Accessing and Customizing Services in Distributed Systems

Ralf Guido Herrtwich[1] and

Uwe Wolfgang Brandenburg[2]

TR-89-059

October 26, 1989

Abstract

In a distributed system, entities access services provided to them by other entities at remote sites. While it may be unimportant to the service users which entities act as service providers, they often have other requirements on the services they use. On the other hand, service providers only have certain possibilities. Both the requirements and possibilities can be described by means of quality-of-service parameters (QOSPs), which have to be determined for each service session. In this paper we design a session establishment service (SES) which takes QOSP values into account. The SES can be used for any kind of QOSPs since it uses badness specifications as a uniform means to identify the usefulness of a certain QOSP value to a service user, to determine the relative importance of single QOSPs, and to calculate the overall quality of a service. Three kinds of QOSPs are distinguished: Static parameters do not change as long as the service is available, dynamic parameters depend on the current state of a service provider, and retrospective parameters result from evaluations of the service which are obtained from previous service users. While some QOSP values are available *per se* others can only be accomplished if the service provider schedules its resources appropriately. The reservation of resources can be integrated within the SES. This is especially important for real-time services.

1. International Computer Science Institute, Berkeley, CA.

2. National Research Corporation for Mathematics and Data Processing (GMD), Research Center for Open Communication Systems (FOKUS), Hardenbergplatz 2, D-1000 Berlin 12.

1. Introduction

Any well-designed software system has a hierarchical structure in which high-level system entities make use of the services provided to them by lower-level entities. It is a general problem to reconcile the needs of the service users with the possibilities of the service providers. This problem becomes especially important in distributed systems, where modules are not statically linked together, but achieve cooperation in a more dynamic way, giving each entity some autonomy about the degree to which it relies on outside functions and to which it offers its own functions to other entities. Here, mechanisms to identify and to access services at run-time are needed. These mechanisms have to be **flexible** (since they should be appropriate for all kinds of services), **dynamic** (since the availability of services will vary), **efficient** (since they are needed before the actual service can be used), and yet **transparent** (since the users should only be concerned with the service itself).

RPCs are a method for service access which has some of these properties [1]. Yet, the only criterion for selecting a server in the RPC model is the name of the procedure. If more than one server is able to execute the procedure it can be chosen at random. For many applications, this random choice will not be appropriate because they have certain requirements on the kind of service provider they use. These applications need some higher degree of control about the service provider selection. In this paper we present a scheme which enables service users to specify their requirements in a way which is independent from the service to be used. This specification is taken into account when the service provider is selected.

While our description here is focussed on the method and not on its implementation, the reader should note that the mechanisms presented have been designed for two systems which are intended to support distributed applications:

- The DASH Project at the International Computer Science Institute and the University of California at Berkeley develops a system to support distributed continuous-media applications including digital video and audio [2].
- The Research Center for Open Communication Systems at the National Research Corporation for Mathematics and Data Processing (GMD) in Berlin investigates mechanisms to support the cooperation of autonomous systems. Part of this work is the BERCIM Project which is concerned with supporting the control of technical processes within a computer-integrated manufacturing framework [3].

In both areas each single distributed application will have highly different requirements, many of which result from the real-time dependencies in these applications. Therefore, it is essential in both systems that an application is able to adjust the functions provided by its supporting system according to its specific needs.

2. Service Sessions

System entities have to know the interface of a service to be able to use the service. This interface consists of the unique name of the service, the names of its different service elements and a specification of their options and parameters. The same service may be offered by different service providers. Sometimes service providers offer services which are slightly different from each other. E.g., one service provider may offer a matrix print service while the other offers a laser print service. Both services have the same service elements and can be operated in the same way. What differs is the quality of the printout being produced.

