16

Determining the Availability of Distributed Applications

Gabi Dreo Rodosek
Thomas Kaiser
Leibniz Supercomputing Center
Barer Str. 21, 80333 Munich, Germany
Email: {dreo, kaiser}@lrz.de

Abstract
Distributed applications can be seen as complex pieces of software which are distributed across various heterogeneous end systems in a network. Mostly, they rely on the provision of other applications as well. Adequate methods for testing the availability of distributed applications do not exist yet. Availability must be determined based on the availability of the involved end systems and network nodes. In view of this, we propose (i) a service graph for the description of functional dependencies, (ii) extend it to a parameterized availability graph to describe the involved components, instantiate the graph, and (iii) give calculation rules for determining the availability of applications. Although most dependencies are described during the design phase, some of them can be recognized only during operation. To deal with this, user trouble reports about unavailable services are used to refine the availability graph and improve the availability calculations.

Keywords: Distributed Applications, Service graph, Availability graph

1 INTRODUCTION

During the past decade worldwide network and system capabilities have rapidly advanced to meet the challenge of the information age imposing high requirements on network and systems management ([HeAb 94], [Slom 94]). The pace has been further fueled by customer demand for a variety of innovative services that require the support of high-quality reliable networks. Availability of provided services, as one of the most important Quality of Service (QoS) parameters, is another customer demand. Until recently, availability is mostly referred to the availability of particular links, network nodes or end systems. However, when talking about the availability of applications, it is necessary to deal with complex dependency relations between several components which are necessary for the provision of an application. Obviously, this increases the complexity substantially. Since users perceive degradations in terms of used services, it becomes crucial to solve this topic. To clarify the terminology, a service provider uses distributed applications (e.g., MHS) to provide services (e.g., email) to users (customers) with certain QoS parameters, as agreed in service-level agreements (SLAs).*

In spite of the relevance of this topic, an adequate solution – being applicable also in a production environment – has not yet been found. The difficulty is to recognize the dependencies, and thus, to identify the involved components for the provision of an application. In general, it is necessary to describe (i) functional dependencies between services, and (ii) environmental dependencies (i.e., how services are realized in a concrete environment). With a service description, it is possible to identify the constituent functional building blocks, and thus recognize functional dependencies. Existing approaches (e.g., [ISO 10746], [ITU-T Q.1201], [TINA-C 95], [ITU-T M.3010], [Gosc 91], [KPM 96], [Dreo 95]) refer to a service description with respect to certain requirements (e.g., for trading, for correlation of trouble tickets). Dependencies between resources are for example modeled using relationships ([Kaet 96]) to determine the root cause of a fault. However, availability aspects have not yet been tackled.

The paper proceeds as follows: Section 2 illustrates the complexity of the problem area, especially functional and environmental dependencies, and clarifies the term availability from the user and the service provider view. Section 3 describes the proposed methodology, including three steps: firstly, we propose a service graph for the description of functional dependencies and extend it to a parameterized availability graph to describe environmental dependencies as well as give calculation rules for determining the availability. Additionally, the instantiation of the availability graph is discussed. Secondly, we answer the question how to test the availability of end systems and network nodes hereby referring to the testing capabilities of management tools. Thirdly, in order to cope with the dependencies which are not known during the design phase, we discuss some enhancements of the proposed approach to calculate the availability as precise as possible. Implementational aspects, namely the realization of the proposed methodology in a production environment, are described in Section 4. Finally, Section 5 gives some concluding remarks and outlines further work.

2 PROBLEM DESCRIPTION

The complexity of determining the availability of distributed applications results from various points such as functional dependencies between applications, their distributed realization on various end systems (e.g., servers) and the used communication infrastructure. To give an idea about the complexity, we describe a relatively simple example of a WWW proxy depicted in Figure 1.

