Modelling Aspects of Model-Based Dynamic QoS Management by the Performability Manager

Leonard J.N. Franken†, Raymond H. Pijpers†
Boudewijn R.H.M. Haverkort‡

†PTT Research
P.O. Box 15000, 9700 CD Groningen, The Netherlands
E-mail: {l.j.n.franken,r.h.pijpers}@research.ptt.nl
‡University of Twente,
Department of Computer Science Tele-Informatics and Open Systems
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: b.r.h.m.haverkort@cs.utwente.nl

Abstract. The *Performability Manager* (PM) is a distributed system component which maintains the application-requested Quality of Service (QoS) by dynamically reconfiguring ANSAware-based distributed applications, using a model-based optimization procedure. The PM receives information about the ANSAware-based application from a distributed monitoring process based on JEWEL and DEMON. With this information, and using predefined stochastic Petri net (SPN) models of ANSAware applications, the PM automatically constructs an overall SPN performability model which is subsequently used for the determination of the provided QoS. Based on the analysis results, the PM can decide to initiate on-line system reconfigurations, if such is needed for maintaining the requested QoS. ANSAware provides facilities for these dynamic reconfigurations.
In this paper we focus on the modelling aspects of model-based dynamic QoS management by the performability manager. We present an ANSAware-based experimental distributed environment in which the modelling and evaluation aspects are totally automated. We also show the feasibility of the proposed PM by presenting some operational results.

1 Introduction

For modern distributed systems it is important to be able to realize and maintain a requested Quality of Service (QoS). This QoS can degrade for several reasons: the addition of new applications, the updating of applications, the change of workload (new users) or the occurrence of failures and repairs.

QoS is difficult to define in general. Most importantly, it describes the *user-perceived performance* [22, 28, 34, 35]. The QoS can be divided in the subjective and the objective QoS. The subjective QoS is user oriented, and hard to quantify and measure. The objective QoS can be measured. We will always refer to the objective QoS. The objective QoS is related to or can be transformed into the subjective QoS, but this is not a one-to-one relation.

Service Performance Parameters (SPPs) is the generic term for provider visible performance parameters [8]. These are quantitative parameters which indicate how well the system (service) is performing. Between the objective QoS parameters and the SPPs there exists a one-to-one mapping [22, 28]. The SPPs can be measured at the service, and they ultimately determine the QoS, but they do not describe the QoS in a way that is meaningful to users (the subjective QoS).

As the QoS describes the user-perceived performance, the separate evaluation of performance, reliability and availability during system design, implementation and maintenance is not sufficient. The mutual influence of these aspects is recognized by the QoS and demands for modelling and evaluation techniques which can handle the combined aspects [24].

In [9] we introduced the *Performability Manager* (PM), a distributed system component which maintains the application-requested QoS by dynamically reconfiguring a distributed system, using a model-based optimization procedure. The PM does so by creating alternative configurations using (or manipulating) a graph-oriented model of the current configuration of the distributed system. The performability models of the alternative configurations are automatically generated from a graph model of the system obtained via DEMON [21], a library of predefined SPN model components, and parameterized via a monitoring process with JEWEL [19, 26]. By carefully selecting an alternative configuration, based on performability evaluation, an alternative (new) configuration can be decided upon. The alternative configuration can dynamically be effected using dynamic reconfiguration [18] as supported by the ANSAware computing platform underlying the distributed application.

For the operation of the performability manager, *model creation* and *evaluation* is both crucial and difficult. An overall performability model must be created, at run-time, out of model components. Apart from that, the performability manager must also be able to create alternative configurations for the current configuration. The performability manager uses performability analysis because that type of analysis provides us with the means to model and evaluate distributed systems with respect to their QoS [7, 9, 14, 23, 24].
In the realization of the PM the following questions arises:

1. how to detect QoS degradations of the current configuration;
2. how to create an alternative configuration;
3. how to determine the QoS of the alternative configuration.

These questions can be answered by either modelling (and evaluation) and/or monitoring. In this paper we will address both modelling (and evaluation) as well as monitoring aspects of *performability or QoS management* as done by the PM. The modelling aspects are important for configuration creation and configuration evaluation. Monitoring is important for parameterization of the models and to detect QoS degradations. We discuss these modelling and monitoring aspects by means of an experimental distributed environment realized using ANSAware.

This paper is further organized as follows. Section 2 presents work related to dynamic reconfiguration and performability management. In Section 3 we present the experimental environment and the ANSAware computational model of our application. Section 4 describes the graph-oriented modelling of distributed environments and shows how our experimental environment can be described using the proposed graph-oriented model. In Section 5 we present the creation of a performability model using predefined stochastic Petri net models of the system components and the graph-oriented model of the experimental environment. The monitoring of the experimental distributed environment is discussed in Section 6, whereas parameterization and first results on measurements are presented in Section 7. Finally, in Section 8, we discuss implementation and operational issues of the presented modelling techniques, discuss our ongoing research and set out lines for future research.

2 Related work

The performability manager maintains the required QoS by dynamic reconfigurations. This requires that facilities for dynamic reconfiguration should be available in the distributed system. Such facilities would include access, concurrency, federation, location, migration and replication transparency [27]. These facilities can be realized at several levels in a distributed system: at the operating system level, at the middle-ware level, think of computing platforms or configuration languages or at the application level itself. The current trend in distributed systems is to provide these facilities at the middle-ware level, i.e. by computing platforms. These platforms allow for heterogeneous distributed systems, which are transparent to the application programmer. Examples of such middle-ware facilities are the configuration languages Gerel, Conic, Argus, Rex, Darwin [18], and the computing platforms ANSAware [1] and DCE [29].

Computing platforms and configuration languages provide the user with the functionality for dynamic or/and static (re-)configurations. Examples of the use of these facilities for qualitative configuration management are for example described by Cole and Dean in [18]. Cristian presents in [18] an approach for a so-called availability manager, which guarantees the availability of the applications using replicated components. A performability manager extends this functionality by also addressing performance aspects.

