The Performability Manager

By dynamically reconfiguring distributed systems, the performability manager contributes to a more effective use of system components and prevents QoS degradation.

Leonard J. N. Franken and Boudewijn R. Haverkort

Future broadband distributed systems will have high functionality, availability, reliability, and performance requirements. Their modular structure, however, can be exploited to perform run-time enchancements to increase functionality and to recover from a decrease in availability, reliability, or performance.

Efficient and effective use of the resources and guaranteeing a desired level of quality of service (QoS) in a distributed environment are complex but important tasks, not only at system initiation but also at run time. In this paper we describe the performability manager, a distributed system component that contributes to a more effective and efficient use of system components and prevents QoS degradation.

The performability manager dynamically reconfigures distributed systems whenever needed, to recover from failures and to permit the system to evolve over time and include new functionality. Large systems require dynamic reconfiguration to support dynamic change without shutting down the complete system [1-4].

A distributed system monitor is needed to verify QoS. Monitoring a distributed system is difficult because of synchronization problems and minor differences in clock speeds (see for example [5]).

Before we describe the functionality and the operation of the performability manager (both informally and formally), we give an overview of our earlier and other related work in the distributed systems area. In the concluding section we put forward some future research issues. Throughout the paper we will illustrate the approach by an example distributed application: an ANSAware-based number translation service (NTS), from the intelligent networks (IN) area.

LEONARD J. N. FRANKEN is a distributed-systems engineer with PTT Research Groningen.

BOUDEWIJN R. HAVERKORT is an assistant professor in the Tele-Informatics and Open Systems group of the department of computer science at the University of Twente.

Related Distributed-Systems Work

For describing the functionality of the performability manager we will use the "common framework" as presented in [6], which describes the components of a distributed system (Figs. 1 and 2):

Task: a certain amount of work to be realized by the system.
Application: the realization of a certain amount of work on a system (part).
System part: an independent component, capable of independently executing an application.
Network: the complex of connections between system parts available to the system parts for communications.
Management: hardware or software components that take care of the mapping of tasks on applications and of applications on system parts, in order to achieve an efficient and effective use of the applications, system parts, and the network.

An example of an application that uses the above framework is described in the box on the next page.

The distributed nature of the systems we study makes their evaluation a complex task. Mapping applications on system parts is driven by functional and quantitative arguments, many of which can be viewed as reasons for distribution [7, 8]:

- Geographical spread of users.
- Geographical spread of information.
- Performance increase.
- Reliability increase.
- Availability increase.

The quantitative reasons, i.e., the performance, reliability, and availability increase can be viewed from two sides: a user point of view and a provider point of view. A provider wants an efficient and effective use, such as high utilization, of the distributed components. To users, quantitative aspects are reflected by the QoS. The QoS describes the user-perceived performance [9-12]. The QoS can be divided into subjective and objective elements. Subjective QoS is user-oriented and hard to quantify and measure; objective QoS, the QoS we refer to in this paper, can be measured. 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), the generic term for provider-visible performance parameters [13], are quantitative parameters that indicate how well the system (service) is performing. Between the objective QoS parameters and the SPPs there

0890-8044/94/$04.00 © 1994 IEEE

exists a one-to-one mapping [9, 10]. 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).

Because 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; modeling and evaluation techniques that can handle the combined aspects are required. The performability manager uses performability analysis because it provides a means to model and evaluate distributed systems with respect to their QoS [6, 14-17].

Functional aspects of distributed systems are generally modeled using so-called formal description techniques (FDTs). The combined evaluation of functional and quantitative aspects requires integrated modeling techniques and tools; however, no well-established modeling frameworks have yet been defined for this purpose [6].

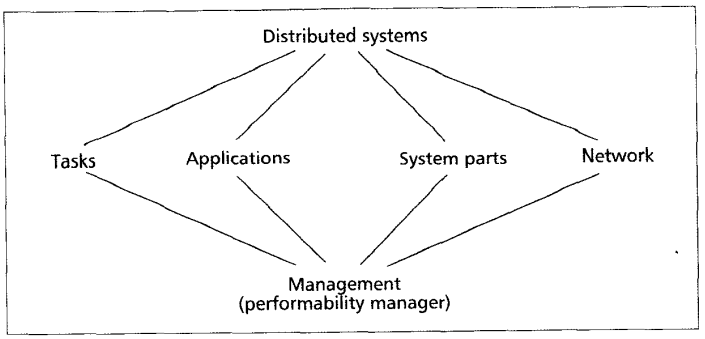
The performability manager maintains the required QoS by dynamic reconfigurations. This requires that facilities for dynamic reconfiguration should be available in the distributed system. Such facilities would include access transparency, concurrency transparency, federation transparency, location transparency, migration transparency, and replication transparency [18]. These facilities can be realized at several levels in a distributed system:

- The operating system level.
- The middle-ware level (at the level of computing platforms or configuration languages).
- The application level.

The current trend in distributed systems is to provide these facilities via computing platforms. These platforms allow for a heterogeneous distributed system that is transparent to the application programmer. Examples of such middle-ware facilities are the configuration languages Gerel, Conic, Argus, Rex, Darwin [1], and the computing platforms ANSAware [19] and DCE [20]. ANSAware is the computing platform on top of which we implement the performability manager and the NTS application. We give more information on ANSAware in the sidebars entitled "The ANSAware Infrastructure" and "An ANSAware-based NTS."

Computing platforms and configuration languages provide users with more or less the same set of functionalities that can be used for dynamic and/or static configuration and reconfiguration. Examples of the use of these facilities for qualitative configuration management are described by Cole [21] and Dean [22]. Cristian [23] presents an approach for an availability manager, which guarantees the availability of the applications using replicated components.