If two services are almost identical, it is more efficient to manage their differences than to manage two entirely different services [4]. These differences are described for each service by a set of **quality-of-service parameters** (QOSPs) [5]. Before a service can actually be used, the requirements of the service user and the possibilities of the service provider are negotiated in terms of QOSPs. Once both parties have agreed on a common set of QOSP values, they have established a **service session** in which they use service elements the semantics of which are determined by the chosen QOSP values. We can use a familiar example to illustrate such a session establishment:

> Imagine, you want to make a hotel reservation for your holiday in a far-away country by phone. Direct dialing is impossible, so the operator will have to assist you. Unfortunately, you only remember the first letter of the name of the hotel your friends recommended. Hence, you call the operator and tell him that you would like to call a certain town in a certain country. You would like to speak to the assistant manager in a hotel the name of which starts with "H". The operator first determines the phone number of the hotel by scanning through the Yellow Pages of your planned destination. Unfortunately, he finds two hotel names starting with "H". You remember that your friends mentioned that this was the only hotel with a swimming-pool. Since the operator is very friendly, he offers you to find out the right hotel. He also asks about your requirements for the connection, e.g., if you would like to have an immediate connection at higher costs during business hours or if he should schedule the connection for the early evening hours. If you want an immediate connection the operator puts you on hold and calls the hotel. If he is able to get a connection he first checks if it is the right one and then asks to be connected to the assistant manager. If the assistant manager answers the phone the operator will put your call through and the connection is established. You may now use it for any service available from the assistant manager.

Session establishment between processes in a distributed system involves the same steps and can be accomplished in a similar way. A **session establishment service** (SES) takes the place of the phone operator. Just as the number of the operator has to be known to all telephone customers, all system entities have to know the address of the SES. This address, however, is the only service address they need to know. All other addresses can be obtained from the SES.

The SES accepts a session establishment request by a user. In this request the user specifies the service it wants to access and its desired QOSP values. The SES will select a service provider according to this specification. To determine potential service providers the SES may make use of a **Yellow-Pages service** (YPS) as described in [6] and [7]. Every service provider registers its services with the

YPS. Part of this registration is the definition of QOSP values available from this particular service provider. The service provider will only be able to specify values for those QOSPs which are static and do not change throughout the time the service is available. A typical example of a static QOSP is the printer used by a certain print server. Values of QOSPs which are **dynamic** depend on the current state of the service provider and may change. They cannot be determined when the service is registered but only when a session is established. An example of a dynamic parameter is the time it takes the service provider to process a certain service request.

The selection of potential service providers obtained from the YPS is based on the service name and the static QOSPs. The main task of the SES then is to find one service provider within this group which can fulfill the requested values for dynamic QOSPs. The SES has to contact the service providers on behalf of the service users since it cannot determine the QOSP values available itself. Once an appropriate service provider is found, the session can be established. For this establishment, the service user may also submit values of parameters and options which serve to adjust all service elements of the service. E.g., it may specify a *verbose* option for the service which causes detailed information about each execution of a service element to be generated later. These values, however, are merely forwarded to the service provider and do not influence the result of the session establishment. We will neglect them in the following.

3. User Requirements

Every session establishment starts with a request by the service user. In this establishment request, the user names the service it wants to use and specifies the set of QOSP values it desires to obtain from the service provider. If the user does not specify a value for a certain QOSP it is either up to the SES or the service provider to choose an appropriate value. By this, transparency can be achieved. In many cases, the user will be satisfied with more than one QOSP value, although the degree of its satisfaction may vary. Many users will rather use a service of poorer quality than to obtain no access to a service at all. Hence, for each QOSP the user can specify

- **a set of acceptable values** and
- **a metric** according to which values shall be chosen if more than one value is available.

The metric determines the usefulness of a QOSP value to the service user. It can be implemented in different ways. One good way is to assign an integer number to each value determining the **badness** of this value (similar to [8]). Consider the following example: In a network service four different lines are available which have transmission rates of 2400, 4800, 9600 and 19200 baud, respectively. The user may want to keep the fast line available in case some other urgent communication needs occur. The 4800 baud line will suit the application best, but if it is not available the selected line should rather be fast. These requirements can be reflected in the following selection of badness values:

2400 baud	*badness* = 1000
4800 baud	*badness* = 0
9600 baud	*badness* = 200
19200 baud	*badness* = 4000
other values	*badness* = 10000

The example shows that a metric does not need to be linear, although in most cases it will rather be defined by means of a function than by enumerating single values. E.g., to get a value x as close to the optimum value x_{opt} as possible one can specify

$$badness(x) = 100\,(|x - x_{opt}|)$$

Since any function can be used to specify badness values, even complex requirements as in [9] can be expressed easily.