Most of the effort in the area of dynamically reconfiguring distributed systems has been put in supplying facilities to perform the reconfiguration rather than on reconfiguration management to guarantee a desired level of QoS. The performability manager is therefore designed to guide reconfigurations by using a model-based optimization procedure. The goal of the reconfigurations is to maintain the QoS as requested by the application users. Performability evaluation will be used in the optimization procedure. A similar, but less general and less "automatic" approach towards resource control has been proposed by Lee and Shin [20, 32].

Further related to our work is the area of optimal system design [11, 15] and dynamic load balancing [3, 33]. Closely related is also the area of task allocation [31]. An example of this has been presented by Bowen [4] in which heuristic and linear-program solution for optimal process allocation in heterogeneous distributed systems are compared. Hariri [12] presents an algorithm which takes care of optimizing reliability and communication delay. The above approaches, however, are all focussed on single aspects of performance or reliability and not on their combination as we propose.

Another area related to our work is network management. From this area Kheradpir *et al* [16] proposes model based network management to manage the *end-to-end* network performance and robustness (dependability). They advocate a model based solution for future telecommunication systems to manage the QoS.

3 An ANSAware-based distributed environment

The application in our distributed environment is realised using ANSAware. In ANSAware the *computational model* is the model used for creation of applications and application components. This computational model is both object-oriented and client/server oriented. From this computational model we want to come to a *performability model*, which will be discussed in later sections.

In Section 3.1 we present a distributed application which will be used throughout this paper to clarify the different models. Section 3.2 describes ANSAware and the ANSAware computational model. Section 3.3 presents the experimental application using the ANSAware computational model.

3.1 An ANSAware-based number translation service

In this section we describe the (telephone) number translation service (NTS) as provided in intelligent networks [2, 10, 36]. In the sequel we will refer to this application as the IN/ANSA application.

End-Users are submitting requests or tasks for the application with a certain rate. Since we do not have real users, we mimick the user behaviour by a so-called *Generating Component* (GC). The GC generates the calls for the NTS. The NTS is provided by the following application components (see also Figure 1):

1. *The Selection Component* (SC): this component selects a service using the contents of the requests it receives (number translation service in this example).
2. *The Number Translation Component* (NTC): this component receives requests for number translations. The NTC sends a request to a database component for the required number and to a billing component for the creation of a bill. The number received from the database is returned to the SC.
3. *The DataBase Component* (DBC): this component receives requests for specific numbers. It will fetch the number from disk and return the number to the component which requested the number.

4. *The Billing Component* (BC): this billing component receives requests for the preparation of a billing record.

The *Management Component* (MC) does not belong specifically to the NTS, but provides the PM with the necessary "buttons to push" for performing a reconfiguration. The MC uses ANSAware facilities to perform necessary reconfigurations. The other components (SC, NTC, DBC and BC) are components of the application and can be controlled by the MC.

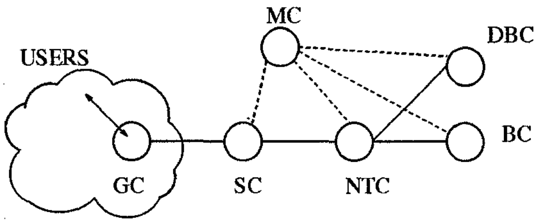

Fig. 1. The experimental ANSAware application

For the experimental application we use a small distributed system consisting of three SUN SPARC workstations connected by an Ethernet as depicted in Figure 2. The workstations run UNIX and, on top of that, ANSAware. Of course, more heterogeneous environments are possible as well, e.g.,using both SUNs and PCs.

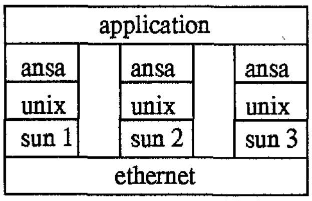

Fig. 2. The distributed system

Within this experimental distributed environment we use two monitors, DEMON and JEWEL. DEMON, the Distributed Environment MONitor [21], is used to visualize the structure of the experimental distributed environment. JEWEL [19] is used to do performance measurements in the experimental distributed environment.

3.2 ANSAware

ANSAware is a suite of programs which allows users to write applications suitable for heterogeneous distributed environments (see also Figure 2, although the possible heterogeneity is not directly apparent there). It is based on ISO-ODP, an emerging ISO standard for Open Distributed Processing [27]. ANSAware essentially consists of an infrastructure placed on top of the operating system, and provides a uniform, technology-independent platform upon which applications can be executed. The infrastructure permits interworking between applications running on remote and dissimilar machines. Several management applications, performing functions identified as important in ODP, are provided for the user's convenience. ANSAware provides a uniform view of a multi-vendor world, allowing system builders to link together components or existing software with minimal changes and overhead.

The basic building block of ANSAware is a *service*. Components that use a service are called *clients*. Components that provide a service are called *servers*. Services are provided at *interfaces*: an interface is a unit of service provisioning. This is also depicted in Figure 3. The ANSA computational model permits an object to be both client and server. A component or object described purely in terms of the way it provides and uses services, is referred to as a computational object. A client can invoke an operation or service at the interface of a server object in two different ways:

1. by interrogation, in which the invoking client waits for the server to perform the operation and return the result (similar to a remote procedure call);
2. by announcement, in which the invoking client does not wait for the server to perform the operation and no result is returned (remote process spawn).

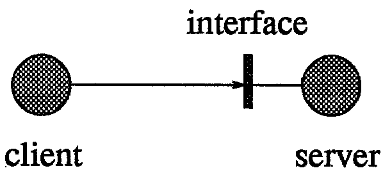

Fig. 3. A client object and a server object with its interface

The location of the computational objects or the type of machine they execute on can be changed at run-time: the ANSAware infrastructure enables a flexible configuration of application components and provides a uniform way of accessing them.

3.3 The computational model of IN/ANSA

In our experimental distributed environment the computational objects are the application components of the distributed system. One or more application components or computational objects make up a distributed application. This is depicted in Figure 4. Each computational object has been implemented as a process. All invocations for the experimental application are announcements, except for those between the NTC and the BC and those between the NTC and the DBC; these are interrogations.