Although the goal of reconfiguration is to maintain the desired QoS or to include new functionality in the distributed system, most of the effort in the area of dynamically reconfigurable 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, however, guides reconfiguration by using a model-based optimization procedure. Performability evaluation is used in the optimization procedure. A similar, but less general and less "automatic" approach towards resource control has been proposed by Lee and Shin [24, 25]



■ **Figure 1.** *The components of a distributed system.*

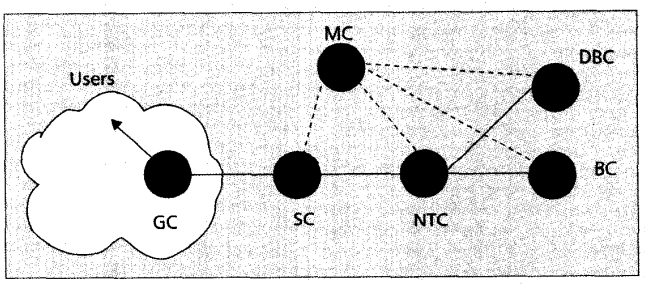
Also related to our work is the area of optimal system design [26, 27], dynamic load balancing [28, 29], and task allocation in distributed systems [30]. Examples of these are presented by Bowen [31] and Hariri [32]: Bowen presents a study on process allocation in heterogeneous distributed systems and compares a heuristic-algorithm with an LP-solution, whereas Hariri presents an algorithm which takes care of optimizing reliability and communication delay. Both approaches focus only on a single quantitative system aspect (either performance or reliability) instead of on a combination of the two aspects, as we propose.

The Number Translation Service

For the number translation service (NTS), as provided in intelligent networks [46, 47], end users submit requests or tasks for the application at a certain rate. Because we do not have real users in our experimental application, we use a component that mimicks user behaviour, the so-called Generating Component (GC). The GC generates the calls for the NTS. The NTS is provided by the following application components (Fig. 2):

- The Selection Component (SC): this component selects a service using the contents of the requests it receives (in this example, the number translation service).
- 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.
- The Database Component (DBC): this component receives requests for specific numbers. It will fetch the number from a disk and return the number to the component that requested the number.
- The Billing Component (BC): this component receives requests for the preparation of a billing record.

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



■ **Figure 2.** *The experimental NTS application.*

Another important related issue is the monitoring of distributed systems [5]. To be able to detect degradations of the QoS, the distributed system must be monitored to detect performability degradation. The monitoring output can also be used to feed the distributed system model with more accurate parameters.

The Performability Manager in Perspective

Because a reconfiguration must be performed with great care, the performability manager uses a model of the current distributed system to prepare a reconfiguration. For deriving a model suitable for evaluation, a distributed system must be described in a way that makes the creation and evaluation of models of alternative configurations possible. Therefore, we use the framework in the previous section to look at a distributed system at four levels: tasks, applications, system parts and the network, and management. Each level consists of components and their relations. The relations can exist between the different levels as well as between the components at one level (mutual relations).

The ANSAware Infrastructure

ANSAware is a suite of programs that allows users to write applications suitable for heterogeneous distributed environments (see also Fig. 6 in the box on the next page, although the possible heterogeneity is not directly apparent there). It is based on ISO-ODP, an emerging ISO standard for Open Distributed Processing [18]. 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 multivendor 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 (Fig. 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:

- By interrogation, in which the invoking client waits for the server to perform the operation and return the result (similar to an rpc);
- 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).

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.

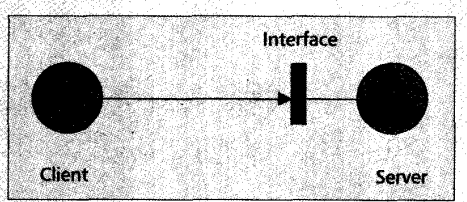


Figure 3. *A client object and server object with its interface.*

Task level — The performability manager views tasks 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 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. In addition to their relations to the task and system level, application components have a mutual relation that represents the communication between the application components. Therefore, the performability manager views an application as a structure of application components and their mutual relations (communication). A single application structure is described by 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 that is active in the current distributed system is denoted by $\mathcal{A}$. Thus, we have $A^j \in \mathcal{A}$, and $\mathcal{A} = \{\langle VA^j, EA^j \rangle \mid j=1, \ldots, n\}$, where n is the number of applications.

System level — System level is also seen as composed of components: system parts $s_k \in SC$ with $SC = \{s_1, \ldots, s_w\}$, and network links, $n_l \in NL$ and $NL = \{n_1, \ldots, n_t\}$ which provide the mutual relations between the system parts. The distributed system structure is described as a graph $S = \langle VS, ES \rangle$ with $VS \subseteq SC$ and $ES \subseteq VS \times VS$ and $F : ES \rightarrow NL$.

Paths are also defined at the system level. A path is a finite sequence of network links (arcs or edges) between any two system parts (nodes); that is, a finite sequence of links in which the terminal node (system part) of each link coincides with the initial node of the following link [33]. P is 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 $p_{kl_v} = \{(s_k, s_h)(s_h, s_f) \ldots (s_g, s_l) \mid (s_k, s_h), \ldots, (s_g, s_l) \in ES\}$. Every path $p_{kl_v} \in P$ can be mapped on a set of links of S, by the function $F : P \rightarrow ES$.

In addition to their mutual relation, the system parts have a relation to the application level. The definition of a path is 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 level — The management level is orthogonal to the other levels. It consists of management tasks, applications and systems and is structured in a similar way as the "normal" distributed system.

Mapping — The mapping is the logical allocation of higher levels to lower levels: the allocation of application components on system parts and the routing of the communication over the network links. This is also reflected in Fig. 4 where the mutual relations are represented by solid 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. Although the other allocations are subject to change, the task components are always allocated to the same application component of an application.

A mapping function is described as M^j for each A^j. The allocation and routing of a mapping can be described in a more formal way.

Allocation — There are two types of allocations: the allocation of task components to application components and the allocation of application components to system parts. The allocation of task components to application components is described 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 components $s_k \in S$, the mapping can be described by $ma^j : VA^j \rightarrow VS$. You 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 you 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.