The values of different QOSPs may depend on each other. E.g., the user of a network service may either be satisfied with a long message delay and a large packet size or a small delay and a small size, but it would not like any other combination. Enumerating these dependencies separately for each set of QOSPs would be very troublesome and inefficient. Since the badness is a metric which can be used for any kind of QOSP, it can be used to determine the overall quality of a certain service. A service can be considered unacceptable if the sum of all single badness values exceeds a certain limit. This **overall badness limit** may be specified when the user asks for the service session to be established. By choosing appropriate badness values, the different QOSPs can be assigned different weights in calculating the overall badness. If a QOSP value cannot be chosen under any circumstances, it is assigned a badness value exceeding the overall badness limit. In the example above, the overall limit could be set to 5000 to avoid the selection of any other line.

In its request the service user may already specify the service provider it wants to access. It may also specify a set of potential providers, e.g., by stating a **domain** [10] in which the service provider shall be located. If a system is organized in different domains service users will in many cases want to access a service provider in their own domain or in the domain closest to them. E.g., a print server should be located closely to avoid long ways to fetch the printout. For the selection of service providers, the user may again specify a metric according to which they shall be chosen. Hence, the name of the service provider is a QOSP itself.

Most QOSPs can be hidden from the service user when establishing a service session. In many cases, the user is not interested in the actual values of QOSPs, but rather in the functions an entire set of values stands for. E.g., a user may need a network service to transfer video data. In this case, the attribute *video* should be the only value by which the service user states its requirements. The SES internally transforms this value into the appropriate set of QOSP values.

4. Service Provider Selection

The SES is represented on each host by an **SES agent** which handles session establishment requests for all service users on this node. To each SES agent a description of all possible services is available which contains a specification of the QOSPs of these services. These descriptions enable the SES agent to check and complete the request obtained from the service user.

The SES agent may confine the QOSP selection by the user according to specifications by the local system administrator. The possible reasons for this confinement are manifold: The user may not be fully aware of the facilities provided by the underlying system, e.g., the user may specify to use a 19200 baud line although there is no such line available. It is also possible that a certain QOSP selection is unfair to other users of the host, e.g., by monopolizing the network connection, or that a user is not entitled to make a certain selection. The SES agent can never select QOSP values outside the selection and overall badness specified by the service user. It rather has to reject the request for session establishment in this case.

After having confined and completed the service request according to its local regulations, the SES agent forwards the request including the values of all static QOSPs, but excluding the metrics, to the YPS. The YPS will then return a list of all service providers which are able to accomplish the desired service and the requested QOSP values. It will also determine if a service provider specified by the service user actually fulfills the desired QOSP values. The SES agent will then contact the potential service providers to determine the available values of dynamic QOSPs. Having obtained this information it will apply the metrics given by the service user and select the best service provider available. It will inform all service providers which were not chosen that they can free any resources which they might have reserved already for the new session. It then submits the address of the service provider to the user, and the session is established.

If users only want to inquire about the availability of a certain service before actually establishing a service session, the same mechanisms can be used. However, we see no reason to provide such a service in addition to the actual session establishment because the values of dynamic parameters might change if no immediate reservation is made. On the other hand, a user may immediately close a session if its only interest was in the availability of the session. Leaving the selection of the service provider up to the user not only diminishes transparency, but may also render resources of service providers unavailable for some time until the service user has made its decision. In general, we believe that control functions which can affect other system entities should not be assigned to user processes. If it is really necessary to do so, at least, a timeout mechanism should be used to control the reply of the user.

If the session cannot be established it should be possible to obtain information from the SES about the best QOSP values available. If service requirements resulted from the needs of certain processes, this might not be very helpful, but if they were derived from specifications of a human user, this user might reconsider his requirement specification. Unlike processes, people tend to **bargain**. Instead of getting into the hassle of specifying the complete list of badness parameters, they rather present their maximum requirements as the only possible values. Only if these requirements cannot be fulfilled they might think about alternatives. Of course, processes could be programmed to bargain, too. In this case, however, the information about their "retreat policy" would be available beforehand. Hence, it could already be reflected in their initial session establishment request, avoiding the inefficiencies of multiple interactions.