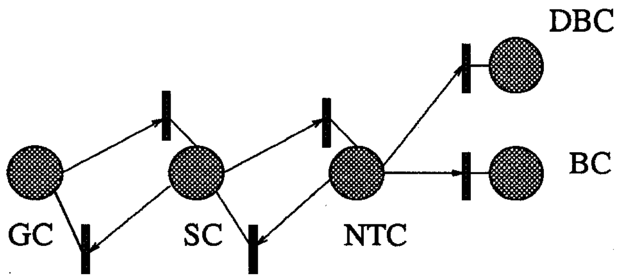

Fig. 4. The experimental application described in a computational form

4 A graph-oriented description of distributed environments

Because a reconfiguration must be performed with great care, the PM uses a model of the current distributed environment to prepare a reconfiguration. The performability manager uses this model to create alternative configurations and as a basis to derive a performability model.

Section 4.1 presents the graph-oriented approach to describe distributed environments. In Section 4.2 the IN/ANSA application is described using the graph-oriented approach.

4.1 Graph-oriented structure model of a distributed environment

In this section we propose a graph-oriented model suitable for the creation of alternative configurations and which allows for the transformation of these alternative configurations into performability models. For the graph-oriented model we look at a distributed environment at four levels (see also Figure 5) [9]:

Task level	Tasks
Application level	Applications

System level	System parts and network
Management level	Management

A similar distinction in levels has been proposed by Calzarossa *et al.* for workload modelling in network-based environments [5]. Each level will be described as consisting of components and their relations. These relations can be between the different levels as well as between the components at one level (mutual relations).

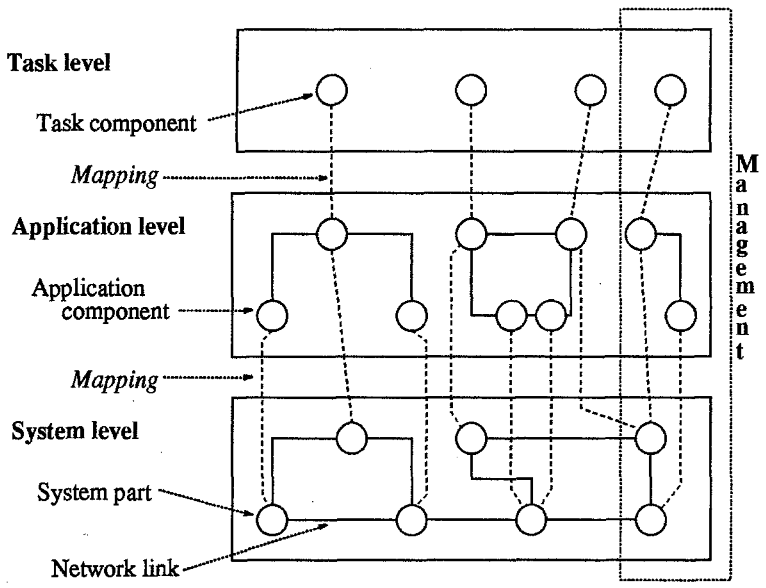

Fig. 5. A graph oriented model of a distributed environment

Task. At task level, tasks are viewed as *task components* t_d. A task is a certain amount of work, initiated by a user. For each application j we define the set of task components $T^j = \{t_1, \ldots, t_o\}$, $T^j \in TC$ with $TC = \{T^1, \ldots, T^n\}$ the set of all tasks for all applications. The task components have a relation with the application level. We assume that there are no mutual relations between task components.

Application. At application level, applications are viewed as composed of *application components* $a_i \in AC$, with $AC = \{a_1, \ldots, a_m\}$ the set of all application components. The application components have besides their relations to the task and system level, a mutual relation which represents the communication between the application components. Therefore, at application level we view an application as a structure of application components and their mutual relations (communication). We can describe a single application structure using a graph

$A^j = \langle VA^j, EA^j \rangle$ with $VA^j \subseteq AC$ and $EA^j \subseteq VA^j \times VA^j$. The complete set of (created) application structures which is active in the current distributed system is denoted by $\mathcal{A}$. Thus, we have $A^j \in \mathcal{A}$, and $\mathcal{A} = \{\langle VA^j, EA^j \rangle | j = 1, \cdots, n\}$, where n is the number of applications.

System. At system level we have a structure of the system level components, i.e. system parts $s_k \in SC$ with $SC = \{s_1, \ldots, s_w\}$ and the network links, $n_l \in NL$ and $NL = \{n_1, \ldots, n_t\}$ which provide the mutual relations between the system parts. We can describe the distributed system structure as a graph $S = \langle VS, ES \rangle$, with $VS \subseteq SC$ and $ES \subseteq VS \times VS$ and $F : ES \longrightarrow NL$.

At the system level we also define *paths*. A path is a finite sequence of network links (arcs or edges) between any two system parts (nodes), i.e.,a finite sequence of links in which the terminal node (system part) of each link coincides with the initial node of the following link [6]. We define P as the set of all paths on S, p_{kl} is the set of paths between the system parts s_k, s_l and $p_{kl_v} \in p_{kl}$, where $p_{kl_v} = \{(s_k, s_h)(s_h, s_f) \ldots (s_g, s_l) | (s_k, s_h), \ldots, (s_g, s_l) \in ES\}$. Every path $p_{kl_v} \in P$ can be mapped on a set of links of S, by the function $F : P \longrightarrow ES$.

The system parts have besides their mutual relation, a relation to the application level. The definition of a path will be used when assigning the communication between the application components to the system level. If no path exists between two system parts then these system parts are not connected.

Management. The management level is orthogonal to the other levels. It consists of management tasks, applications and systems, i.e.,the MC component and the monitors. They are structured as described above and are composed of components with the already mentioned relations and mutual relations.

Mapping. The *mapping* is the logical allocation of higher levels to lower levels, i.e.,the *allocation* of application components onto system parts and the *routing* of the communication over the network links. This is also reflected in Figure 5 where the mutual relations are represented by thick lines and the relations between levels (the mapping of the levels) by dotted lines. The allocation of tasks to application components is a special case of allocation. Are the other allocations subject to change, by migration etc., the task components are always allocated to the same application component of an application.