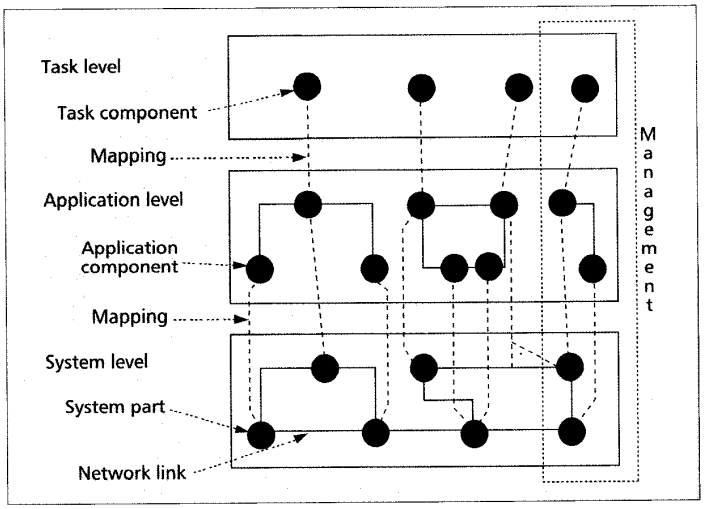
Routing — We now present a way of describing the routing for an application A^j. From an application point of view, the communication between application components is described by $(a_i, a_j) \in EA^j$. The routing of the communication of $(a_i, a_j) \in EA^j$ on a path between $(s_k, s_l) \in ES$ is described by $mr^j : EA^j \rightarrow ES$. We 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_{i,j,kl_v}, where:

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

Every path $p_{kl_v} \in P$ can be projected on a set of physical links of S; that is, by the earlier derived function $F : P \rightarrow ES$. Thus, $p_{kl_v} = \{es_g, \ldots, es_h \mid es_g, \ldots, es_h \in ES\}$ and $es_g = \{(s_k, s_r) \mid s_k, s_r \in VS\}$. Using this notation allows us to create a routing matrix $Y^j = EA^j \times ES$, where $Y^j[\{(a_i,a_j),p_{kl_v}] = y^j_{i,j,kl_v}$. 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 $M^j = \{Z^j, X^j, Y^j\}$. The overall mapping is defined: $\mathcal{M} = \cup^{\mathcal{A}}_{\forall A^j} M^j$.

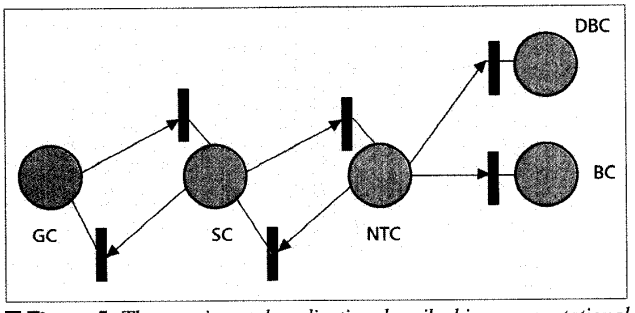
Using the view just presented, we can state that a configuration of a distributed environment consists of the structures and mapping of the distributed system; that is, the constellation of components, their physical interconnection, and their mapping on each other. As a consequence, a reconfiguration 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 [26, 29, 30, 31,



■ **Figure 4.** *The distributed system configuration.*

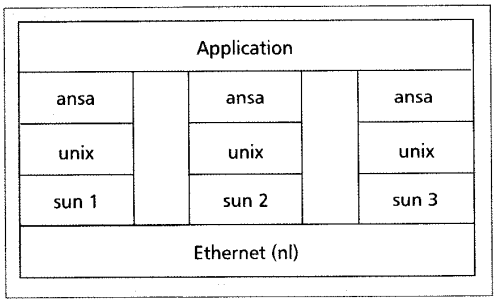
An ANSAware-based NTS

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 (see Fig 5). 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, which are interrogations.



■ **Figure 5.** *The experimental application described in a computational form.*

For the experimental application, we use a small distributed system consisting of three SUN SPARC workstations connected by an Ethernet as depicted in the two lower layers of Fig. 6. Within this experimental distributed environment we use two monitors: DEMON, the Distributed Environment MONitor [48] is used to visualize the structure of the experimental distributed environment and JEWEL is used to do performance measurements in the experimental distributed environment [5].



■ **Figure 6.** *The experimental distributed environment.*

34]. Formally, the distributed system configuration, Ω, is a function of the task components, the application and system structure and the mapping: $\Omega = F(TC, A, S, \mathcal{M})$.

In the sidebar entitled "Formal Description of the ANSAware-based NTS," we used the notation presented in this section to describe the experimental NTS application mentioned in the previous sidebars.

Formal description of the ANSAware-based NTS

We can describe the NTS using the graph notation as presented before. The available components are:

$TC = \{T^1\}$ $\quad P = \{p_{2,3_1}\}$

$SC = \{sun2,sun3\}$ $\quad NL = \{n1\}$

$AC = \{GC,SC,NTC,DBC,BC\}$

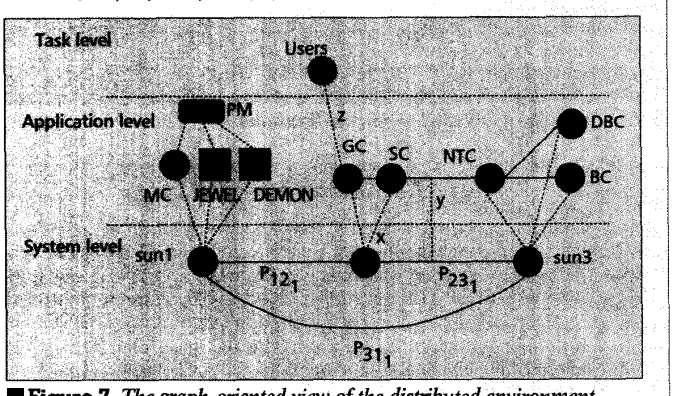


Figure 7. *The graph-oriented view of the distributed environment.*

We left the management level out of the description because it is not assumed to have influence on the performance and therefore does not contribute to the performability model to be created. Sun 1 (Fig. 7) is reserved for management-level activities. Although the user component is included in the GC, we present it as a separate component at task level for parameterization reasons.