Obviously, the selection of the best available service provider described above can be very time-consuming and inefficient. One the other hand, many service users will not need to be provided with the best service possible. To select service providers more efficiently, the SES agent can apply certain **heuristics**. It may first query the YPS only for service providers which allow those QOSP values which have the lowest badness. Again, the different weight of QOSP values provides a clue about their relative importance and can be used for these heuristics. Having obtained a list of service providers, the SES agent can contact one of them to see if it also accomplishes the requested dynamic QOSP values. Once an appropriate service provider is found, a session is established with this provider.

This method may not provide the best choice but a reasonably good one. It is efficient and takes the preferences of the service user into account. For efficiency reasons, we have assumed that values of static QOSPs prevail against dynamic QOSPs, i.e. the best selection based on dynamic QOSPs is likely to be a subset of the best selection based on static QOSPs. In case this assumption is not valid or some service users really want to establish the best possible service session, a user can specify a number of service providers which shall be contacted before one service provider is selected. E.g., the users may determine that the best out of the first ten service providers found shall be selected. Obviously, the more service providers are contacted, the higher are the costs of session establishment. A service user will only pay these costs if they are compensated by the service quality achieved.

There are even more ways to enhance the heuristics on which the selection of a service provider is based. It is reasonable to assume that the values of dynamic QOSPs of some service providers do not vary dramatically within a certain time interval. For these service providers, the recent dynamic QOSP values can be stored by the SES in a database and serve as another criterion for selecting a service provider to be contacted. A certain life-span can be assigned to dynamic QOSPs during which the SES has not to renegotiate their values.

It is also possible that service users deliver an evaluation of the service provided to the SES after they have closed the service session. E.g., they can inform the SES about the average queue length at a certain print server. By that, new QOSPs become possible which allow to consider previous experiences of service users with certain service providers. Examples of such **retrospective QOSPs** are the reliability of service providers or the degree to which the actual service quality corresponded to the negotiated service quality. The collection of data to determine retrospective QOSPs can be hidden from the service users by incorporating it into the procedures by which the service users access the service interface. Once a service session is closed, this data is transmitted to the SES.

Retrospective QOSPs might also help to cope with the problem that claiming to provide a certain QOSP value and really providing it are two different things. As one cannot prove that a certain function is provided unless one has access to all the software which is used for accomplishing it, one has to be satisfied with a statement of quality. This statement, i.e. the submission of a certain QOSP value by the service provider, can be seen as a **guarantee**. If a lack of quality occurs, at least it is clear which system entity is to blame and which programmer may be held liable.

Just as everyday warranties for a device are only valid as long as the device is used according to its purpose, a guarantee given by the service provider will be limited to a certain behaviour of the service user (see also [11]). It may also be restricted by the provider to accommodate other requests in the future. E.g., the service provider may reserve the right to revoke the given guarantee if it has to fulfill another service with a higher priority which is unknown at the time the guarantee is given. Any guarantee becomes obsolete in the case of a hardware or software failure. It is also limited to actions which affect the service quality, but are not under the control of the service providers. E.g., if an operator changes the paper tray of the laser printer so that all of a sudden a guaranteed format is no longer available, the service can no longer be provided.

5. Session Management

To establish a service session, the SES agent has to contact the service provider. For this purpose, every single service has a **session manager** (SM) representing the service provider during the session establishment. The SM first checks if the user is entitled to access the service. If this is the case, it determines the values of dynamic QOSPs and submits them back to the SES. Some dynamic QOSPs may depend on circumstances in the service environment beyond the control of any SM. The paper format available from a print service that was mentioned above is an example of such a parameter. The SM can only determine the values of these parameters and report them to the SES. The situation is different if a dynamic parameter depends on the resources which are controlled by the SM itself, i.e. which are assigned by it to different service sessions. If the SM schedules its resources in a fair manner, all service user will eventually have their service requests executed. This might not be enough for service users which have certain requirements on the **performance** of a service. Performance QOSPs are important in two respects: They apply to all services, regardless of the function of the service, and they are essential for all distributed real-time applications, which form an increasing portion of all distributed systems.