The example starts with a user requesting a WWW page. The WWW client requests first the IP address of the proxy server for which a request is sent to the DNS (domain name service) and served by a corresponding DNS server of the provider (steps 1-4). Afterwards, the WWW client sends its request for a WWW page to the proxy server (step 5). The proxy server itself requires the IP address of the remote WWW server and therefore makes a request to the DNS server (steps 6-9). The WWW proxy either retrieves

*The terms service and distributed application are used interchangeably.

the page from the remote WWW server (steps 10-11) or provides the WWW page from the cache (steps 10'-11'). If all steps are successful, the proxy server sends the reply to the WWW client (step 12). This example simplifies the real situation. For example, we assume that there is only one configured DNS server in the resolver part. Besides, we have not discussed the functionality of the WWW client (e.g., the possibility of internal caching). Additionally, we assume (IP) connectivity between all involved devices, that the WWW proxy server has enough resources (e.g., memory, CPU), and all required processes are running. Finally, we have demonstrated only some functional dependencies between applications. For example, that a WWW application requires only the provision of a DNS. We have not assumed that the WWW application can be distributed over a distributed file system. As illustrated, the concrete environmental dependencies are very complex and cannot all be foreseen.

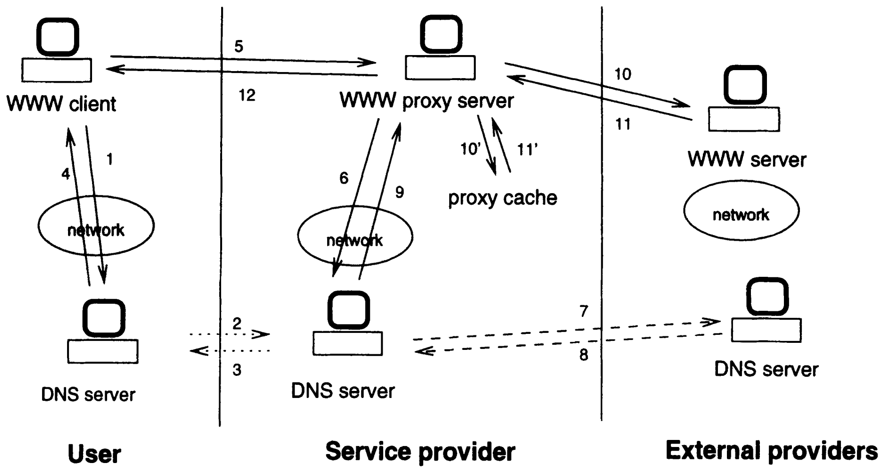

Figure 1 Example: WWW proxy

Since the provision of services in today's distributed environment is a complex task, many applications rely on a hierarchy of underlying services. A distributed application (e.g., remote printing) depends on client, server and gateway processes, which themselves depend on system software and hardware. Besides, networking devices and communication links must be properly configured and in operating state. For example, the usability of a WWW application depends on whether the DNS application is available, whether the DNS server can resolve the stated name, whether the WWW server workstation is in operation and the network connectivity provided. In general, the availability of a distributed application depends on the availability of (i) other applications, such as DNS, the availability of (ii) end systems (e.g., workstations) where the software is running, and availability of the (iii) connecting communication infrastructure, the network.

As depicted in Figure 1, it is necessary to distinguish between different views of availability. A service is available for a user if he can use it at a certain time with the agreed QoS parameters at a predefined access point. It is the obligation of the service provider to assure the correct provision of services at this access point. We define the interface between a user and a service provider as a service access point (SAP) where the usage and provision of services take place. If the provision of a service relies on the provision of services from an external service provider, we define this interface between providers as SAPs as well. The QoS parameters at the SAP are handled within SLAs, too.

To illustrate the various availability views, let us consider the Leibniz Supercomputing Center (LRZ) as a service provider of network connectivity and computing power for the Munich Universities. The Department of Computer Science of the University uses the provided IP connectivity from the LRZ to realize a distributed file system. In our example, the connecting router interfaces are the SAPs. If one of these interfaces is down, the availability of services such as NFS mounts within the department is 0% for the time of the outage of the interface. However, from the service provider's viewpoint, only one of 250 interfaces is down. On the other side, the LRZ itself uses services from other service providers. In such a case, the LRZ acts as a user.

