables users to view and maintain, on a graphical user interface, customer data with marketing significance originating partly from a mainframe-based application and partly from an SAP database.

CoNCCaT runs on the "GemStone/J" application server by GemStone Systems written entirely in Java and implemented on an NT platform. The GemStone server is connected via a JDBC (Java Database Connectivity) driver to an Oracle-8i database that provides temporary storage for customer data (CIO database).

The client, also programmed in Java, gives users the following use cases:

- to create new customer data records (***insert***),
- to search existing data records (***search***) and
- to modify existing data records (***modify***).

Use cases were modeled with the aid of the eponymous UML diagrams. To describe workloads and user performance requirements, the latter were extended by means of annotations and constraints. Annotations were used to define the type of request AA (or request chain type AKA), the frequency q of the AA or AKA and the respective client preparation time AGVZ (thinking and typing times) in a UML use case diagram. Using constraints, we were able to define the user requirements for the response time (t_{Ref}) and the throughput (B_{Ref}) of a specific request type. Extension of the use cases was done with reference to ISO 14756 and the "synthetic clients" defined there [Dirlewanger 1994].

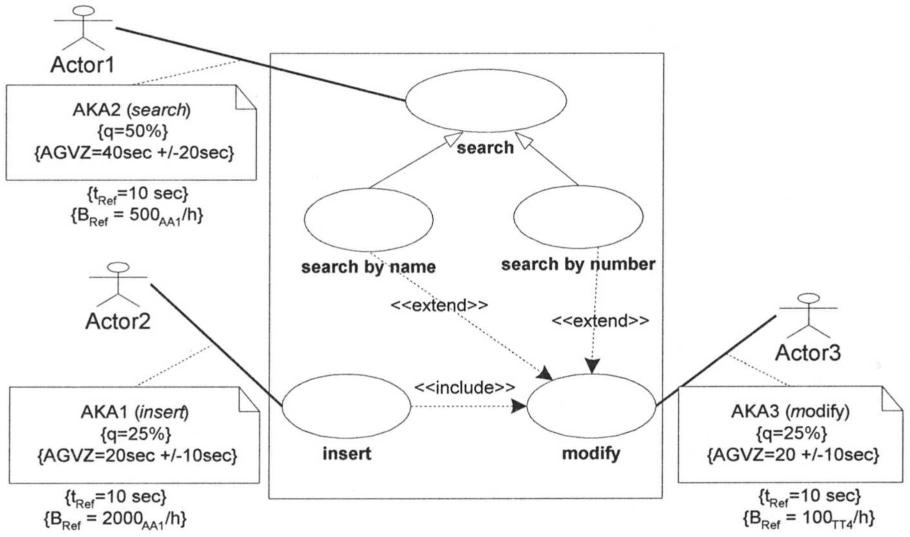


Figure 1: Extended UML use case diagram of the CoNCCaT proptotype

Figure 1 shows the specified use cases for the application analyzed. In a subsequent step the use cases were reduced using appropriate UML interaction and sequence diagrams, and these were extended by constraints and tagged values to model the performance-related characteristics of the system analyzed.