We distinguish between two kinds of services: For **real-time services** reservations are made to provide them with the resources they need, while **best-effort services** are executed according to the capacity available. To determine if a guarantee can be given for real-time services, a worst-case analysis is made, i.e. the maximum workload of all real-time service sessions is considered. In practice, the load will be less heavy, leaving enough capacity for the best-effort services. In the remainder of this section we concentrate on real-time services. Our description corresponds to the mechanisms used in the DASH system to create network sessions for transmitting continuous-media data [12].

The performance of a service depends on the way the competition for resources among service sessions is solved. No service session can be established if its requirements exceed the maximum number of resources available to the service provider. This number, of course, is a static parameter and will be checked before the service provider is even contacted. Let us assume that in no session exclusive access to a resource is required for a longer time than it takes to execute a single service request, i.e. that resources can be preempted between service requests. Then, the service provider can multiplex its resources in time among the service requests of different sessions.

The parameters needed to determine the service performance are the same regardless of the service considered. Whether an SM is able to give certain performance guarantees for a session s, depends on

- the **maximum rate** R_s of service requests (*requests/second*),
- the **maximum execution time** T_s of each request (*seconds/request*), and
- the **deadline** D_s for the execution of service requests (*seconds*).

For the sake of simplicity, let us assume that all service requests of a service session can be handled in the same way so that only one set of parameters for each session is needed. Of course, a more detailed specification would be possible, but one can also simply establish different service sessions if requests have different characteristics. Consider, e.g., a computation service in a process automation system where processes of fixed length are started periodically. One session is used for processes with an execution time of 5 seconds which are started every 20 seconds. Another session is used for processes which last 10 seconds and occur every 30 seconds. In each session, the previous process has to be completed before the next one is started.

Periodical use of resources is typical for real-time applications, hence, the maximum rate of service requests can easily be determined. It can, however, not be determined by the service provider, but has to be stated by the service user when the session establishment is requested. All guarantees given by the SM depend on this statement. If the execution time of a service request depends on parameters of this request it might not even be possible for the service provider to determine the value of this parameter. In a distributed system, messages are used to transmit service requests. For some services it is possible to determine the time of service execution from the maximum size of messages. E.g., the time it takes to print a text is directly proportional to the size of the text. For other services, the time it takes to execute a service request is not a function of the message size. In a computation service the size of the process code provides no hint about the execution time of the process. Here, again, the service provider has to rely on a statement from the service user; its guarantees are valid as long as the information obtained from the user is correct. The service user can use monitoring tools like [13] to determine the worst-case execution time.

Even in real-time applications, service requests may not be delivered at a constant rate. Users may rather issue several requests at a time and then back-off for a while, obeying the maximum message rate in the long run. For these service users a **maximum burst size** B_s (*messages*) is given in addition to the previous parameters [14]. Even if a service user causes bursts, deadlines are calculated as if the regular message rate is obeyed. The service requests are considered to arrive ''ahead of schedule''. Assume, a burst of service requests $b_1, b_2, \dots b_n$ (where $n \le B_s$) is received at time t_0 on a session s. For each request, the **regular arrival time** A (*seconds*) is defined as

$$A(b_r) = t_0 + R_s^{-1}\,(r-1)\ \textit{requests}$$

(Note, that by allowing bursts we do not rely on the RPC property that a client is blocked during a request. This property has been subject to debate in numerous papers, e.g. [15].)

If the SM has already given guarantees for other sessions no new session may be established the service requests of which might violate the previous guarantees, unless the session to be established has a higher priority than these other sessions and causes their abortion. A session may not be established if it exceeds the overall capacity of the service provider. Hence, if S is the set of all sessions established and the new session t, t can only be established if

$$\sum_{s \in S} R_s T_s \le 1$$

holds. Sessions may only be established if each service request of any session can be executed before its given deadline. If W_s is the **maximum waiting time** (*seconds*) for each service request, the relation

$$T_s + W_s \le D_s$$

has to hold for all sessions $s \in S$.