We can describe a mapping function M^j for each A^j. The allocation and routing of a mapping can be described in a more formal way. We will now elaborate more on the description of the allocation and routing.

Allocation. The allocation consists of two types of allocations: the allocation of task components to application components and the allocation of application components to system parts. We start with the former. We describe the allocation of task components to application components by the matrix $Z^j = T^j \times VA^j$:

$$z^j_{d,i} = \begin{cases} 1, \text{if the } t_d \in T^j \text{ is allocated on } a_i \in VA^j, \\ 0, \text{otherwise.} \end{cases}$$

For the allocation of application components $a_i \in A^j$ on system component $s_k \in S$ the mapping can be described by $ma^j : VA^j \rightarrow VS$. We can derive such a function for all $VA^j \in \mathcal{A}$.

For the allocation of $a_i \in VA^j$, and $VA^j \in \mathcal{A}$ on system component $s_k \in VS$, we define a parameter $x^j_{i,k}$ as follows:

$$x^j_{i,k} = \begin{cases} 1, \text{if the } a_i \in VA^j \text{ is allocated on } s_k \in VS, \\ 0, \text{otherwise,} \end{cases}$$

For A^j we can create an allocation matrix $X^j = VA^j \times VS$, where $X^j[i,k] = x^j_{i,k}$. The matrix X^j represents the allocation part of the mapping M^j of A^j and still leaves the routing to be solved.

Routing. We now present a way of describing the routing for an application A^j. From an application point of view the communication between application components is described by $(a_i, a_j) \in EA^j$. The routing of the communication of $(a_i, a_j) \in EA^j$ on a path between $(s_k, s_l) \in ES$ can be described by $mr^j : EA^j \rightarrow ES$. We can derive such a function for all $EA^j \in \mathcal{A}$.

For the routing of (a_i, a_j) of EA^j on the path $p_{kl_v} \in P_{k,l}, v = 1, \ldots, n$ and $P_{k,l} \in P$ we define a parameter $y^j_{(a_i,a_j),p_{kl_v}}$, where:

$$y^j_{(a_i,a_j),p_{kl_v}} = \begin{cases} 1, \text{if } (a_i, a_j) \in EA^j \text{ is routed on } p_{kl_v} \in P_{k,l}, \\ 0, \text{otherwise.} \end{cases}$$

Every path $p_{kl_v} \in P$ can be projected on a set of physical links of S, i.e.,by the earlier derived function $F : P \rightarrow ES$. Thus $p_{kl_v} = \{es_g, \ldots, es_h | es_g, \ldots, es_h \in ES\}$ and $es_g = \{(s_k, s_r) | s_k, s_r \in VS\}$. Using this notation allows us to create a routing matrix $Y^j = EA^j \times ES$, where $Y^j[(a_i, a_j), p_{kl_v}] = y^j_{(a_i,a_j),p_{kl_v}}$. Y^j represents the routing part of the mapping M^j of application A^j.

Overall mapping. The mapping is determined by the set of routing vectors $Y = \{Y^1, \ldots, Y^n\}$, and the set of allocation vectors $X = \{X^1, \ldots, X^n\}$ and $Z = \{Z^1, \ldots, Z^n\}$, thus $M^j = \{Z^j, X^j, Y^j\}$. We define mapping as follows: $\mathcal{M} = \bigcup^{\mathcal{A}}_{\forall A^j} M^j$.

Using the above view we can state that a *configuration* of a distributed environment consists of the structures and mapping of the distributed system, i.e.,the constellation of components, their physical interconnection and their mapping on each other... As a consequence, a *reconfiguration* is the changing of the structure or the mapping. For the creation of alternative configurations we intend to use application placement procedures as proposed in [4, 11, 30, 31, 33]. Formally, the distributed system configuration, Ω, is a function of the task components, the application and system structure and the mapping, i.e.,$\Omega = F(TC, \mathcal{A}, S, \mathcal{M})$.

4.2 A graph model of IN/ANSA

In this section we describe our experimental environment using the graph notation as presented above. First we start with the presentation of the available components for our distributed application (see also Figure 6):

$TC = \{T^1\}$
$AC = \{\text{GC}, \text{SC}, \text{NTC}, \text{DBC}, \text{BC}\}$
$SC = \{sun2, sun3\}$
$NL = \{nl\}$
$P = \{p_{2,3_1}\}$

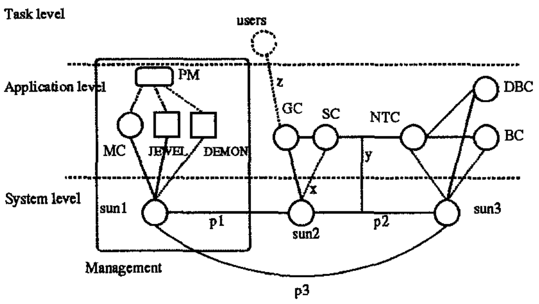

Fig. 6. The graph-oriented view of the distributed environment

In this description we left out the aspects of the management level, because they are not assumed to have influence on the performance and therefore do not contribute to the performability model to be created. *sun1* (see Figure 6), has been reserved for the management level activities. The user component is included in the GC as said before, still we present it also as a separate component at task level for parameterization reason to become clear below.

Below we describe the task, application and system level of the environment using the notation presented in the previous section:

$T^1 = \{\text{USERS}\}$
$\mathcal{A} = \{\langle VA^1, EA^1\rangle\}$
$A^1 = \langle VA^1, EA^1\rangle$
$VA^1 = \{\text{GC}, \text{SC}, \text{NTC}, \text{DBC}, \text{BC}\}$
$EA^1 = \{(\text{GC}, \text{SC}), (\text{SC}, \text{NTC}), (\text{NTC}, \text{DBC}), (\text{NTC}, \text{BC})\}$

$S = \langle VS, ES\rangle$
$VS = \{sun2, sun3\}$
$ES = \{(sun2, sun3)\}$
$p_{2,3_1} = \{(sun2, sun3)\}$

The only aspect that still needs to be described is the mapping $\mathcal{M}$ of the different levels. Because there is only one application, A^1, the mapping remains simple, $\mathcal{M} = \{M^1\}$. For the mapping we use the allocation and routing matrices, $M^1 = \{Z^1, X^1, Y^1\}$.

$$Z^1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}, X^1 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}, Y^1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

This mapping in combination with the application, task and system level components results in a configuration as presented in Figure 6; we can see that the application components NTC, BC and DBC are allocated on one system part and therefore do not use any communication paths as shown by Y^1.