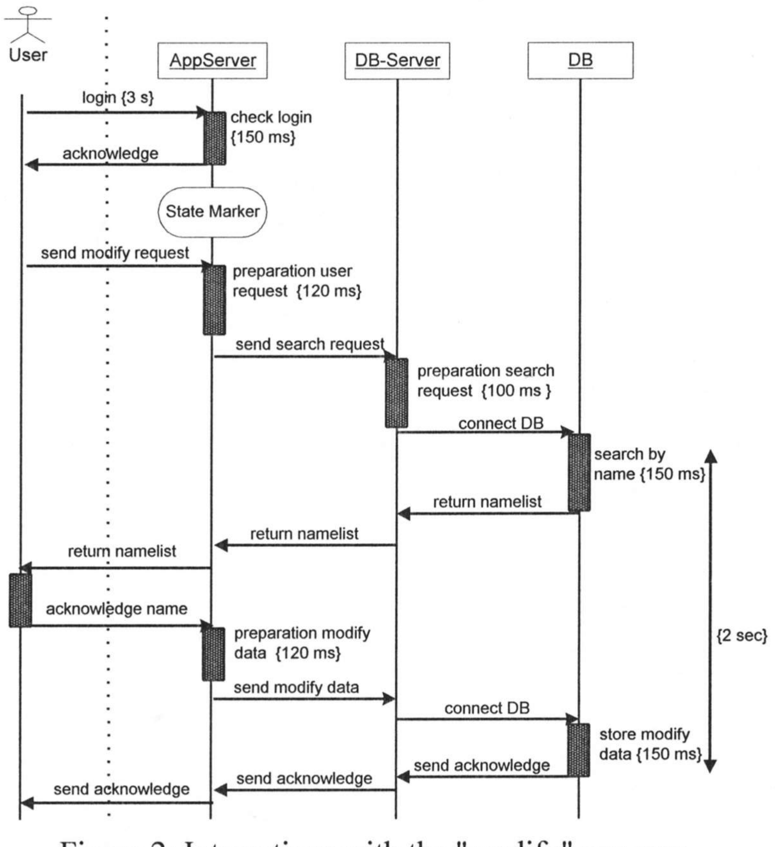
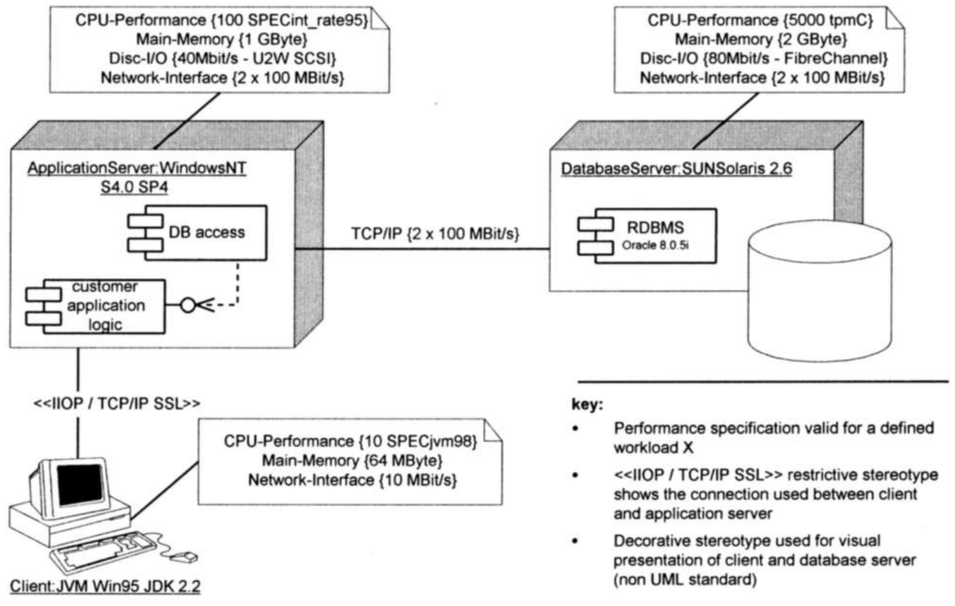


Figure 2: Interactions with the "modify" use case

The sequence diagram shown in Figure 2 demonstrates the software components involved in executing the "modify" use case. The following individual steps can be identified:

0. Login; Client sends modified request to application server
1. Preparation user request via the application server and send search request to DB-Server
2. Opening of a database connect; search by name and return name list to the user
3. Client acknowledges the name and sends modified customer data record to application server
4. Preparation of the modified data via the application server and sending this data to the DB-Server
5. Opening of a DB-connect; store the modified data record in the DB
6. Confirmation the change to the application server and send acknowledge to the requesting client.

Figure 3 shows the general architecture of the prototype analyzed with the aid of a UML deployment diagram. The individual functions of the server are performed using the corresponding Enterprise Java Beans (customer application logic). Again extensions were applied consisting of annotations, constraints and stereotypes, partly in order to define the performance capability of the server systems (in relation to the assumed workload scenario) and partly to determine the network bandwidths needed. The client may be connected to the server via the IIOP protocol (Internet Inter ORB Protocol), for example, which is defined in the CORBA specification, runs on a TCP/IP and also uses SSL (Secure Socket Layer) functions. SSL functions have a considerable influence on performance in practice and need to be looked at in more detail elsewhere. On this subject see, for example ([Menasce 2000] .p.127 ff.). The extensions introduced so far are based on a comprehensive analysis of UML notation in the context of performance modeling [Schmietendorf et. al. 2000].