The task, application and system level of the environment using the presented notation is as follows:

$T^1 = \{USERS\}$ $\quad S = \langle VS, ES\rangle$

$A = \{\langle VA^1, EA^1\rangle\}$ $\quad VS = \{sun2,sun3\}$

$A^1 = \langle VA^1, EA^1\rangle$ $\quad ES = \{(sun2,sun3)\}$

$VA^1 = \{GC,SC,NTC,DBC,BC\}$ $\quad p_{2,3_1} = \{(sun2,sun3)\}$

$EA^1 = \{(GC,SC),(SC,NTC),(NTC,DBC),(NTC,BC)\}$

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, $\mathcal{M} = \{M^1\}$. We use the allocation and routing matrices, $M^1 = \{Z^1, X^1, Y^1\}$:

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

This mapping in combination with the application, task, and system level components results in a configuration as shown in Fig. 7, where you 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. Although the relations between the management level and the application components have not been shown, each of the application components actually has a relation with all the management components.

Annotation with Service Performance Parameters

In this section we present the set of service performance parameters (SPPs) used to estimate the QoS of a distributed system. The set of SPPs is split into performance and reliability parameters. We annotate the graph models described in the previous section with these parameters (see also [27, 33, 35]).

For the parameterization of the model, we expand our notation of an application component. An application component a_i can perform one or more operations (use services): $o_u \in a_i$ with $a_i = \{o_1, \ldots, o_n\}$, the set of operations performed by a_i. Operation (or service) o_u of a_i is represented by a_{io_u}.

Let $v_{a_{io_u}}$ be the average processing workload requirement of operation o_u of application component a_i (for example, the processing requirements of an application component can be expressed as the number of instructions to be executed).

A system part s_k has a processing capacity expressed by γ_{s_k}, (for example, the capacity of a cpu can be expressed as the number of instructions executed per second). The mean service rate of s_k with respect to a_{io_u} is:

$$\mu_{i_u,k} = \frac{v_{a_{io_u}}}{\gamma_{s_k}} \tag{1}$$

The average communication workload requirement, the number of bytes transferred for communication between application components per invocation of operation a_{io_u} by a_j can be expressed as $v_{(a_j, a_{io_u})}$.

A path p_{kl_v} has a communication capacity expressed by γ_{pkl_v} (for example, expressed as the number of bytes that can be transferred per second). The mean service rate of p_{kl_v} w.r.t. (a_j, a_{io_u}) as:

$$\mu_{i_u,j,kl_v} = \frac{v_{(a_j, a_{i_{o_u}})}}{\gamma_{p_{kl_v}}} \tag{2}$$

The communication work capacity of a path is based on the individual capacities offered by the network links.

The above are generic terms to express the load incurred by an application component and the load of communication between two application components as well as the capacity of system parts and paths. Using these, we can annotate or label the earlier defined application and system graphs with the parameters we previously defined.

For each o_u of $a_i \in A^j$ we can derive the average number of invocations per task submitted to A^j [36-39], which is expressed as λ^j_{aj,aio_u}. The average request rate Λ^j_i for an application A^j is given by task component $t_d \in T^j$.

Using γ_{s_k}, v_{aio_u} and λ^j_{aj,aio_u}, for example, we can derive the utilization of p_{kl_v} by $a_i \in A^j$ as:

$$\rho^j_{i,s_k} = \sum_{\forall o_u}^{a_i} \frac{\sum_{\forall a_j}^{A^j} \lambda^j_{a_j, a_{i_{o_u}}}}{\mu_{i_u,k}} \tag{3}$$

Using γ_{pkl_v}, $v_{(aj,aio_u)}$ and λ^j_{aj,aio_u}, we can derive the

utilization of p_{kl_v} by $a_i, a_j \in A^j$ for operation o_u of a_i as:

$$\rho^{j}_{i,j,p_{kl_v}} = \sum_{\forall o_u}^{a_i} \frac{\lambda^{j}_{a_j, a_{io_u}}}{\mu_{i_u, j, kl_v}} \quad (4)$$

For the reliability aspects it is assumed that only the hardware can fail and can be repaired [40]. Therefore, we only have to assign failure and repair rates to system parts and the paths between them. For each system part s_k we define the failure and repair rate of that system part as f_{sk} and r_{sk}, respectively. The failure and repair rate of a path p_{kl_v} is defined as f_{pkl_v} and r_{pkl_v}, respectively.

Performability Manager Functionality and Environment

The performability manager should become active whenever the QoS is violated or when a degradation of performance, reliability or availability occurs. We will describe some of these situations in more detail below.

A task initiated by a user can either be completed using already running applications or a new application must be created and placed in the system. This leads to two situations in which a reconfiguration can occur.

In the first situation, there is only an increase of the workload for the existing distributed system configuration. An increase of the workload might lead to a violation of the QoS of other tasks. When this is the case, a reconfiguration by the performability manager of the distributed system might be necessary so that the QoS is restored and remains guaranteed. The second situation leads to a change in the distributed system configuration, which consequently leads to a distributed system reconfiguration.

Another situation in which the support of the performability manager is desired is when a fault occurs in the distributed system that leads to a violation of the QoS.

