NAMING IN THE PEACE DISTRIBUTED OPERATING SYSTEM

M. Sander, H. Schmidt, W. Schröder-Preikschat

GMD FIRST
Berlin, FRG

ABSTRACT

The PEACE naming facility distinguishes between low-level naming, used to address system-wide unique communication endpoints for message passing activities, and high-level symbolic naming, used to identify communicating processes. Symbolic naming in PEACE is application-oriented, providing name space isolation in case of different application programs. This introduces security and avoids the problem of name clashes if a global name space is shared by different applications. The paper describes rationale and concepts of the PEACE naming system.

Keywords: operating system family, message passing, process structured systems, symbolic naming, system-wide unique identifiers, application-oriented name spaces, name space isolation.

1. INTRODUCTION

In order to make distributed systems work, naming and addressing of distributed objects must be provided in a flexible, reliable and convenient way. A distributed application must be enabled to identify its communicating processes in a network transparent manner. Moreover, the identification must be application-oriented, to restrict the communication with processes constituting a different application and to introduce some means of security in terms of name space isolation.

Typically, a distributed computer system will be managed by a distributed operating system. All state-of-the-art distributed operating systems, V (Ref. 3), AMOEBA (Ref. 12), MACH (Ref. 25), CHORUS (Ref. 18), etc., itself can be viewed as a dedicated distributed system application, constituted by a multitude of system processes. The distributed operating system must be capable to identify and address its system processes, leading to a separate system name space. Following the traditional model of user and system mode of execution as provided by todays processors, the system name space is to be protected and, thus, is to be isolated from name spaces related to distributed user applications.

Obviously, being based on these types of distributed operating systems, distributed user applications depend not only on the availability of user application processes. They also depend on the presence and availability of system processes. Consequently, naming is to be used not only to locate and address user processes, but also to locate and address system processes which provide dedicated system services such as file i/o, process management, memory management, and so on. This necessitates the communication between different distributed applications of a computer system, represented by distributed user application programs and the distributed operating system. It also implies that different name spaces must be combined such that user processes are able to identify and address system processes, which is the absolute precondition for system service invocation.

Without sacrificing name space isolation to support communication security, a structured name space is required. The chosen organization must fulfill at least two contradictory demands, name space isolation and name space sharing; whereby the latter mentioned aspect is related to operating system services which are made available to application programs. If the shared name space is structured as well, a model can be realized in which operating system services are only made available to specific application programs. Moreover, this includes that merely those service providing system processes which are required by the application program need to be present. Similar to the construction of program families (Ref. 15), a family of distributed operating systems is made feasible if the naming system is properly organized. Exactly this idea is followed by the PEACE® naming facility.

In addition to message passing, naming is the most basic functionality of PEACE (Ref. 21). Following the pattern of FAMOS (Ref. 7) and MOOSE (Ref. 20), it provides fundamental services for the construction of a family of operating systems. An important aspect was to support dynamic alterable operating system architectures, such as DAS (Ref. 11), and to extend these concepts into the area

of process structured and distributed systems. In this sense, PEACE aimed in the design and development of a process execution and communication environment for distributed user/system applications, rather than providing another distributed operating system. The paper discusses the PEACE naming approach, giving its rationale and explaining its concepts. It is shown by what means in PEACE high-performance message passing, based on system-wide unique communication endpoint identifiers, co-operates with symbolic and application-oriented naming, based on distributed name server.

2. RATIONALE AND CONCEPTS

In addition to provide a basis for a family of distributed operating systems, naming in PEACE is also influenced by a specific hardware organization. The following subsection explains this organization and the impacts on the naming system. Following that, the general concepts of PEACE naming are illustrated, focusing on different types of distribution transparency.

2.1 The SUPRENUM System

PEACE was specifically designed for SUPRENUM (Ref. 6), a largely parallel MIMD (multiple instruction, multiple data) supercomputer based on a distributed memory architecture. SUPRENUM is a scalable multiprocessor system, consisting of up to 16 *clusters* interconnected by a 125 Mbit/sec token-ring network. Each cluster groups 20 processors, so called *nodes*, which are interconnected by two 64-bit parallel cluster busses, each one providing a physical bandwidth of 160 Mbytes/sec. The nodes are distinguished into 16 processing nodes, used for numerical processing, and 4 system nodes, providing services such as disk i/o, cluster diagnosis, routing, internetworking, and so on. A 20 MHz Motorola MC 68020 is used as node CPU, supported by the paged memory management unit MC 68851. Up to 8 Mbytes local (on-board) memory is available on the nodes.