CPU-Performance {100 SPECint_rate95}
Main-Memory {1 GByte}
Disc-I/O {40Mbit/s - U2W SCSI}
Network-Interface {2 x 100 MBit/s}
CPU-Performance {5000 tpmC}
Main-Memory {2 GByte}
Disc-I/O {80Mbit/s - FibreChannel}
Network-Interface {2 x 100 MBit/s}
ApplicationServer:WindowsNT S4.0 SP4
DB access
customer application logic
TCP/IP {2 x 100 MBit/s}
DatabaseServer:SUNSolaris 2.6
RDBMS
Oracle 8.0.5i
<<IIOP / TCP/IP SSL>>
CPU-Performance {10 SPECjvm98}
Main-Memory {64 MByte}
Network-Interface {10 MBit/s}
Client:JVM Win95 JDK 2.2

key:

- Performance specification valid for a defined workload X
- <<IIOP / TCP/IP SSL>> restrictive stereotype shows the connection used between client and application server
- Decorative stereotype used for visual presentation of client and database server (non UML standard)

Figure 3: General architecture of the CoNCCaT prototype

After careful investigation of the feasibility and possible information payload in the modeling, it was decided to ignore the connection to the mainframe databases for the purposes of this analysis.

4 GN model of the CoNCCaT prototype

4.1 Model structure

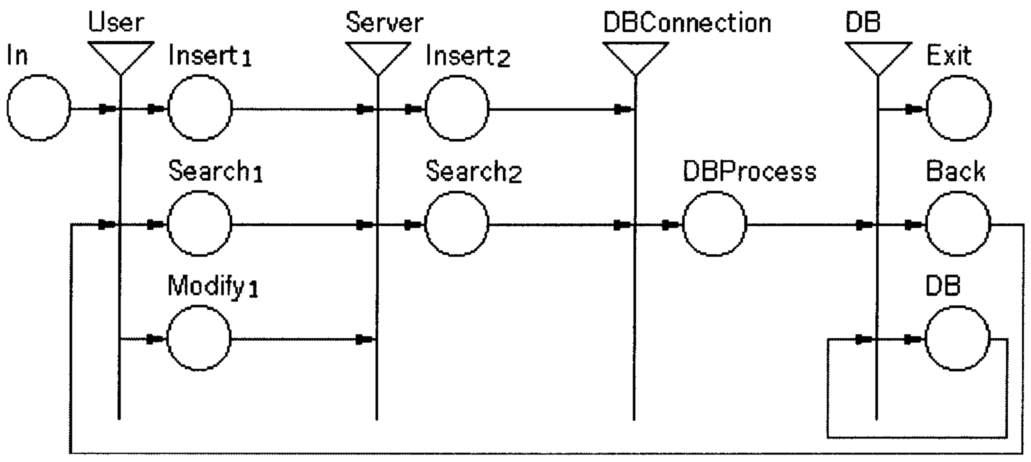


Figure 4: GN model of the CoNCCaT prototype

The GN model of the CoNCCaT prototype consists of four transitions that are interpreted as follows:

1. The "***User***" transition: models the activities (*insert*, *search* and *modify*) of the client component. The three exit places of the transition indicate the current type of activity.
2. The "***server***" transition: models the activities of the server component
3. The "**DBConnection**" transition: represents the connection with the database.
4. The "***DB***" transition: models the activities of the database.

Client requests are depicted by GN tokens. We can distinguish between 4 categories of token:

- Generation of a new customer (*insert*) – 25 %
- Search for customer data in the DB (*search*) – 50%
- Change of customer data in the DB after searching by name (*modify*) – 12.5 %
- Change of customer data in the DB after searching by account number (*modify*) – 12.5 %

For each request the client preparation time and the processing times for the order at the various workstations (GN transitions) are given as input values for the model. After the request is processed, its response time and the server workload are recorded. Both the server workload and the necessary resources are taken into account. The resources are represented by specifying a server capacity.