Let us assume, that scheduling for service requests is **rate-based**, i.e. every session gets a share of the resources in a Round-Robin fashion depending on its rate, so that one service request can be executed before the next request is received (considering its regular arrival time). By that, malevolent service users sending requests at a higher rate than previously announced cannot obstruct well-behaved service users. The worst situation for a request on session s would be that all sessions r issue requests at their maximum rate and that all of these requests have to be processed before the request on session s is executed. The request on s can arrive just after the system has started to execute the longest request of all sessions. Let $O = S - \{s\}$ be the set of other sessions, then the maximum waiting time is

$$W_s = \sum_{r \in O} \left(\left\lceil R_s^{-1} R_r \right\rceil T_r \right) + \max_{l \in S} (T_l)$$

If scheduling is **deadline-based**, the situation is slightly different. Let $n = |S|$, then we can define a total ordering $s_1, s_2, \dots s_n$ of sessions according to their increasing deadlines. (Sessions which have the same deadlines are ordered arbitrarily.) The worst case for a service request on session s is that service requests on all sessions with lower deadlines have to be processed. Let s_i be the session considered. In this case $O = \{s_j \mid 1 \le j < i\}$. The same holds for **laxity-based** scheduling, only that sessions are ordered according to $D_s - T_s$.

So far, we have assumed that all service requests are executed sequentially and that they need the resources of the service during their entire execution time. The following two examples illustrate that this assumption is not true for important services:

- A computation service on a multiprocessor is able to execute service requests in parallel. More than one resource (here: a processor) is available for the service.
- In a network service, each message takes some time to propagate through the network until it is delivered. The resource (here: a network module) can execute service requests although previous ones have not been completed.

In a more detailed model, the maximum execution time of a service request consists of three components:

- the **maximum synchronous execution time** ST_s, where only one service request can be executed, e.g., to assign a processor in a computation service,
- the **maximum asynchronous execution time** AT_s, where the service request can be executed in parallel to a certain number N of other service requests (where N results from the resources available so that no competition between service requests occurs), e.g., to execute a computation on its own processor, and
- the **maximum propagation time** PT_s, where no resource is needed for the service, but the service result still is not available, e.g., the time it takes a message to be forwarded through a network.

$$T_s = ST_s + AT_s + PT_s$$

ST, AT and PT do not need to be continuous. Unfortunately, in a worst-case scenario we cannot assume that service requests arrive at times which allow them to be executed in parallel. At least, the maximum waiting time is reduced by the propagation time:

$$W_s = \sum_{r \in O} \left(\left\lceil R_s^{-1} R_r \right\rceil \; (ST_r + AT_r) \right) + \max_{l \in S} (ST_l + AT_l)$$

The service elements of one service may make use of other services. The best example is the SES itself: To be able to contact a remote SM, it has to make use of the network service. To access the underlying services, the SM uses the SES – the establishment scheme is strictly recursive. Just as the requirements on these underlying services may result from the parameters requested and the information given by the original service user, the dynamic QOSP values of a service provider can depend on the services it uses. If a service request in session s uses other service sessions r its execution time consists of the accumulated execution times T_r of these underlying services and an additional time T_s' actually spend for intrinsic functions of the service itself, i.e.

$$XT_s = XT_s' + \sum_{r \in Q} XT_r$$

where X specifies the kind of execution time (i.e. S, A or P) and Q is the set of service sessions s uses.

To establish a session with a remote SM the SES has to contact the SM of the network service, too. One session has to be established for sending service requests and one – if needed – for receiving reply messages. Whereas in the traditional RPC model one request causes one reply, in a real-time environment for continuous media, e.g., one service request to switch on a television channel causes a continuous stream of reply messages. Hence, the session characteristics for request and reply messages can be quite different. While the SES agent of the service user installs a session for request messages, we leave it to the SM of the remote service to establish the other network session. The SM has more information about the QOSP values needed for this connection. Inside the network service, again, it has to be tested if the requested QOSP values are available. A description of the methods applied there can be found in [12]. A similar method is presented in [16] which takes a third kind of network services with statistical properties into account.