Based on this architecture, PEACE is responsible for the management of up to 320 nodes. However, from the operating system designer viewpoint, the most important characteristic of SUPRENUM is the cluster organization which requires the presence of functional replicated system services. A typical example in SUPRENUM is the disk service, which is to be provided only on the disk node of each cluster and, hence, is local to a cluster. In addition to that, there are node-relative services such as address space and process management as well as global system services such as internetworking and external host access. Thus, in PEACE, service replication is not primarily a requirement of fault tolerance, but rather it is enforced by the underlying SUPRENUM hardware architecture.

Generally, PEACE is concerned with the management of a *functional dedicated and replicated server system*. At the operating system level, a distinction is to be made between local system services and global system services. The SUPRENUM case study especially shows that a two-level approach is not sufficient, rather one has to distinguish at least between node services, cluster services and system services. At the application level, transparency is to be achieved, however without necessarily enforcing transparency at all. Applications should always have the opportunity to choose the type, i.e., scope, of service they want to access. All these aspects significantly influence the functionality of the naming mechanism, because in PEACE, operating system services are referred to by location independent names.

2.2 Separation of Concerns

For performance reasons, PEACE message passing is non-buffered, (i.e., synchronous) and defined between processes. A process is associated with an *absolute address* which designates the communication endpoint specified in the message passing primitives. This address is represented by a numerical value. In order to introduce different types of transparency, at a higher level a process is associated with at least one *relative address*, represented by a symbolic name. The PEACE naming system then is used to map relative addresses onto absolute addresses. Figure 1 illustrates the complete scenario.

A process is addressed by its *system-wide unique identifier* containing a hint on which node the corresponding process object is located. Obviously, network-wide message passing requires no time consuming mapping functions in order to determine the destination address for a message. This improves the overall performance of message passing operations. However, it makes processes dependent on the actual program distribution, because they will have to remember system-wide unique identifiers for later message passing operations. In order to meet the message passing performance requirements for SUPRENUM (Ref. 9), local caching of these identifiers is not implemented by the standard PEACE message passing kernel implementation.

System services are invoked on a *remote procedure call* (Ref. 13) basis, which introduces access transparency, i.e., hides the difference between local and remote operation. Merely the PEACE message passing primitives are invoked in the traditional sense of system calls, namely on a trap handling basis. A procedure name stands for a particular service function which is associated with a relative address in PEACE and to be provided by a server process. Usually, a complete service interface encompasses a set of service functions, thus representing a set of relative addresses for the same server process. Dependent on the type of service, relative addresses may also be related to argument (i.e., object) names. A typical example is the file service, where each file name is given a relative address in PEACE.

A relative address is represented by a *symbolic identifier*, i.e., as a symbolic name. All symbolic identifiers together constitute a global name space. They will be used to generally identify distributed objects independent of the nature of the object. That is to say, symbolic names are used in PEACE to identify devices, nodes, processes, functions, data, and so on. This introduces location transparency, because the mapping function defined between absolute process address and relative process address, e.g., usually is dynamic. It makes dynamical reconfiguration feasible and leads to a flexible system organization, at all.