4.2 Input data of the model

The input data for the model generally consists of:

- the percentage probability of occurrence of the three use cases (see section 3)
- the client preparation times including thinking and typing times, and
- the execution times for DB queries.

Table 1: Input data of the model

Use Case	Client preparation time	Execution time / DB query
insert	20 sec. +/-10 sec.	5.37 ms
search	40 sec. +/-20 sec.	3.20 ms
Modify		
1. Search by name	20 sec. +/-10 sec.	3.20 ms
2. Search by Account No.	20 sec. +/-10 sec.	2.04 ms
3. Update	20 sec. +/-10 sec.	5.37 ms

The elementary values for the execution times for database queries were obtained by querying the relevant database discretely 5000 times.

Other values defined:

- Time required to execute a DB-Connect: 107 ms
- For a one-off execution of the request type *insert,* a mean response time of 233 ms was assumed, and for the request type *search* - 285 ms.

4.3 Results

The following capacities were defined for the application and database servers for executing the simulation model:

Application server (capacity of the places Insert2 and Search2): 120

Database server (capacity of the place DBProcess): 160

The distribution of the requests received at the start of the simulation was defined randomly, while, over the total duration of the simulation, requests were executed according to the defined workload mix.

Total number of *insert* requests: 2400

Total number of *search* requests: 4800

Total number of *modify* requests (search by name): 1200

Total number of *modify* requests (search by number): 1200

Figure 5 shows the response times calculated for the use cases analyzed. In the model executed the *modify* use case taken, differentiated (by number and by name), is striking for its poor response times.

One finding not explicitly depicted here is the comparison between the time taken to process requests in the application server and the times taken in the system as a whole. For the use cases *search* and *insert* run discretely, the increase in the server times is approx. 50-100%, while for the *modify* use case the time increases by approx. 600 %.

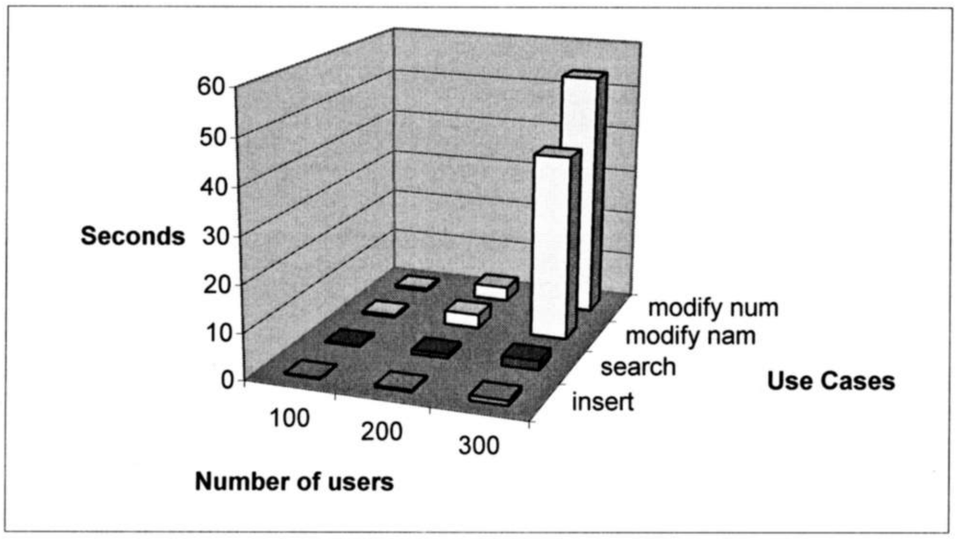


Figure 5: Response time data

It was not possible to show in this model workload variations of the IT system resources used. The main reason for this is that this type of resource was not explicitly covered in the model. We can therefore only make statements on capacities in terms of the overall application and database servers. A reverse transformation to a corresponding real IT system can therefore only be made with the assistance of appropriate benchmark results in association with performance results obtained for systems in use.

5 Validation and verification

It should be possible to validate the modeling results achieved by carrying out a benchmark test ([PerfEng 2000] gives a full description) using a genuine application and a workload driver system.