We can change the mapping and thereby create an alternative configuration. For the experiments in Section 7 we used the following two alternative mappings, M^2 and M^3. In the first alternative configuration (represented by $M^2 = \{Z^2, X^2, Y^2\}$) we moved all the application components to the same system part, *sun*2. For this configuration the application does not use the communication paths. In the second alternative configuration (represented by $M^3 = \{Z^3, X^3, Y^3\}$), replicated components are used, i.e.,NTC', DBC' and BC'. For this alternative configuration we also have to change (expand) the set of application components and the application graph. These replicated components are treated as independent components. The corresponding allocation matrices then have the following form:

$$Z^2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}, X^2 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}, Y^2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

$$Z^3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}, X^3 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}, Y^3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

In Figure 6, the relations between the management level and the application components have not been shown, but each of the application components actually has a relation with all the management components.

5 A performability model of IN/ANSA

For the evaluation of an alternative configuration we need a performability model. Therefore, an alternative configuration Ω_{alt} is transformed into a performability model Ψ_{alt}. We do this by replacing each component of T^1, A^1 and S by a predefined stochastic Petri net (SPN) model. We use the mapping, the application and system graphs to create the overall performability model. The resulting performability models are both flexible and relatively easy to solve by current day software tools [13]. In this paper we will deal with the performance aspects of the model only.

In Section 5.1 we present the generic SPN modelling of user, application and system components. The performability model of the experimental distributed environment is presented in Section 5.2.

5.1 The SPN models used to realize the performability model

In this section we present a generic way to transform each component of the distributed environment into an SPN sub-model. We start with the application level, then the system level and finally present how the users are modelled using SPN.

For each operation or service provided by an application component a SPN model component is predefined. In such an SPN model a service is represented by a timed transition. The invocation of a service at the interface by a client is represented by putting a token in the corresponding "service-input place". Resources must be allocated (e.g.,an CPU) and the operation can be performed (the timed transition). In Figure 7 we see (at the right hand side of the arrow) the SPN representation of one operation of a computational object (shown at the left ahnd side of the arrow) or application component. The output of the timed transition, i.e. the operation, is an announcement or an interrogation to another operation (see Figure 8). With an interrogation invocation as output the component will await an answer and continue operation.

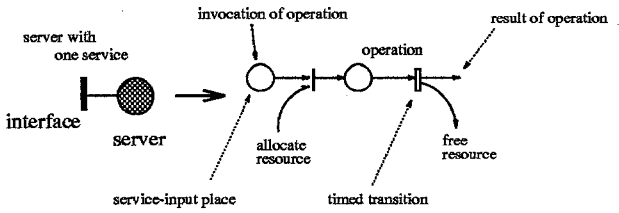

Fig. 7. The SPN representation of an ANSAware service provision

The duration of an operation is represented by a timed transition. These transitions represent the work demanded from the resource, for example the CPU busy time ($1/\mu_{i_v,k}$, the average service time of operation v of application component i on system component k) if the CPU is the scarce resourse. We can estimate these parameters by running and monitoring the component in isolation (one component on a single workstation).
The communication between components can be represented in a similar way as the operations. Per (remote) operation, or communication between two application components allocated to different system parts, a network link must be allocated. The duration of a communication operation is also represented by a timed transition. In this case a timed transition represents the communication time ($1/\mu_{i_v,j,n_i}$, the average communication time of application component j to

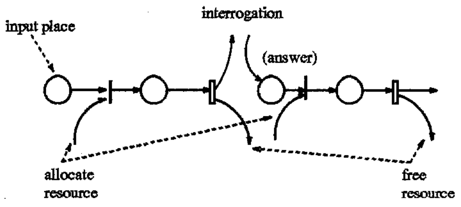

Fig. 8. The SPN representation of an announcement or interrogation operation

application component i for operation v using network link i) per invocation of an operation per network link (see [9]).

The generation of requests by the USERS is modelled as a Poisson arrival process, represented by a single timed transition.

5.2 The performability model of the IN/ANSA environment

In this section we discuss the SPN performability model of the IN/ANSA application. Tools for SPN analysis normally only allow finite state space models. This does not correspond to the experimental environment. However, we can approximate an open model by closed model with a very large customer population. The average *request rate* Λ for an application A^j is modelled by the task component USERS $\in T^j$. In Figure 9 the SPN representation of the distributed environment, using configuration M^1, is given using the predefined SPN models.

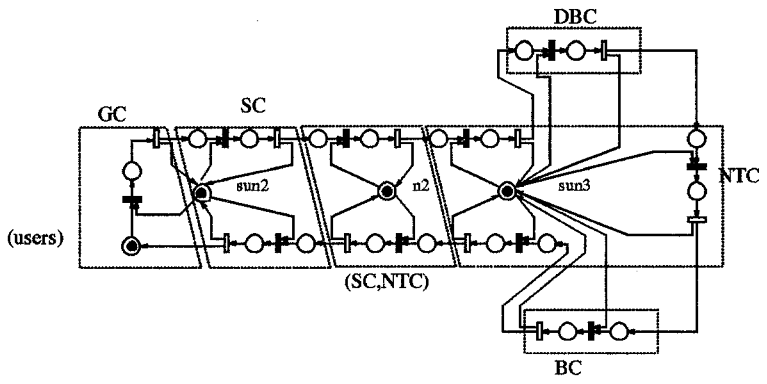

Fig. 9. The SPN model of configuration M^1

For the evaluation of the model we need the labels for the components as presented in [9]. These labels are the transition rates (service rates) of the timed transitions in the SPN model.