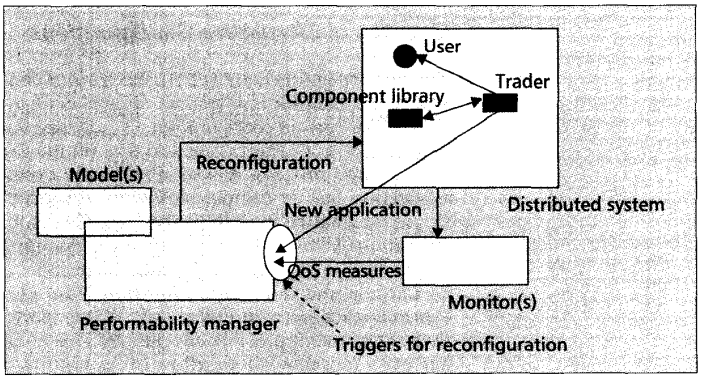
Reconfigurations can either be performed statically (stop the distributed system, perform the reconfiguration, and start the system again), or dynamically (on the fly). The performability manager we present in this paper focuses on dynamic reconfiguration.

More generally, the performability manager must perform reconfigurations of the distributed system configuration, whenever necessary to recover from and prevent degradation of QoS.

To fulfill its task, the performability manager needs supporting facilities. A global description of these facilities is given in Fig. 8; the performability manager is situated between them.

In future environments, users must be able to compose new applications. Therefore, a trader facility is needed to communicate or negotiate with users. With the trader, users will agree upon the functionality of the application and the desired QoS and costs. The trader will ask the performability manager if it is possible to realize the QoS and cost requirements.

New applications are composed of application components from the component library, which contains, apart from a description of the application components, a description of the system parts and network components. The performability manager needs a description of all these components to create a distributed system model suitable for performability evaluation.



■ **Figure 8.** *The environment of the performability manager.*

The preparation of the dynamic reconfiguration by the performability manager is performed by using a model according to the current distributed system configuration. With this model, the performability manager can create and evaluate alternative configurations and choose the optimal configuration.

To detect a violation of the QoS, a performability monitor is needed to check the applications of the distributed system with respect to their QoS. If this performability monitor detects a violation, it must report it to the performability manager.

Operation of the Performability Manager

Once triggered, the performability manager establishes a reconfiguration in three steps following the trigger:

0: Trigger for a reconfiguration.
1: Creation of alternative configurations.
2: Evaluation of the alternative configurations.
3: Performing the reconfiguration.

Triggers for a Reconfiguration

The need for a reconfiguration can arise because of the introduction of a new application or to a perceived degradation of the QoS. The latter can be due to a failure or due to the first.

When a reconfiguration occurs, $\Omega_{current}$ will change. This means that TC, $\mathcal{A}$, S or $\mathcal{M}$ will change. When a new application application is introduced, TC, $\mathcal{A}$, and $\mathcal{M}$ will change. For a new application new tasks will be generated, so that $TC \leftarrow TC \cup TC_{new}$. The set of applications will of course be expanded, $\mathcal{A} \leftarrow \mathcal{A} \cup A_{new}$, and the mapping will change. Consequently, the addition of a new application results in a new mapping, $\mathcal{M} \leftarrow \mathcal{M} \cup M_{new}$.

The second possible trigger is a perceived degradation of the QoS, such as that resulting from a failure. This can be coped with by rerouting of communication or reallocation of applications. If this happens, only the mapping is subject to change. If a system part failure occurs, the set of available system components becomes smaller, $S \leftarrow S - S_{failed}$, and therefore the mapping must be changed. If an application is to be removed, then the set of tasks, applications and the mapping will change.

Creation of Alternative Configurations

The performability manager creates a set of configurations in order to select the optimal configuration. The current configuration, $\Omega_{current}$, and the trigger for reconfiguration serve as inputs for this creation process. As explained in the previous section, first the component sets TC, $\mathcal{A}$, and S of $\Omega_{current}$ are adjusted according to the trigger for reconfiguration. Then the decision upon the mapping for the new configuration Ω_{new} must be made.

The mapping is a complex function. If we allocate m application components on n system parts, then there are n^m possible allocations. This number even increases if we allow for component replication or if we take into account the routing of the communication between components. The optimal mapping is a problem of exponential complexity.

In general, the goal is to find the mapping that minimizes some cost function but fulfills other constraints. The constraints can have various origins, such as resource constraints (memory, cpu, disk, communication), availability and reliability constraints, or dependency constraints.

These constraints have a direct (cpu capacity) or indirect (precedence relations) relation with the SPPs and therefore with the established or required QoS. Using these constraints we can create various cost functions, which will in general lead to different mappings. Our specific goal is to fulfill the required QoS related constraints and optimize the other QoS related constraints.

We define $c_{i,k}$ as the incurred cost for the allocation of application component a_i to system part s_k and c_{ij,kl_v} as the cost for the routing of the communication (a_i,a_j) if path p_{kl_v} is used. The costs must be derived from the QoS-related requirements. For each type of application A^j there are different QoS requirements QoS_{A^j}. From the required QoS for A^j, we derive QoS bounds B_{A^j}, for example on performance and reliability, $B_{A^j}\{B_{perf_{A^j}}, B_{rel_{A^j}}\}$. This results in a general cost function:

$$\sum_{\forall s_k}^{S}\sum_{\forall a_i}^{VA^j} c_{i,k}x_{i,k}^{j} + \sum_{\forall p_{kl_v}}^{P}\sum_{\forall (a_i,a_j)}^{EA^j} c_{ij,kl_v}y_{(a_i,a_j),p_{kl_v}}^{j} < B_{A^j}$$

The costs in such a function can represent the failure rate of application A^j. If we assume that no application (software) failures occur and that the system parts and network links (hardware) are never turned off [35], then the failure rate of A^j is expressed using the failure rate of the system components, as shown in the algorithm below. In the cost function, f_{sk} is the failure rate of system part s_k and f_{pkl_v} the failure rate of path p_{kl_v} both used by application A^j. We then derive a mapping algorithm in which the failure rate of an application A^j is optimized and capacity constraints must be fulfilled:

- Given TC, $\mathcal{A}$ and S and $\text{QoS}_{\mathcal{A}}$
- For all $A^j \in \mathcal{A}$ and QoS_{A^j}
- Realize $M^j = [Y^j, X^j]$
  - Minimize the failure rate $F_{A^j} \in B_{rel_{A^j}}$:

$$\sum_{\forall s_k}^{S}\sum_{\forall a_i}^{VA^j} f_{s_k}x_{i,k}^{j} + \sum_{\forall p_{kl_v}}^{P}\sum_{\forall (a_i,a_j)}^{EA^j} f_{p_{kl_v}}y_{(a_i,a_j),p_{kl_v}}^{j} < F_{A^j}$$

  - Subject to capacity constraints:
    * For all $s_k \in S$:

$$\sum_{\forall A^j}^{A}\sum_{\forall a_i}^{A^j} \rho_{i,s_k}^{j}x_{i,k}^{j} < 1$$

    * For all $p_{kl_v} \in P$

$$\sum_{\forall A^j}^{A}\sum_{\forall a_i}^{A^j}\sum_{\forall a_j | j\neq i}^{A^j} \rho_{i,j,p_{kl_v}}^{j}y_{(a_i,a_j),p_{kl_v}}^{j} < 1$$

With this mapping algorithm, we can create a set of configurations by using different cost functions or different heuristics for solving the optimization problem [38]. Two alternative mappings for the experimental application presented earlier are shown in the sidebar entitled "Describing Alternative Configurations with Graphs."

Evaluation of Alternative Configurations

Performability models are required for the evaluation of alternative configurations with respect to QoS. Therefore, every alternative configuration Ω_{alt} must be transformed into a performability model Ψ_{alt}. This is done by replacing each component of TC, $\mathcal{A}$, and S by a stochastic Petri net (SPN) submodel (see the sidebar entitled "SPN Submodels"). These models are both flexible and relatively easy to solve with current software tools [41]. We use the mapping and the application and system graphs to create the overall performability model. Notice

Describing Alternative Configurations

Changing the mapping of application to system parts results in alternative configurations of the distributed system. The mapping algorithm presented can lead to alternative configurations with 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, *sun2*. 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 (NTC', DBC', and BC'). For this alternative configuration we also have to change (expand) the set of application components and the application graph.

AC = {GC,SC,NTC,DBC,BC,NTC',DBC',BC'}

VA^3 = {GC,SC,NTC,DBC,BC,NTC',DBC',BC'}

EA^3 = {(GC,SC),(SC,NTC),(NTC,DBC),(NTC,BC),(SC,NTC'), (NTC',DBC'), (NTC',BC')}

The replicated components are treated as independent components. The corresponding allocation matrices are:

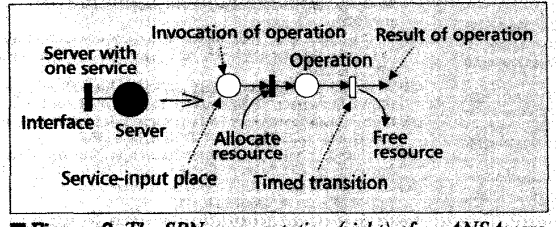
$$Z^2 = \begin{pmatrix}1&0&0&0&0\end{pmatrix},\ X^2 = \begin{pmatrix}1&0\\1&0\\1&0\\1&0\\1&0\end{pmatrix},\ Y^2 = \begin{pmatrix}0\\0\\0\\0\end{pmatrix}$$

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

SPN Submodels

We present a generic way to transform each component of the distributed environment into an SPN submodel. We describe the modeling of the three levels, the application level, system level, and the users, using SPNs. Finally, we give a complete SPN representation of the experimental environment.

For each operation or service provided by an application component, a generic SPN submodel is built. In the SPN submodel 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, such as a CPU, must be allocated and the operation can be performed (the timed transition). The SPN representation of one operation of a computational object or application component is shown in Fig. 9. The output of the timed transition (the operation) is an announcement or an interrogation to another operation. With an interrogation invocation as output the component will await an answer and continue operation.



■ **Figure 9.** *The SPN representation (right) of an ANSAware service provision (left).*

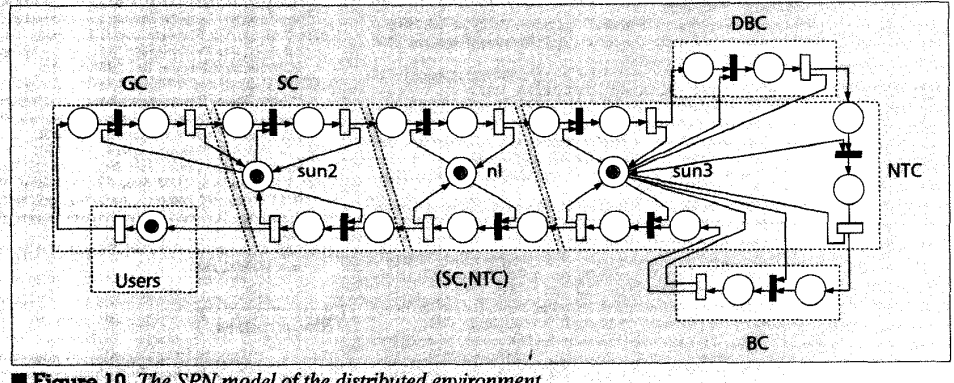
The duration of an operation is represented by a timed transition. These transitions represent the work demanded from the resource, for example the CPU busy time ($1/\mu_{i_u,k}$, the average service time of operation u of application component i on system component k). We can estimate these parameters by running and monitoring the component in isolation (one component on a single workstation), or use a more theoretical way as discussed in the section concerning the annotation with SPPs of the performability manager's view.

The system-level components of the distributed environment are the resources in the SPN model (Fig. 10). In this model, for example, the CPU of a workstation is the resource which must be allocated. For communication, the resources are the network links.