Besides, the term availability of distributed applications itself is imprecise. Is an application available, for example, if the end systems and the network nodes involved in its provision are available or if only n-1 of n building blocks of an application are available or if there is no user trouble report?

3 PROPOSED APPROACH

Beside the specification of an appropriate testing interval, the question is how to test whether a complex distributed application, like DNS, is available at the SAP. A straightforward method could be to test the usage of a service also by a service provider with the maximal possible request rate for a service. This is of course unrealistic. Another unrealistic approach is to document user trouble reports about unavailabilities of services, and perform availability calculations solely on these reports.

Availability is closely related with faults. Because a distributed application is realized with components (network nodes, end systems), a straightforward definition is that an application is available if there are no faults in a distributed system. Faults in our context are breakdowns of devices or wrong configurations of software which could have an influence on the availability. From the user's view, performance degradations may also be seen as faults, which we assume are handled with SLAs.

The proposed methodology consists of the following three steps.

3.1 Identifying the involved components for the provision of distributed applications

In order to recognize the involved components for the provision of an application, we need to describe (i) generic functional dependencies between applications, and (ii) specific environmental dependencies (i.e., the realization of a distributed application in a concrete environment).

We propose to describe the functional dependencies between services in terms of a layered *service graph* where the nodes represent the services and the edges represent the relationship *is_functional_dependent*, as depicted in Figure 2. For example, the provision of the WWW service depends on the provision of the IP connectivity service. In terms of a service hierarchy, we may distinguish between application services (e.g., email, backup) as well as basic and communication services, which are used for the provision of application services. In order to determine the availability of distributed applications, it is

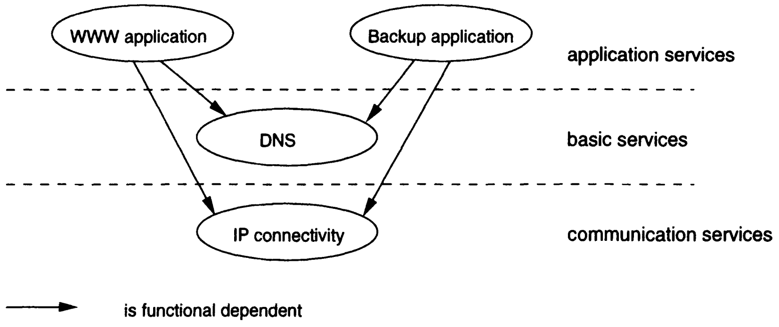

Figure 2 Layered service graphs

necessary to (i) recognize the involved components, and (ii) to determine the availability of these components. For this purpose, we use the service graph as a basis to generate a *parameterized availability graph* (Figure 3). Some ideas for the design of the availability graph have been adopted from [JeVa 87].

The availability graph serves two purposes: firstly, it describes dependencies in the system, and secondly, it gives calculation rules for determining the availability of applications.

The nodes of the availability graph represent the parameterized service descriptions, such as IP connectivity(client, server), whereas the edges are used for calculation purposes. As depicted in Figure 3, we distinguish between AND and OR edges.

The skeleton of the availability graph (refer to Figure 3) represents the functional dependencies between services (e.g., the WWW service requests the provision of the DNS). For example, the WWW service is available if the DNS service AND the IP connectivity service are available. Referring to Figure 3, the service WWW(client, server, file) is available if WWW_client(host) AND DNS(client, server, request) AND WWW_server(host) are available. In other words, the application is not available if there is a fault(WWW_client(host)) OR a fault(DNS(client, server, request)) OR a fault(WWW_server(host)).

From this skeleton, we further refine the nodes. The refinement of the DNS node is shown in Figure 3. The DNS service is available if either one of the two name servers is available (an example of the OR operation).

Parameters are used to represent functional and environmental (i.e., implementational) aspects. Functional aspects, like WWW(client, server, file), refer to the functionality of a service (e.g., the functionality of the WWW service which is to transfer files from server to client). Environmental aspects are used to describe, for example, that a WWW server may be realized over a distributed file system (AFS) or local disk. In such a case, we extend the node WWW server with the graph on the lowest layer. For example, the WWW server runs on a host with minimum requirements on cpu power and disk space, which are additional parameters of the node host(process, cpu, disk). This means that the WWW server on the host(process, cpu, disk) is available if the *process* is running, the currently available cpu is over the limit *cpu*, and there is at least *disk* space available.