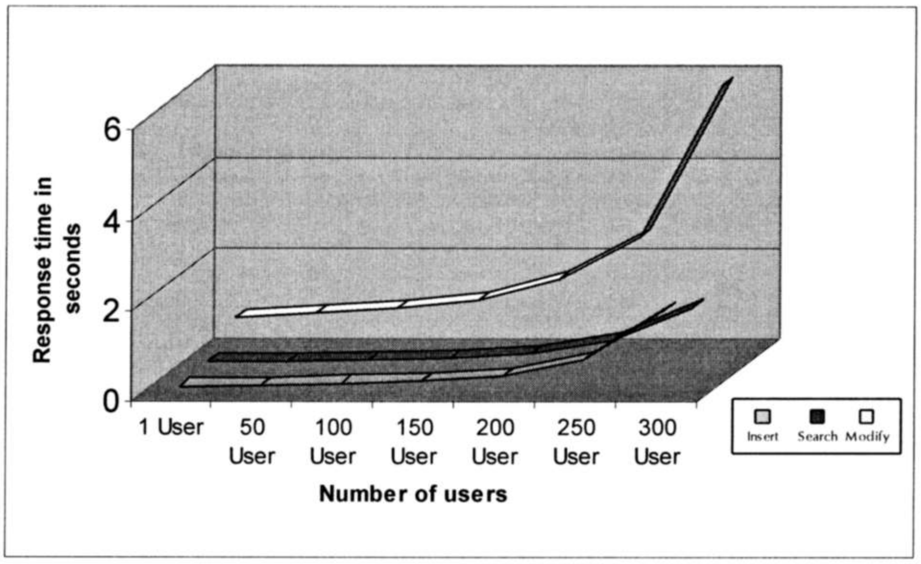


Figure 6: Results of benchmark test

Figure 6 shows the results of this measurement, with the response times plotted as a factor of the numbers of users working with the system. A detailed analysis, which for reasons of space can only be touched on here, also shows an almost 400% increase in response time for the *modify* request in comparison with the *insert* and *search* requests when measured with the defined workload mix.

The *modify* transaction, in particular, in comparison with the other two request types, clearly shows the congestion point in the prototype under review. This becomes even more marked if we consider the figures for another series of tests in which all the users only triggered *modify* requests, which is shown in Figure 7.

It was also clear from the test that 87 out of 250 users received no answer at all from the application server, and the corresponding figure for 300 users was 138. This demonstrates that the server has a problem holding large numbers of user connections. Unfortunately, it was not possible to compare the absolute response and delay times because of the difference in the workload scaling in the model and the benchmarking.