In PEACE, system services are provided by processes and,

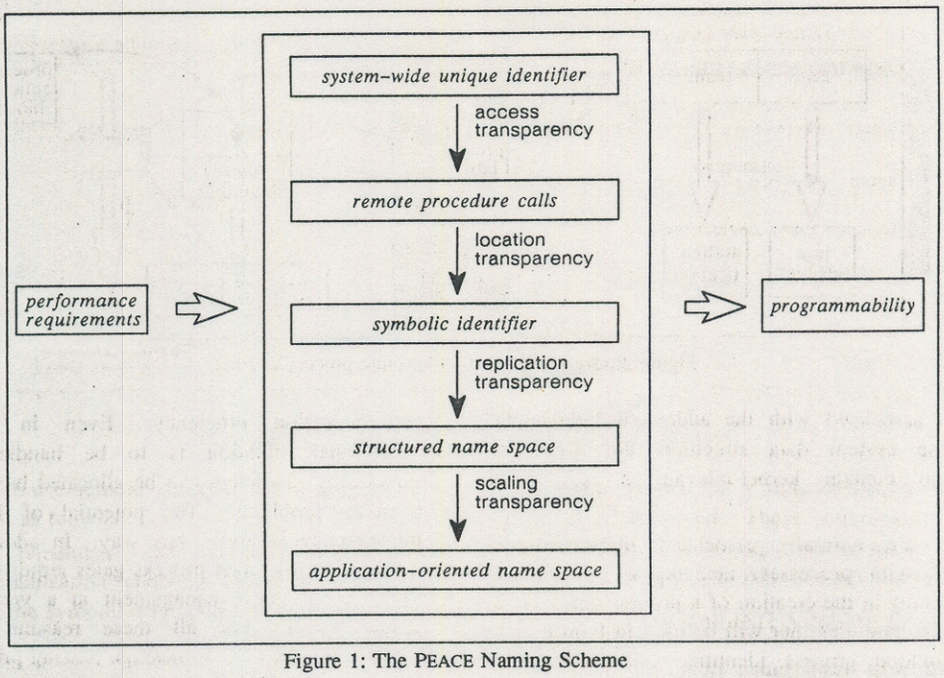

Figure 1: The PEACE Naming Scheme

hence, associated with symbolic names. In case of replicated services, such as the disk service located on the disk node of each SUPRENUM cluster, the same service name is to be used to address different server processes. Within a flat name space, this service name potentially is ambiguous, excepted the node address is encoded within the name – meaning the creation of an absolute address. Hence, replication transparency is to be provided. In order to select the corresponding server process in a convenient manner, a *structured name space* is required. Names of replicated services constitute a specific name space partition. In general, such a partition guarantees the uniqueness of names relative to a specific context and is called a *name plane*. According to the SUPRENUM hardware organization, e.g., PEACE name planes containing local cluster service names are replicated. The sharing of name planes then gives different processes access to the same set of system services.

Scalability is one of the most important characteristics of distributed systems, related both to hardware and software architecture. From the application program viewpoint, scaling transparency is desirable such that program execution works independently of the actual underlying hardware and software organization. Above all, the operating system family concept as followed with PEACE requires a scalable operating system architecture. The idea is that dedicated PEACE system processes provide application-oriented operating system services and that these processes are loaded at the time the distributed application is installed. This leads to an *application-oriented name space*, used to isolate distributed applications and to model the set of system services available for the given application. It also requires a hierarchically structured organization of name planes, building a *name domain* for a specific set of processes.

3. PROCESS LOCALIZATION AND ADDRESSING

PEACE is faced with two contradictory demands; high-performance message passing and location transparency. The former aspect requires an absolute addressing scheme of processes, whereas the latter aspect calls for some means of symbolic naming.

3.1 Absolute Addressing

In order to perform message passing, the kernel addresses processes via port-like system objects. For this purpose, each process is associated with a *gate*, which acts as a communication endpoint without buffering capabilities, but rather routing capabilities. A communication endpoint is referred to by a *system-wide unique identifier*, i.e., an absolute address, as illustrated in figure 2.

As shown in the figure, the communication endpoint address is represented by a low-level path name constituted by the triple {*host, team, lightweight process*}. As with THOTH (Ref. 2), each process in PEACE is member of a *team*. The team concept provides a common execution domain for several *lightweight processes*, meaning efficient support for memory sharing, concurrent program execution, asynchronous communication, etc.

Giving a PEACE system-wide unique identifier, the team membership of a lightweight process as well as the host membership of a team is determined without performance limiting mapping overhead. This identifier enables fast low-level localization and addressing of process objects within a network environment. The *host* field of this identifier is used to distinguish between local and remote message passing and to determine the physical host address of the corresponding process gate. By means of the *team* field, authentication is enabled; a prerequisite for the selection of the local process gate table. The *lightweight process* field then is used as gate table index, whereby a single entry refers to the system data structure (i.e., process