To summarize, during the design phase the availability graph is generated from the service graph and extended/refined to deal with known environmental dependencies. In case new dependencies or "hidden" side-effects are encountered during operation, the availability graph needs to be extended. An example of such a "hidden" side-effect is the unavailability of a service due to performance problems of another workstation caused by NFS timeouts. We have the possibility to extend the graph by including either new AND and/or OR edges and/or new parameters.

The next step is the instantiation of the parameterized availability graph, which means to replace parameters with concrete values. Afterwards, the instantiated availability graph is the basis for the calculation of the availability of a service, because the leaves of the graph specify the involved components for the provision of a service and the dependencies in terms of AND and OR. By testing the correct operation of the involved components, it is possible to calculate the availability of applications.

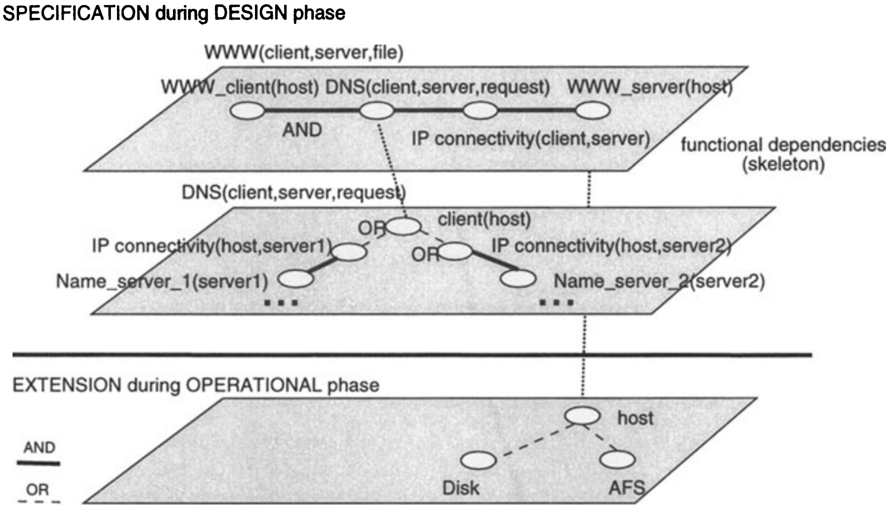
SPECIFICATION during DESIGN phase
WWW(client,server,file)
WWW_client(host)
DNS(client,server,request)
WWW_server(host)
AND
IP connectivity(client,server)
functional dependencies (skeleton)
DNS(client,server,request)
client(host)
OR
OR
IP connectivity(host,server1)
IP connectivity(host,server2)
Name_server_1(server1)
Name_server_2(server2)
EXTENSION during OPERATIONAL phase
host
Disk
AFS
AND
OR

Figure 3 Parameterized availability graph

A topic deserving more attention is the instantiation of the parameterized availability graph. For example, host(www process, cpu, disk) is instantiated to *sunmanager(httpd, 10MIPS, 10MB)*. The problem herewith is that a complete automatic instantiation is not possible due to the complexity of the environment. A semi-automatic instantiation can be achieved if the (i) distributed system is described with relevant attributes (i.e., workstation with a specific amount of memory, CPUs), and (ii) if there exist certain rules (e.g., that a DNS server workstation needs to have certain attributes.) These rules may represent a part of the organizational policies. Combined with inventory tools to recognize the devices and their attributes in a system, such instantiation can be performed automatically to a large extent. Despite this, certain steps during the instantiation need to be performed manually.

An open question is the granularity of the availability graph. The refinement of the graph should stop if the leaves of the instantiated availability graph can be either tested with existing testing methods or they point to other services, as shown in Figure 3 and Figure 4.