6 Monitoring of ANSAware applications

Two different monitoring tools, the DEMON and the JEWEL tool, monitor the experimental environment introduced in Section 3. The DEMON tool [21] monitors and visualizes the functional behaviour and configuration of the ANSAware components (as defined in the set AC) on the system nodes (as defined in the set SC). These can be used to provide the performability model with configuration information. The JEWEL monitoring tool [19] extracts performability indices from the ANSAware environment and visualizes them for each component on a graphical display. The performability indices are used to detect a decrease of QoS and to parameterize the performability model.

In order to provide the monitoring tools with the information needed, the ANSAware application components has to be instrumented with additional code for both monitoring systems. Instrumentation for the DEMON tool is performed automatically by a pre-compiler designed and implemented at PTT Research [17]. Instrumentation for the JEWEL monitoring tool is performed in a generic manner using the ANSAware operations as a reference point to detect relevant events. The implementation of the invocation of an operation is embraced by the two events: *request* and *confirm*. These events are detected by JEWEL and used to derive the turnaround time of an operation. The implementation of the operation is also embraced by two events: *indication* and *response*, these are detected by JEWEL and used to derive the service time of an operation, see Figure 11. A detailed prescription of generic instrumentation for ANSAware is provided in [25].

7 Experiences with monitoring, modelling and evaluation

A performability model of a distributed application can automatically be constructed guided by three input sources (see also Figure 10):

1. *A library of predefined SPNmodels.* For each ANSAware and system component a model has to be available in a library .
2. *Configuration determination.* The configuration has to be obtained from the system to construct the model from the predefined model components in the library. The DEMON monitor provides this configuration information.
3. *Performability indices determination.* We use the performability monitoring measurements provided by JEWEL to determine the transition rates of the timed transitions in the SPN.

The method of tuning the performability model, provided in [9], obtains the transition rates for the SPN from the requirements of the components and the capacities of the system nodes. A major drawback of this method is the required *a priori* determination of the requirements and capacities. Because the source code of the ANSAware components is processed by several pre-compilers and linked with library functions, exact requirements of the components with respect to processing workload, communication workload, memory access, etc.

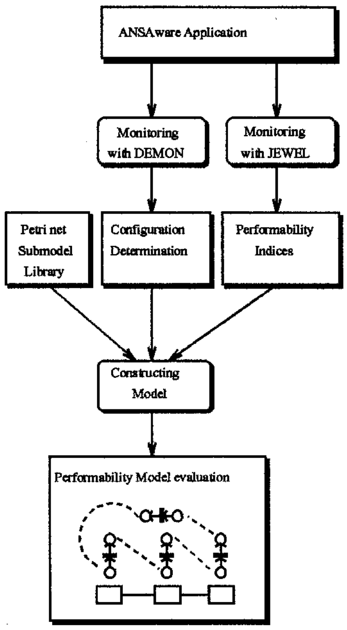

Fig. 10. The performability model is constructed from three input sources

are hard to assess. Capacities of the system nodes may be exactly specified by the manufacturers, but mechanisms like memory caching or disk access cause dynamically changing capacities of the system nodes. Therefore, we have used a more practical approach to parameterize the performability model, guided by the measurements provided by the JEWEL monitoring tool. The transition rates can be obtained by measuring the service times of the individual components. In Figure 11 a timing diagram is depicted containing the monitored time-stamps of the events: request, confirm, indication and response.

The service times can be derived from these measurements under minimal load. No queueing will occur under minimal load, so the residence time of a component will be equal to the service time of that component decreased by the residence times of the interrogation operations invoked during the service provisioning and the encountered communication delays:

$$T_j = \frac{1}{R_j - \sum_{i \in K} R_i - \sum_{i \in K} C_i}$$

where T_j is the transition rate of the sub-model of component j, R_j is the average turnaround time of component j and K is the set of operations invoked (as an interrogation) by component j. R_i is the average turnaround time of operation i and C_i the average communication delay to component i. As an example consider

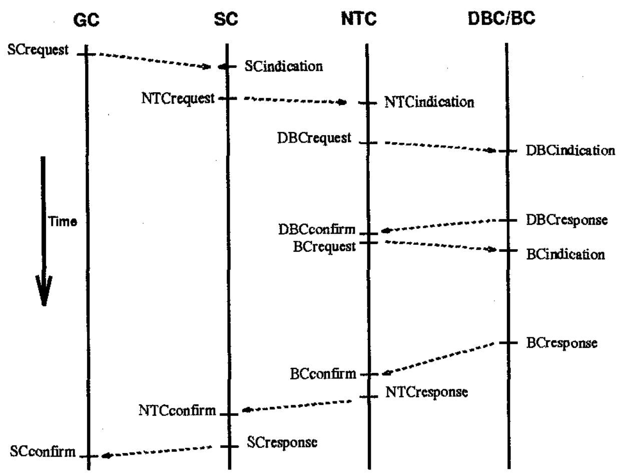

Fig. 11. The timing diagram for one configuration

the service time of the NTC in Figure 11. The service time of component NTC can be derived from the residence time of NTC (NTCresponse - NTCindication) decreased by the residence times of DBC and BC and the communication delays (differences between request and indication and the differences between response and confirm). In this way service times and communication delays can be derived from the measurements depicted in the diagram. A drawback of this method is that for each combination of components and system nodes a measurement under minimal load has to be done to obtain the residence time without queueing. A major advantage of this method, however, is the higher level of abstraction maintained, i.e. the capacities of the system nodes and the requirements of the components are implicitly incorporated.

We now discuss some comparative results from the modelling and monitoring. The IN/ANSA application has been monitored using different (alternative) configurations. The performability model has been parameterized with statistics (averages) over the measurements, obtained by monitoring the different configurations under minimal load. This leads to one set of parameters applicable for all configurations. The SPNP implementation of the performability model has been verified with the performability indices actually measured by the JEWEL monitor under various workloads.

In Table 1 the monitored results for the three different configurations are presented in comparison with the values calculated by SPNP. We see that the model results, under minimal load, come very close to the actually measured values. Notice that these results are obtained using a very simple performance model, only taking into account application components and CPU possession.