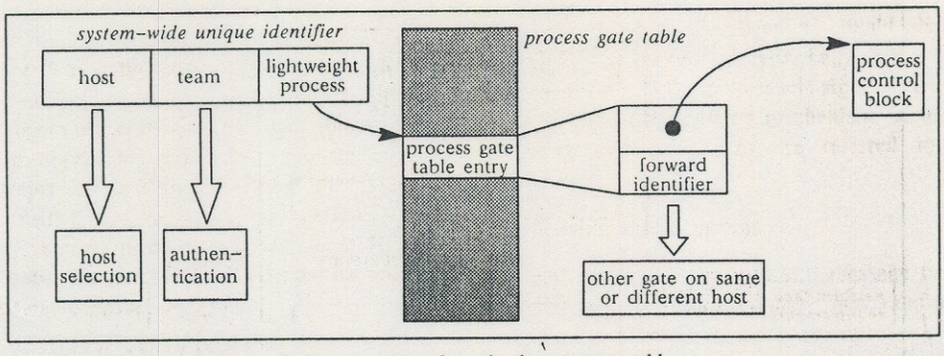

Figure 2: Organization of an absolute process address

control block) associated with the addressed lightweight process. These system data structures are used for scheduling and contain kernel-relevant process state information.

Process gates are normally associated in one-to-one correspondence with processes, i.e., the creation of a process also results in the creation of a process gate. As a consequence, the gate identifier will be used in PEACE as a system-wide unique process identifier and vice versa. Gates can also be associated dynamically with processes, thus allowing dynamic reconfiguration in case of team migration. There might be several gates associated with the same process. According to the information stored in the process gate, the kernel automatically routes an incoming message to the receiving process. In this sense, a process gate can be viewed as a *forward identifier* for a specific process. In case that the process object is present, the process gate contains the memory address of the corresponding process control block. Otherwise, a system-wide unique identifier refers to the process gate which is to be used instead of the currently selected one.

3.2 Symbolic Naming

Absolute addressing at the kernel interface implies loss of location transparency. On the other hand, it implies an utmost efficient message passing implementation. Location transparency can be achieved at the kernel level only if processes deal with relative communication endpoint identifiers. Concerning PEACE – as well as other distributed operating systems –, this implies the following:

- different teams must be allowed to use the same communication endpoint identifier for the addressing of different processes.
- a team should know only local communication endpoint identifiers.

On this basis, it will be the task of the kernel to map team-relative communication endpoint identifiers onto system-wide unique process identifiers. A tradeoff between communication efficiency and scaling transparency exists. The efficient solution at the kernel level calls for fixed-size and team-relative mapping tables, which implies scaling problems in cases where the number of communicating processes exceeds the number of map table entries. The scalable solution at the kernel level calls for arbitrary sized and team-relative hash lists of forwarding identifiers, which implies resource management problems and loss of communication efficiency. Even in this case, the exceptional situation is to be handled if no more forwarding identifiers can be allocated because of memory resource problems. The potential of loss of location transparency is given, any way. In addition to that, the replication of cached process gates requires precautions for cache consistency management at a very low operating system level. For all these reasons, the functional specification of PEACE message passing primitives excludes a guarantee for location transparency.

In order to keep kernel complexity small and achieve high-speed interprocess communication, location transparency is not exclusively supported by the kernel, but rather in co-operation with symbolic naming functions provided by dedicated system processes, so called *name server*. As with any other PEACE system service, these naming functions are invoked on a remote procedure call basis, using PEACE message passing primitives. Obviously, a name server is to be associated with a system-wide unique communication endpoint identifier. In PEACE, each team, and thus each lightweight process of the team, is bound to a *domain identifier* which is the system-wide unique name server identifier. This approach either enables the sharing of a name domain, in case of identical domain identifiers for different teams, or leads to a complete isolation of name domains, in case of different domain identifiers for different teams.

The domain identifier, i.e., the system-wide unique name server identifier, of a given team is determined by querying the kernel, which always is a node-relative PEACE system service. This kernel service is provided on a remote procedure call basis, too. However, the naming service cannot be applied to locate this kernel service, rather a fixed and absolute process address is needed. This address always is associated with the *ghost* (Ref. 21), bound to the communication endpoint identifier $\{host, 0, 0\}$. Note, the *ghost* is the only PEACE process with a fixed address.

Symbolic names are used to designate distributed objects (i.e., processes), which requires the mapping between relative and absolute addresses. This mapping will be performed by a name server, however without having any knowledge of the semantic of the name. Figure 3 shows the principle mapping function.