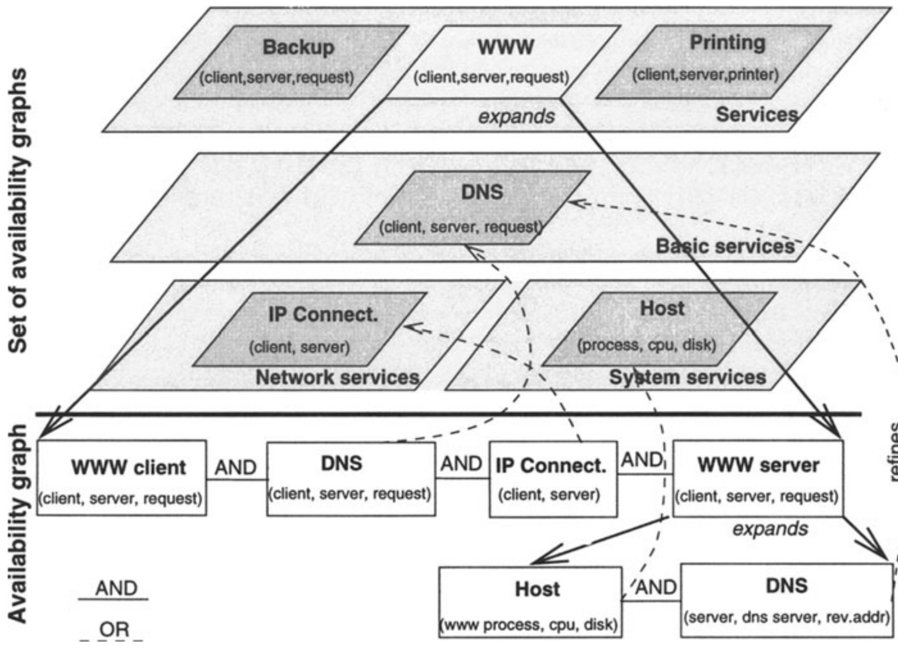

Figure 4 Example: Intranet WWW service

3.2 Testing and calculating the availability of involved components

After having identified the involved components, it is necessary to analyze how to test the availability of these components. Rather then describing various testing methods, we analyze this problem by calculating the availability of the IP connectivity service in our environment. In this case, the components we have to deal with are IP routers with IP interfaces. Because there exists a relatively simple built-in IP test (e.g., "icmp echo request"), this is considered to be a simple service.

To determine the availability of components we need to repeat tests for each device every Δt, which needs to be short enough to recognize relevant faults, i.e. there is no degradation of the service perceived. Thus, the availability A of an IP interface I within a time period $[t_1, t_2[$ is defined as

$$A(I)_{t_1,t_2} := 1 - \frac{\sum_i \{\Delta t \mid t_1 \leq \tau_i < t_2 \ \wedge \ \text{fault}(I, \tau_i)\}}{t_2 - t_1}$$

where $\tau_i = t_1 + i * \Delta t$ for every $i \geq 0$ and fault(I, t) means fault of I at time t.

For the provision of the IP connectivity service, a backbone of IP routers is required. The SAPs are the interfaces of the routers. The task is to determine the availability of the IP connectivity service between these SAPs. However, to test the connectivity of n interfaces in a shared medium backbone, we need to perform $n \times (n-1)$ tests. Due to the enormous effort it is necessary to reduce the number of tests. Therefore, the testing is performed centrally from one interface (i.e., the interface I_P where the provider has its workstations) to any other interface. In this case, we do not measure the availability $A(I_1, I_2)$ of IP connectivity between the two interfaces I_1 and I_2, but the availability $A(I_P, I_1)$ and $A(I_P, I_2)$ combined to $A(I_P, I_1, I_2)$. We can assume that $A(I_P, I_1, I_2) \leq A(I_1, I_2)$†. We also assume that all interfaces have the same priority for the provider.