The communication between components can be represented in similar way as the operations. Per (remote) operation, or communication between two application components allocated to different system parts, a path, and subsequently a network link or a set of network links 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.

The generation of requests by USERS, which is a Poisson arrival process, is modeled as a timed transition.

The SPN representation of the distributed environment in Fig. 10 uses configuration M^1. We left out of the SPN representation the failure and repair behaviour of the system level components. The inclusion of these aspects, along the lines presented in [41], will be discussed elsewhere in this article.



■ **Figure 10.** *The SPN model of the distributed environment.*

that we use an SPN model library for all standard system, application, and task components.

Once all the alternative models have been constructed and solved, the optimal alternative configuration is selected, based on the expected QoS derived by the model evaluation.

Note that for this approach to be useful in a real-time environment, the amount of calculations to be performed for the performability evaluations must be small. The involved models must therefore be very simple. More information on model creation and solution can be found in [42].

Once one out of possibly many alternative configurations has been selected, the reconfiguration is performed by the computing platform that supports the distributed system. How the computing platform goes along with this, is not the concern of the performability manager. The only concern of the performability manager is to make sure that it does not request reconfigurations that cannot fulfilled.

Conclusion

We have presented the performability manager, a distributed system components that takes care of guaranteeing user-requested QoS in a failure-prone environment. It does so by carefully selecting an alternative configuration, based on performability evaluation of a number of possible alternative configurations. The models of the alternative configurations are automatically constructed, using a library of SPN model components. Once a reconfiguration has been selected, it is put into effect by a computing platform that supports the distribut-

The performability manager dynamically reconfigures distributed systems whenever needed, to recover from failures and to permit the system to evolve over time.

ed system. We illustrated the approach by describing the role of a performability manager in an ANSAware-based distributed environment.

Model creation is both crucial and difficult. An overall system 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. The problem of the mapping must be solved using graph theoretical, linear programming, or heuristic methods. Because of the expected unstructuredness and complexity of the problems, heuristic solutions seem to be the most promising.

The creation of alternative configurations can theoretically provide us with a huge number of alternative models. A solution to this problem might be to use the performability model of the current configuration for a sensitivity analysis [43, 44, 45]. The results of this analysis can guide the choice of sensible alternative configurations.

References

[1] J. Kramer, ed., Proc. of the Int'l 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.

[2] J. Kramer and J. Magee, "Dynamic Configuration of Distributed Systems," *IEEE Trans. on Software Engineering*, 11(4):424-436, April 1985.

[3] J. Kramer and J. Magee, "The Evolving Philosophers Problem: Dynamic Change Management," *IEEE Trans. on Software Eng.*, 16(11):1293-1306, Nov. 1990.

[4] J. Magee, J. Kramer, and M. Sloman, "Constructing Distributed Systems in Conic," *IEEE Trans. on Software Eng.*, 15(6):663-675, June 1989.

[5] F. Lange, R. Kroeger, and M. Gergeleit, "JEWEL: Design and Implementation of a Distributed Measurement System," *IEEE Tran. on Parallel and Distributed Systems*, 3(6):657-671, Nov. 1992.

[6] L. J. N. Franken and B. R. Haverkort, "Distributed Computer Systems and Logistics Systems: What Do They Have in Common Besides Distribution," Research review, Royal PTT Nederland NV, 3(3):35-71, Aug. 1993.

[7] S. Ceri and G. Pelagatti, Distributed Databases, (McGraw-Hill, 1984).

[8] H. Muhlhauser, Software Engineering for Distributed systems: The Design Project, IEEE, 1988.

[9] L. Mejlbro, "QOSMIC-Deliverable D1.3C: QoS and Performance Relationships," Deliverable QOSMIC R1082, RACE, 1992.

[10] QOSMIC, "General Aspects of Quality of Service and System Performance in IBC," Deliverable RACE D510, RACE, 1991.

[11] ITUnion, "General Characteristics of International Telephone Connections and Circuits," Red Book Fsc. II.1, CCITT, 1985.

[12] ITU, "Telegraph and Mobile Service and Quality of Service," Blue Book Fsc. II.4, CCITT, 1989.

[13] ETSI, "Network Aspects (NA); General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDN," Technical Report ETR 003, ETSI, 1990.

[14] N. M. van Dijk, B. R. Haverkort, and I. G. Niemegeers, Guest editorial: "Performability Modeling of Computer and Communication Systems," *Performance Evaluation*, 14(3-4):61-78, Feb. 1992.

[15] B. R. Haverkort, "Performability Modeling Tools, Evaluation Techniques, and Applications," Ph.D. thesis, Univ. of Twente, 1990.

[16] J. F. Meyer, "Performability Evaluation of Telecommunication Networks," in Network Teletraffic Science for Cost-Effective Systems and ITC-12 Services, M. Bonatti, ed., pp. 1163-1172, IAC, (Elsevier Science Publishers B.V., 1989).

[17] J. F. Meyer, "Performability: a Retrospective and some Pointers to the Future," Performance Evaluation, 14(3-4):139-156, February 1992.

[18] 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 Reference SC21 N7053, 1993-1-28.

[19] APM Ltd., Cambridge, U.K., "ANSA: An Engineer's Introduction to the Architecture," Nov. 1989.

[20] W. Rosenberry, D. Kenney, and G. Fisher, Understanding DCE, (O'Reilly & Associates, Inc, 1992).

[21] R. Cole, "Application Configuration in a Client-Server Distributed System," in Proceedings of the International Workshop on Configurable Distributed Systems, J. Kramer, ed., pp. 309-317, Computing Control Division of the Institution of Electrical Engineers, IFIP, Imperial College of Science, Technology and Medicine, IEE, March 1992.

[22] G. Dean *et al.*, "Cooperation and Configuration within Distributed System Management," in Proceedings of the International Workshop on Configurable Distributed Systems, J. Kramer, ed., pp. 274-285, Computing Control Division of the Institution of Electrical Engineers, IFIP, Imperial College of Science, Technology and Medicine, IEE, March 1992.