Finally, we compare the measured results with the model evaluation results under higher load. Note that the models parametrization is the same as for the

config.		Time in ms. Monitored	SPNP	Difference
M^1	Turnaround time of GC	112	113	+0.9%
	Turnaround time of SC	91	88	-3.4%
	Turnaround time of NTC sun3	76	73	-4.1%
M^2	Turnaround time of GC	157	157	0%
	Turnaround time of SC	132	132	0%
	Turnaround time of NTC sun2	103	103	0%
M^3	Turnaround time of GC	138	135	-2.2%
	Turnaround time of SC	115	110	-4.5%
	Turnaround time of NTC sun2	106	103	-2.9%
	Turnaround time of NTC' sun3	73	73	0%

Table 1. Evaluation SPNP model with monitoring results under minimal load for configurations M^1, M^2 and M^3.

minimal load case. The configurations investigated are M^1, M^2 and M^3. Due to scheduling strategies of ANSAware the approximations for the turnaroundtime of the "internal" components, i.e., NTC, SC, BC and DBC are not comparable to the SPNP results. More important for the performance, however, is the QoS provided to the user, i.e.,the turnaroundtime of the complete application, which is equal to the turnaround time of GC. We therefore address this measure.

The results of the SPNP model and the measurements are graphically depicted in Figure 12. For each configuration eight monitoring sessions were conducted for different workloads (λ). The results of the SPNP model are reasonably good (less than 10% error) when the load is low to moderate. When the load increases, however, the monitoring results differ substantially from the results calculated by SPNP. The workload range of our interest is the moderate range were the turnaround time does not exceed the requested QoS. If the QoS is violated (or the turnaroundtime has increased significantly) the performability manager is triggered and runs the Performability Model for alternative configurations.

Furhter research is necesary to estimate the level of confidence we can put in our models.

8 Discussion and future work

In this paper we focused on the modelling aspects of model-based dynamic QoS management by a PM. We presented an ANSAware-based experimental distributed environment in which the modelling and evaluation aspects are totally automated. We proposed a generic modelling strategy in which the structure of the client/server and the computational model of the ANSAware computing platform are used. This structure allows for a generic transformation of the com-

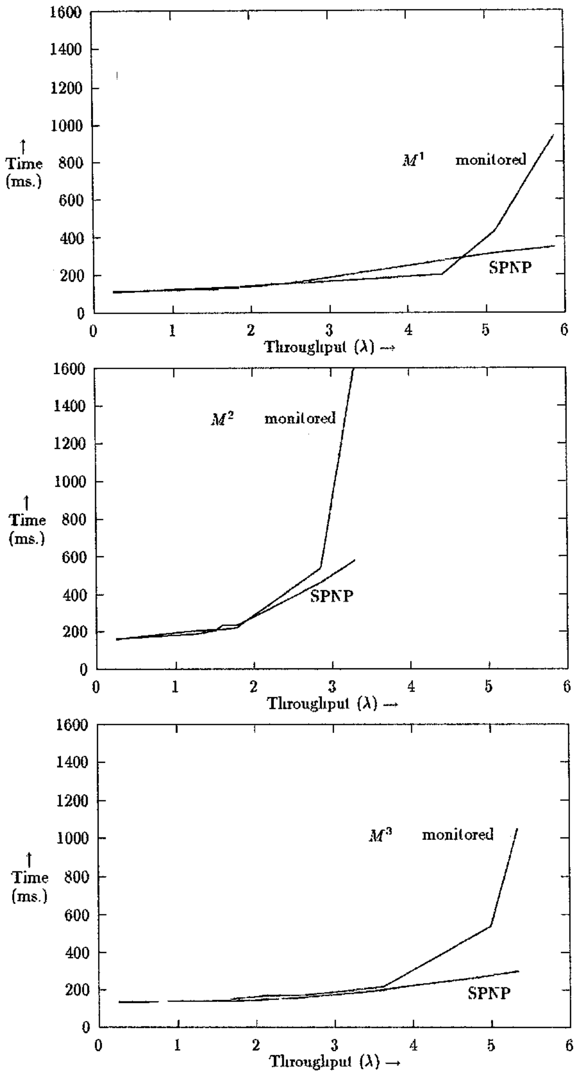

Fig. 12. Evaluation SPNP model with monitoring results for configurations M^1, M^2 and M^3.

putational models into performability models using predefined SPN models. The PM receives information about the ANSAware-based application from a distributed monitoring process based on JEWEL and DEMON. With this information, and the SPN model library of ANSAware applications, the PM automatically constructs an overall SPN performability model which is subsequently used for the determination of the provided Quality of Service (QoS).

The modelling as presented is based on our general view on distributed systems. This view only bears four levels of which we modelled three, which might not be sufficient. For example the scheduling activities, i.e. the use of threads and tasks, in ANSAware are not modelled at the moment which introduces inaccuracies. Because ANSAware is implemented on top of the system level (and even partly resides in the management level), the introduction of a middle-ware level might be necessary.

We showed the feasibility of the proposed PM by presenting some operational results. More work, however, will be necessary to make the PM fully operational. The process of automatically creating alternative configurations, selecting the best and make it operational is not completely automated yet. Currently we are working on proper mapping algorithms for the creation of the alternative configurations.

The required on-line and therefore necessarily fast evaluation of the created SPN models also requires further study. Currently we are experimenting with MVA algorithms (thereby ignoring the "simultaneous resource possession" aspects) and the use of closed-form solutions for the SPNs [25].

The monitoring process is realised using two monitoring tools. In a future environment the use of one monitoring tool is preferred because of the interference of the monitoring process with the monitored applications. The current experience with generic monitoring shows satisfying results which makes it applicable for further use.

In this paper we mainly adressed pure performance issues of the performability manager. The use of replicated components and the evaluation of the models w.r.t. "real" performability measures, i.e.,including dependability aspects, will be subject of further study. We also intend to use the performability manager as a conceptual framework for the study of resource control issues in multimedia conferencing systems.

References

1. APM Ltd., Cambridge, U.K. *ANSA : An Engineer's Introduction to the Architecture*, November 1989.
2. R.L. Bennett and G.E. Policello II. Switching Systems in the 21st Century. IEEE Communications Magazine: *Feature Topic: Toward The Global Intelligent Network*, 31(3):24–30, March 1993.
3. Y. Berders and P. Dickman, editors. *Workshop on Dynamic Object Placement and Load Balancing in Parallel and Distributed Systems.* The Sixth European Conference on Object-Oriented Programming, ECOOP'92, 1992.