For the IP connectivity service, the following availability calculations are of interest:

- Availability $A(I)$ for one user access point (i.e., interface I) which can be calculated as stated previously.
- Assume that a user (e.g., the Department of Computer Science) runs, for example, a distributed file system over the Munich Network. Thus several SAPs (i.e., $I_1, I_2, \ldots$) are of interest. The availability $A(I_1, I_2, \ldots)$ of all relevant IP interfaces for a user is reduced by the sum of time intervals in which at least one interface is down.
- Availability $A(I_{all})$ for all IP interfaces of the service provider is the average of the availability of all interfaces. This result is relevant for the service provider to gain information about the reliability of the used devices.

Determining the availability of the IP connectivity service is relatively simple because the service itself is simple. The availability of the IP service depends only on the availability of the IP network devices. Besides, there exists a simple testing mechanism.

If we want to determine the availability of end systems like hosts for our applications, there exists no easy test mechanism. We do not test only the connectivity, but need to test at least whether all required system processes are running and whether there are enough free resources. Existing management tools enable us to perform some of these tests. However, without describing some dependencies, we cannot conclude that an application running on some end systems is working correctly. In order to calculate the availability of distributed applications calculation rules and adequate testing methods are necessary.

To improve and to refine the availability graph as well as to recognize what are the most relevant parameters in a production environment, we introduce another step proposing a learning approach.

3.3 Improving the methodology by refining the availability graph

It is impossible to specify all dependencies in a large production environment during the design phase. Some of them cannot be foreseen and are only recognized during operation. Therefore, it is necessary to extend and refine the availability graph.

For this, we propose a "learning" approach by analyzing user trouble reports about unavailable services. When a user reports an unavailable service, and a fault (the cause of the reported unavailable service) has not been diagnosed by a provider, a discrepancy has occurred. In general, there may be three reasons for this:

- either the testing methods are imprecise for this specific case, or
- the existing availability graph for this specific problem is not complete, or
- the problem description of the user was imprecise or has pointed to another fault.

†In practice a partial outage only between I_1 and I_2 is rare in a shared medium backbone.

Since user trouble reports are in general imprecise, we developed the so-called "Intelligent Assistant", a specialized tool which supports the user during the report and the localization of a fault in a predefined way. When the cause of a fault is identified, we can either (i) improve our testing method, or (ii) extend the availability graph, or (iii) change the decision trees in the "Intelligent Assistant". Let us illustrate this on an example. A user reports a trouble ticket with "WWW service unavailable", but a fault could not be recognized by the provider by testing the components as identified by the instantiated availability graph. After diagnosing the fault, we may recognize that the reason for the degradation of the service was, for example, an overload due to NFS problems. Thus, we need to extend our availability graph to describe a dependency with NFS and to install or develop an appropriate testing method.

3.4 Assessment of the solution

To summarize, we have proposed a methodology which consists of the following steps:

- build generic service graphs for each service, extend the service graphs to parameterized availability graphs to describe the involved components, instantiate the availability graphs in a concrete environment,
- perform testing of involved components and calculations based on the given calculation rules,
- refine the availability graphs with additional parameters and nodes to improve the availability calculations.

A relevant point is that a core set of service graphs and parameterized availability graphs can be specified. Such a core set can be applicable also for other service providers. Only the instantiation of the availability graphs is environment dependent.

The proposed solution – giving us the possibility to determine the actual and statistical availability of applications – is evaluated with respect to the following criteria: (i) scalability, (ii) manual effort, and (iii) dynamic changes.

There are two types of scalability we have to consider. Firstly, with respect to the number of services and nodes in an availability graph, and secondly, to the number of instantiations. Adding a new service requires a specification of a new service graph which can be considered to scale well. The same appears also for nodes in the availability graphs. The increasing number of instantiations of the derived availability graph is approached by adding parameters.

With respect to the manual effort, the main part consists of the definition of the service and the availability graphs. As already said, it is possible to specify a core set of generic service graphs and derived availability graphs. The effort to incorporate specifics is considered to be acceptable with appropriate tool support.

Dynamic changes refer to the instantiation of the availability graph as a consequence of changes in the system or network configuration. An approach to deal with new configurations is a version controlled database. Because network and system configurations should be stored in such a version controlled database, the additional storage of the availability graphs can be considered to be acceptable.