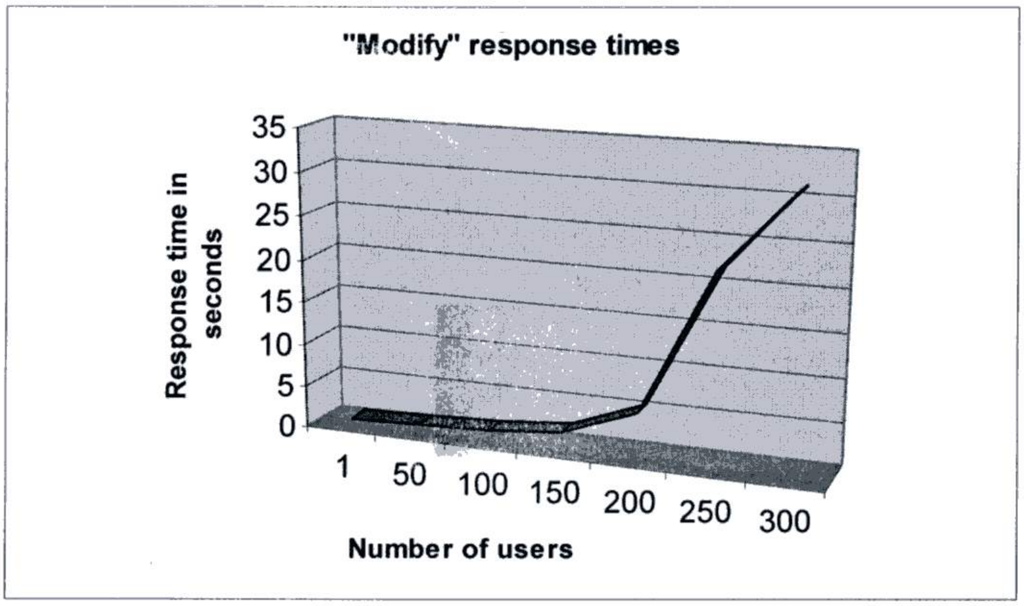


Figure 7: Test using only the *modify* request

6 Assessment and conclusions

Although the absolute values for the execution times of the *modify* use case in the Petri net model do not match the benchmarked values precisely, the results of modeling with generalized Petri nets indicate at least similar tendencies and identify the *modify* use case critical to performance.

Furthermore, by analyzing the difference between the times takes on the application server and the system as a whole shown in the model, we can demonstrate that the application server is <u>not</u> the limiting factor.

The causes of the differences in results with the model and the benchmarking are partly the degree of abstraction in the performance model itself (i.e. no exact replication of the application server and software components running on it), and partly the abstract nature of the database compared with the genuine application and the interface used to communicate with it.

7 Bibliography

[Atanassov 1991] Atanassov, K: Generalized Nets, Singapore, New Jersey, London, World Scientific, 1991.

[Atanassov 1997] Atanassov, K.: Generalized Nets and Systems Theory, "Prof. M. Drinov" Academic Publ. House, Sofia, 1997.

[Atanassov 1999] Atanassov, K.: Intuitionistic Fuzzy Sets, Springer, Heidelberg, 1999.

[Dimitrov 2000] Dimitrov, E.; Schmietendorf, A.: UML basiertes Performance Engineering. Integration der Aufgaben des Performance Engineering in die objektorientierte Softwareentwicklung unter UML, 1. Workshop Performance Engineering, Darmstadt, 2000, S. 10-20 (in German)

[Dirlewanger 1994] Dirlewanger, W.:Messung und Bewertung der DV-Leistung auf Basis der Norm DIN 66273. Heidelberg: Hüthig Verlag, 1994

[Gartner 1999] GartnerGroup, Symposium/ITxpo99 Documentation on CD-ROM, 1-4 November, 1999 Cannes France

[Menasce 2000] Menasce, D. A.; Almeida V. A. F.: Scaling for E-Business – Technologies, Models, Performance and Capacity Planning, Prentice Hall, Upper Saddle River, NJ 2000

[Molloy 1982] Molloy, M.K.: Performance Analysis Using Stochastic Petri Nets, IEEE Transaction on Computers, Vol. C-31, No. 9, 1982

[PerfEng 2000] Project PerfEng: Parts 1- 7, Internal Report, T-Nova / Deutsche Telekom, Development Center Berlin, 2000 (in German)

[Petri 1962] Petri, C. A.: Kommunikation mit Automaten, Dissertation, Bonn, 1962 (in German)

[Ramchandani 1974] Ramchandani, C.: Analysis of Asynchronous Concurrent Systems by Petri Nets, Technical Report, MIT, Laboratory of Computer Science, Cambridge, Massachusetts, 1974

[Schmietendorf 1999] Schmietendorf, A.: Anwendung von Modellösungen für ein Performance Engineering, MMB-Mitteilungen Nr. 36, Herbst 1999, pp. 10-20, (in German)

[Schmietendorf 2000] Schmietendorf, A.; Modellbezogene Notationen, Methoden und Tools für ein Software Performance Engineering, Preprint Nr. 15 der Fakultät Informatik, Otto-von-Guericke-Universität Magdeburg, December 2000 (in German)

[Schmietendorf et. al. 2000] Schmietendorf, A.; Dimitrov, E.; Dumke R.: Overview about a UML-based Performance Engineering, Preprint Nr. 14 der Fakultät Informatik, Otto-von-Guericke-Universität Magdeburg, November 2000

[Smith 1990] Smith, C.: Performance Engineering of Software Systems, Reading, MA et al., 1990

[Xu 1998] Xu, J.; Kuusela, J.: Modeling Execution Architecture of Software System Using Colored Petri Nets, in Proc. First International Workshop on Software and Performance WOSP98, Santa Fe, NM, USA, 1998