6. Conclusion

We have presented a scheme for entities in distributed systems to adjust services according to their needs and possibilities. This scheme is just another step towards more flexibility and autonomy in distributed systems. Its implementation is straightforward and in line with all other service implementations. Particular attention has to be paid to a proper description method of services and their QOSPs in order to avoid excluding some services from being negotiated and to make the specification of requirements easier for the service user. Standardization initiatives like [5] are helpful in this regard.

The user interface of handle sessions consists of only two procedures, one to create a session and one to abandon it. An additional third procedure can be used to ease the renegotiation of QOSP values should the requirements on an already existing session change. In addition to these procedures, the service user has to provide an exception handler in case the service provider has to revoke its guarantees. The service user does not have to deal with these mechanisms unless it decides to do so on its own account. If the user has no specific requirements, sessions with default QOSP values can be established implicitly whenever the user calls a service for the first time. Every user has as much influence on the selection of service providers as it wants to.

Acknowledgements. We would like to thank Domenico Ferrari and David Anderson from the University of California at Berkeley for helpful comments on an earlier draft of this paper.

References

[1] Birrell, A.D., Nelson, B.J.: Implementing Remote Procedure Calls, ACM Transactions on Computer Systems 2, 1, 39–59, 1988

[2] Anderson, D.P., Ferrari, D.: The DASH Project – An Overview, Report No. UCB/CSD 88/405, Computer Science Division (EECS), University of California, Berkeley, January 1988

[3] BERCIM – Bericht zum 2. Meilenstein (Zusammenfassung), Deutsche Telepost Consulting GmbH, DETECOM, BERKOM, Berlin, Mai 1989

[4] Robinson, D.C., Sventek, J.S.: Interface Trading Concepts, Report No. ST.15.01, ANSA Project, Cambridge, August 1988

[5] Methodology for the Specification of QOS Parameters, Draft, NETMAN – R1024, Deliverable 3, RACE Project, February 1989

[6] Oppen, D.C., Dalal, Y.K.: The Clearinghouse – A Decentralized Agent for Locating Names in a Distributed Environment, ACM Transactions on Office Information Systems 1, 3, 230–253, 1983

[7] Peterson, L.L.: A Yellow-Pages service for a Local-Area Network, Proceedings ACM SIGCOMM '87, Workshop on Frontiers in Computer Communications Technology, Stowe, 235–242, August 1987

[8] Knuth, D.E.: The TEXbook, Addison-Wesley, Reading, 1984

[9] Tokuda, H., Wendorf, J.W., Wang, H.-Y.: Implementation of a Time-Driven Scheduler for Real-Time Operating Systems, Proceedings IEEE Real-Time Systems Symposium, San Jose, 271–280, December 1987

[10] Sloman, M.S., Moffett, J.D.: Domain Management for Distributed Systems, Proceedings Symposium on Integrated Network Management, Vol. 1, Boston, Meandzija, B., Westcott, J. (Editors), North-Holland, May 1989

[11] Jones, C.B.: Tentative Steps Toward a Development Method for Interfering Programs, ACM Transactions on Programming Languages and Systems 5, 4, 596–619, 1983

[12] Anderson, D.P., Tzou, S.-Y., Wahbe, R., Govindan, R., Andrews, M.: Support for Continuous Media in the DASH System, Report No. UCB/CSD 89/537, Computer Science Division (EECS), University of California, Berkeley, October 1989

[13] Haban, D., Shin, K.: Application of Real-Time Monitoring to Scheduling Tasks with Random Execution Times, Report TR-89-028, International Computer Science Institute, Berkeley, May 1989

[14] Cruz, R.L.: A Calculus for Network Delay and a Note on Topologies of Interconnection Networks, Report UILU-ENG-87-2246, University of Illinois, July 1988

[15] Chang, C.-C.: REXDC – A Remote Execution Mechanism, Proceedings ACM SIGCOMM '89, Symposium on Communications Architectures and Protocols, Austin, 106–115, September 1989

[16] Ferrari, D., Verma, D.C.: A Scheme for Real-Time Channel Establishment in Wide-Area Networks, Report TR-89-036, International Computer Science Institute, Berkeley, May 1989