4 IMPLEMENTATIONAL ASPECTS

Due to being in a production environment, we started from a bottom-up approach to configure and implement a tool set for determining the availability of our IP router backbone (3.2). To supervise our network nodes, we take two different approaches: firstly, we use a developed tool to test IP connectivity which allows us to ping 250 nodes in parallel in a few seconds. Secondly, special configurations of HP OpenView Node Manager are used to poll all important network nodes every 5 minutes. We use both approaches to tune our tests. In recent years we adopted and refined tests of the former OpC, now IT/Operations, to supervise our server systems, and automatically generate trouble tickets. We used these tools as a basis for our own developments. As depicted in Figure 5, we

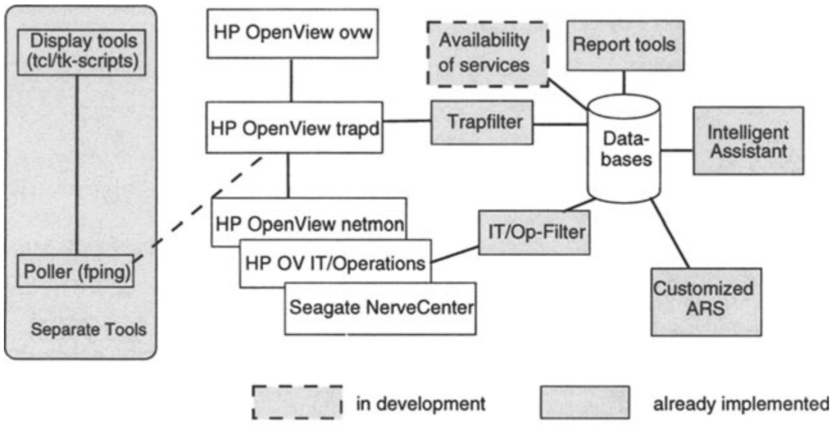

Figure 5 Tool support in our production environment

developed several tools to implement the proposed methodology. For the evaluation of the events and to store the detected faults in a database, we have implemented an application which hooks in HP OpenView event processing and stores them after some preprocessing in an ARS schema. With this tool, we have implemented the availability calculations of Step 2. Report tools have been developed to display the calculated data. Besides, tools for the specification, modification and usage of the availability graphs are under development (Step 1). These tools aquire availability data out of several databases.

To test end systems more precisely, and to obtain more precise information about the status of the network, we specified rules to correlate network and end systems alarms. Due to the fact that the correlated alarms generate trouble tickets in ARS, and user trouble reports are documented in the ARS as well, we apply the proposed approach in [Dreo 95] for the correlation of device-oriented and service-oriented symptoms. This is the realization of Step 3. Fault diagnosis is performed according [DrVa 95].

As seen from Figure 5, almost all necessary building blocks (i.e., tools) to develop the tool to monitor the availability of services have already been implemented and are in production use. So far, we are testing the semi-automatic realization of the instantiation of the availability graph. We extend the attributes of the documented components in our inventory database as well.

Being in an open university environment means that our customers do not have service subscription profiles. Therefore, we use the "Intelligent Assistant" to enable a user to specify his used services, and start the availability calculation for these services.

5 CONCLUSION AND FURTHER WORK

The rapid growth of complex, heterogeneous and distributed systems as well as distributed applications impose new complexity to IT management. In particular, management of distributed applications is a challenging topic. Distributed applications can be seen as complex pieces of software which are distributed across various, heterogeneous end systems, the network, and rely on the provision of other applications.

This paper proposes a methodology for determining the availability of distributed applications. We approach this by illustrating the complexity of the dependencies between involved components on an example and discuss the various views of availability. If we want to determine the availability of a complex distributed application (e.g., DNS), we have to deal with (i) dependencies between applications, (ii) non-existent testing mechanisms (e.g., providing information whether the DNS service is operating correctly), and (iii) not knowing which devices are involved in the provision of which applications. Solutions for this are presented in the first step. In the second step of the methodology, we clarify the term availability, and present the availability calculations for the simple IP connectivity service. This is possible because (i) simple testing methods do exist (e.g., icmp echo request), (ii) there are no dependencies with other applications, and (iii) it is clear what devices to test. Besides, in practice, we have to deal with dependencies which are not known at design phase. Therefore, we need further to improve and extend the availability graph during the operational phase by analyzing user trouble reports (step 3). In view of this, we propose a service graph to describe the functional dependencies which are known at design phase, and extend it to a parameterized availability graph to identify the involved devices. The availability of an application is calculated based on the availability of devices and the rules as given by the availability graph.