[23] F. Cristian, "Automatic Reconfiguration in the Precence of Failures," in Proceedings of the International Workshop on Configurable Distributed Systems, editor, J. Kramer, pages 4-17, Computing Control Division of the Institution of Electrical Engineers, IFIP, Imperial College of Science, Technology and Medicine, IEE, March 1992.

[24] Y. H. Lee and K. G. Shin, "Optimal Reconfiguration Strategy for a Degradable Multimodule Computing System," *J. of the ACM*, 34(2): 326-348, April 1987.

[25] 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, Oct. 1989.

[26] A. Gersht and R. Weihmayer, "Joint Optimization of Data Network Design and Facility Selection," *IEEE JSAC*, 8(9):1667-1681, Dec. 1990.

[27] K. Kant, Introduction to Computer System Performance Evaluation, (McGraw-Hill, 1992).

[28] Y. Berders and P. Dickman, eds., Workshop on Dynamic Object Placement and Load Balancing in Parallel and Distributed Systems, The Sixth European Conference on ObjectOriented Programming, ECOOP '92, 1992.

[29] N. G. Shivaratri, P. Kreuger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, 25(12):33-44, Dec. 1992.

[30] S. M. Shatz and J. Wang, eds., Tutorial: Distributed Software Engineering, IEEE Computer Society Press, 1989.

[31] N. S. Bowen, C. N. Nikolaou, and A. Ghafoor, "On the Assignment Problem of Arbitrary Process Systems to Hetrogeneous Distributed Computer Systems," *IEEE Trans. on Computers*, 41(3):257-273, March 1992.

[32] S. Hariri and C. S. Raghavendra, "Distributed Functions Allocation for Reliability and Delay Optimization," Proceedings of the Fall Joint Computer Conference (IEEE), pp. 344-352, 1986.

[33] B. Carre, Graphs and Networks, (Clarendon Press, Oxford, 1979).

[34] S. M. Shatz, J. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," *IEEE Trans.on Computers*, 41(9):1156-1168, Dec. 1992.

[35] J. Laprie and K. Kanoun, "X-Ware Reliability and Availability Modeling," *IEEE Trans. on Software Engineering*, 18(2):130-147, February 1992.

[36] W. Chu and L. Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems," *IEEE Trans. on Computers*, 36(6):667-679, June 1987.

[37] W. W. Chu, L.J. Holloway, M. Lan, and K. Efe, "Task Allocation in Distributed Processing," *IEEE Computer*, 13(11):57-69, November 1980.

[38] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *IEEE Computer*, 15(6):50-56, June 1982.

[39] J. P. Huang, "Modeling of Software Partition for Distributed Real-Time Applications," *IEEE Trans. on Software Eng.*, 11(10):1113-1126, 1985.

[40] L. J. M. Nieuwenhuis, "Fault Tolerance Through Program Transformation," PhD thesis, PTT Research, 1992.

[41] 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.

[42] L. J. N. Franken, R. H. Pijpers, and B. R. Haverkort, "Modeling Aspects of Model Based Dynamic QoS Management by the Performability Manager," submitted to the 7th Int'l Conference on Modeling Techniques and Tools for Computer Performance Evaluation, May 1994.

[43] J. T. Blake, A. L. Reibman, and K. S. Trivedi, "Sensitivity Analysis of Reliability and Performability Measures for Multiprocessor Systems," *ACM Perf. Eval. Review*, 16(1):177-186, 1988.

[44] P. Heidelberg and A. Goyal, Sensitivity Analysis of Continuous Time Markov Chains Using Uniformization, in Computer Performance and Reliability, G. Iazeolla, P. J. Courtois and O.J. Boxma, eds., pp. 93-104, (Elsevier Science Publishers B.V., 1988).

[45] R. Marie, A. L. Reibman, and K. S. Trivedi, "Transient Analysis of Acyclic Markov Chains," *Perf. Eval.*, 7:175-194, April 1987.

[46] L. J. N. Franken and J. W. Spee, "IN/ANSAware," PTT Research report TI-RA-93-154, PTT Research, the Netherlands, Feb.1993.

[47] Studygroup XI. Q.1200, Draft recommendations, Tech'l report, CCITT, 1991.

[48] MARI Computer Systems Ltd., DEMON W.0 User's Guide and Reference Manual, 1993.

Biographies

LEONARD J. N. FRANKEN [M '90] received a Bachelor's degree at the H.T.S Rijswijk and an M.Sc. from Delft University of Technology, both in electrical engineering. He joined PTT Research Groningen, the Netherlands in 1990, where he started in the field of performance modeling and evaluation of telecommunication services. He is currently active in the field of distributed system design, in particular with distributed-system management and the application of performance, dependability, and performability modeling and evaluation. He is working on his Ph.D. in a joined research with the Tele-Informatics and Open Systems group of the University of Twente. His email adress is l.j.n.franken@research.ptt.nl.

BOUDEWIJN R. HAVERKORT [M '88] received M.Sc. and his Ph.D. degrees in computer science, both from the University of Twente in 1986 and 1991, respectively. Since January 1990 he has been an assistant professor in the Tele-Informatics and Open Systems group of the department of computer science at the University of Twente, Enschede, the Netherlands, where he teaches courses on performance analysis of communication networks and computer systems. From January through April 1993 he was a visiting professor at the Teletraffic Research Centre of the University of Adelaide, Australia. His current research interests include performance, dependability, and performability evaluation of fault-tolerant and distributed computer and communication systems and software tools to support these evaluations. He served as a guest editor for a special issue of *Performance Evaluation* devoted to performability modeling of computer and communication systems in 1992. He is a member of the IEEE Computer Society and the ACM. He received a Koninklijke/Shell research prize early 1991. His e-mail address is b.r.h.m.haverkort@cs.utwente.nl.