4. N.S. Bowen, C.N. Nikolaou, and A. Ghafoor. On the Assignment Problem of Arbitrary Process Systems to Hetrogeneous Distributed Computer Systems. *IEEE Transactions on Computers*, 41(3):257–273, March 1992.
5. M. Calzarossa and G. Serazi. Workload Charaterization: A Survey. *Proceedings of the IEEE*, 81(8):1136–1150, August 1993.
6. B. Carré. *Graphs and Networks.* Clarendon Press, Oxford, 1979.
7. N.M.van Dijk, B.R. Haverkort, and I.G. Niemegeers. Guest editorial: Performability Modelling of Computer and Communication Systems. *Performance Evaluation*, 14(3-4):61–78, February 1992.
8. ETSI. Network Aspects (NA); General aspects of quality of service and network performance in digital networks, including ISDN. Technical Report ETR 003, ETSI, 1990.
9. L.J.N. Franken and B.R.H.M. Haverkort. The Performability Manager. IEEE Network: The Magazine of Computer Communications *Special Issue on Distributed Systems for Telecommunications*, 8(1), Januari 1994.
10. J.J. Garrahan, P.A. Russo, K. Kitami, and R. Kung. Intelligent Network Overview. IEEE Communications Magazine: *Feature Topic: Toward The Global Intelligent Network*, 31(3):30–38, March 1993.
11. A. Gersht and R. Weihmayer. Joint Optimization of Data Network Design and Facility Selection. *IEEE Journal on Selected Areas in Communications*, 8(9):1667–1681, December 1990.
12. S. Hariri and C.S. Raghavendra. Distributed Functions Allocation for Reliability and Delay Optimization. *Proceedings of the Fall Joint Computer Conference (IEEE)*, pages 344–352, 1986.
13. B.R. Haverkort and K.S. Trivedi. Specification and Generation of Markov Reward Models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.
14. B.R.H.M. Haverkort. *Performability Modelling Tools, Evaluation Techniques, and Applications.* PhD thesis, University of Twente, 1990.
15. K. Kant. *Introduction to Computer System Performance Evaluation.* McGraw-Hill, Inc., 1992.
16. S. Kheradpir, W. Stinson, J. Vucetic, and A. Gersht. Real-Time Management of Telephone Operating Company Networks: Issues and Approaches. *IEEE Journal on Seclected Areas in Communications*, 11(9):1385–1403, December 1993.
17. H. Korte. Visualising ANSAware Programs with EXP93. Technical report, PTT Research, the Netherlands, unpublished, June 1993.
18. J. Kramer, editor. *Proceedings of the International Workshop on Configurable Distributed Systems.* Computing Control Division of the Institution of Electrical Engineers, IFIP, Imperial College of Science, Technology and Medicine, IEE, March 1992.
19. F. Lange, R. Kroeger, and M. Gergeleit. JEWEL: Design and Implementation of a Distributed Measurement System. *IEEE Transactions on Parallel and Distributed Systems*, 3(6):657–671, November 1992.
20. Y.H. Lee and K.G. Shin. Optimal Reconfiguration Strategy for a Degradable Multimodule Computing System. *Journal of the ACM*, 34(2):326–348, April 1987.
21. MARI Computer Systems Ltd. *DEMON V3.0 User's guide and Reference manual*, 1993.
22. L. Mejlbro. QOSMIC-Deliverable D1.3C: QoS and Performance Relationships. Deliverable QOSMIC R1082, RACE, 1992.

23. J.F. Meyer. Performability Evaluation of Telecommunication Networks. In Network Teletraffic Science for Cost-Effective Systems and ITC-12 Services, editors, *M. Bonatti*, pages 1163–1172. IAC, Elsevier Science Publishers B.V. (North Holland), 1989.
24. J.F. Meyer. Performability: a Retrospective and some Pointers to the Future. *Performance evaluation*, 14(3-4):139–156, Februari 1992.
25. R.H. Pijpers. Performability Monitoring and Modelling of ANSAware Environments. M.Sc. thesis, University of Twente, the Netherlands, December 1993.
26. R. Pooley and J. Hillston, editors. *Proceedings of the Sixth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation.* University of Edinburgh, Athony Rowe Ltd, Chippenhame, Wiltshire, September 1992.
27. Project JTC1.21.43. Reference Model for Open Distributed Processing. Draft Recommendation X.901: Basic Reference Model of Open Distributed Processing Part 1: Overview and Guide to use reference SC21 N7053, , 1993-1-28.
28. QOSMIC. General Aspects of Quality of Service and System Performance in IBC. Deliverable RACE D510, RACE, 1991.
29. W. Rosenberry, D. Kenney, and G. Fisher. *Understanding DCE.* O'Reilly & Associates, Inc, 1992.
30. S. M. Shatz, J. Wang, and M. Goto. Task Allocation for Maximizing Reliability of Distributed Computer Systems. *IEEE Transactions on Computer*, 41(9):1156–1168, December 1992.
31. S.M. Shatz and J. Wang, editors. *Tutorial: Distributed Software Engineering.* IEEE Computer Society, Press, 1989.
32. K.G. Shin, C.M. Krishna, and Y. Lee. Optimal Dynamic Control of Resources in a Distributed System. *IEEE Transactions on Software Engineering*, 15(10):1188–1197, October 1989.
33. N.G. Shivaratri, P. Kreuger, and M. Singhal. Load Distributing for Locally Distributed Systems. *IEEE Computer*, 25(12):33–44, December 1992.
34. International Telecommunication Union. General Characteristics of International Telephone Connections and Circuits. Red Book Fsc. II.1, CCITT, 1985.
35. International Telecommunication Union. Telegraph and Mobile Service and Quality of Service. Blue Book Fsc. II.4, CCITT, 1989.
36. Studygroup XI. Q.1200, Draft recommendations. Technical report, CCITT, 1991.