With the availability graph, we have a structure to describe complex dependencies in a distributed heterogeneous environment. Because we are a large service provider, experiencing the mentioned problems, we are focusing on completing the availability graph as precisely as possible according to our expertise. Such an availability graph can be applicable also for other service provider environments, as well as other applications, such as event correlation. Another aspect worth noting is that the availability graph – and the described dependencies – gives the basis to analyze bottlenecks in the system either on the service or device level. Our further work will therefore concentrate on the optimization of the availability graph, and the development of methods to recognize bottlenecks (i.e., weak points) in the system.

Acknowledgements

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team is a group of researchers of the Munich Universities and the Leibniz

Supercomputing Center of the Bavarian Academy of Sciences. It is directed by Prof. Dr. Heinz-Gerd Hegering.

6 REFERENCES

[Dreo 95] G. Dreo, *A Framework for Supporting Fault Diagnosis in Integrated Network and Systems Management: Methodologies for the Correlation of Trouble Tickets and Access to Problem-Solving Expertise*, PhD thesis, University of Munich, 1995, Verlag Shaker.

[DrVa 95] G. Dreo and R. Valta, "Using Master Tickets as a Storage of Problem-Solving Expertise", In [INM-IV 95], pages 328–340.

[Gosc 91] A. Goscinski, *Distributed Operating Systems – The Logical Design*, Addison-Wesley, 1991.

[HeAb 94] H.-G. Hegering and S. Abeck, *Integrated Network and System Management*, Addison-Wesley, September 1994, Reprint 1995.

[ICDP 96] A. Schill, C. Mittasch, O. Spaniol and C. Popien, editors, *Proceedings of the IFIP/IEEE International Conference on Distributed Platforms: Client/Server and Beyond: DCE, CORBA, ODP and Advanced Distributed Applications*, IFIP, Chapman & Hall, 1996.

[INM-IV 95] Y. Raynaud, A. Sethi and F. Faure-Vincent, editors, *Proceedings of the 4th IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara*, IFIP, Chapman & Hall, May 1995.

[ISO 10746] "Open Distributed Processing – Reference Model", IS 10746, ISO/IEC, 1995.

[ITU-T M.3010] "Principles for a Telecommunications Management Network", Recommendation M.3010, ITU-T, October 1992.

[ITU-T Q.1201] ITU-T, *Principles of Intelligent Network Architecture*, Q.1201, 1993.

[JeVa 87] E. Jessen and R. Valk, *Computer systems: Basics of Modelling (in german)*, Springer Verlag, 1987.

[Kaet 96] S. Kätker, "A Modeling Framework for Integrated Distributed Systems Fault Management", In [ICDP 96], pages 186–198.

[KPM 96] A. Kuepper, C. Popien and B. Meyer, "Service Management using up-to-date quality properties", In [ICDP 96], pages 448–459.

[Slom 94] M. Sloman, *Distributed Systems Management*, Addison-Wesley, June 1994.

[TINA-C 95] TINA-C, "TINA-C Deliverable: Service Architecture Version 2.0", Technical Report, TINA Consortium, 1995.

7 BIOGRAPHY

Gabi Dreo Rodosek received B.S. and M.S. degrees in computer science from the University of Maribor, Slovenia, and the degree of a Dr.rer.nat. in 1995 from the University of Munich, Germany. **Thomas Kaiser** received his diploma (Diplom-Informatiker, M.Sc.) in computer science in 1989 from the Technical University of Munich, Germany.

Both are research staff members at the Leibniz Supercomputing Center and members of the Munich Network Management Team, being engaged in several network and systems management projects.