Each symbolic name is associated with user-defined numeric values. This user information consists of three elements, defined at name creation time. For example,

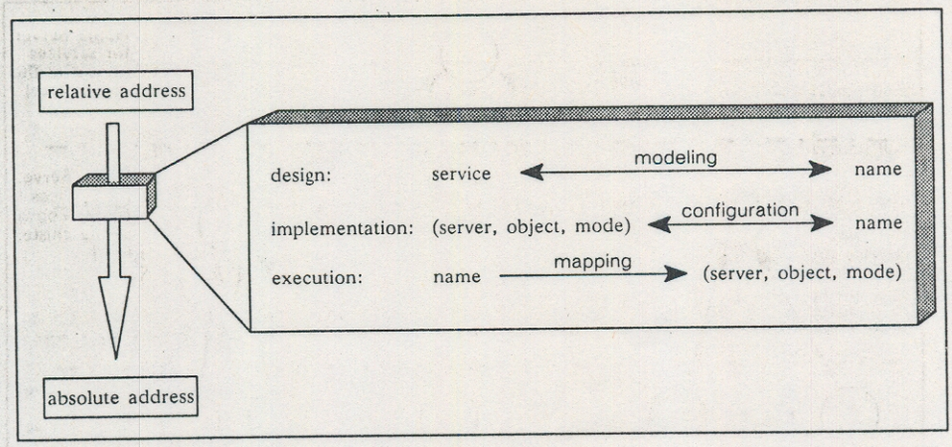

Figure 3: Mapping between relative and absolute addresses

concerning the PEACE remote procedure call system, the elements are used as follows:

server: To remember the system-wide unique communication endpoint identifier associated with the symbolically addressed server process.

object: To remember a server-relative object identifier.

mode: To remember object-specific access modes or types.

This way, various strategies for the addressing of server processes are supported. A server process can be addressed either by a server name, meaning one-to-one correspondence between relative and absolute identifier, or by a service/object name, meaning many-to-one correspondence between relative and absolute identifier.

Creation and destruction of names is dynamic. Typically, at initialization time, a server process exports its services by creating corresponding names. Name mapping is done at run time, too. For example, the first time a system service is invoked, the corresponding stub routine (on behalf of the calling process) requests name resolution and caches the information associated with the name. This way, system-wide unique server identifiers are cached within the team context of the calling process. Cache updates then are to be performed by the processes itself, as part of handling naming exceptions. The exception handler (i.e., the process itself) requests name resolution, again.

Usually, name cache updates are to be performed in cases of process termination, server migration, service migration and object migration. Cache consistency relies on the assumption that the per-team exception handler at the naming library level will perform the update, invisible to the application team. This strategy may lead to a temporary cache inconsistency, caused by the latency of exception propagation. Because all types of migration exceptions will result in the change of the server value stored with a name, the process gate associated with the server process at the kernel level is reconfigured accordingly, i.e., a forwarding identifier for the process is established. In addition to that, service and object migration then is made feasible by temporarily monitoring all requests directed to the original server process. Requests referring to either migrated services or migrated objects are redirected accordingly. If either the caches have been updated or a timeout occurs, the forwarding identifier is destroyed. These activities are controlled by the reconfiguration management system of PEACE and, usually, are independent from the naming system.

4. MODELING A NAME SPACE

In addition to support name space isolation, a structured and application-oriented name space is necessary in order to manage a functional dedicated server system as well as a family of operating systems. The global PEACE name space is implemented by a multitude of name server, each one controlling a single name plane. The resulting organization is application-oriented, defining a unique *name domain* for each distributed application.

The structure of name domains is influenced by several aspects. Concerning PEACE and the SUPRENUM architecture, these aspects comprise:

- the organization of a distributed application;
- the mapping of the application onto the underlying hardware system;
- the isolation of name spaces for different applications;
- the operating system services exclusively related to the application;
- the global operating system services.

It was one of the major requirements in the design and development of the PEACE naming system to cover all these aspects with a single mechanism. This mechanism is based on a generic name server implementation supported by a dedicated naming library.

4.1 Structured Name Space

On the basis of PEACE naming, replication transparency is provided in order to arbitrarily map distributed application programs onto the underlying hardware architecture. In case of replicated system services, the name space is arranged such that a team by default selects the most nearest server process. As illustrated in figure 4, this is accomplished by a hierarchically structured set of name server.

In case of SUPRENUM, the PEACE system name space is tree-structured and shows for several node name server,

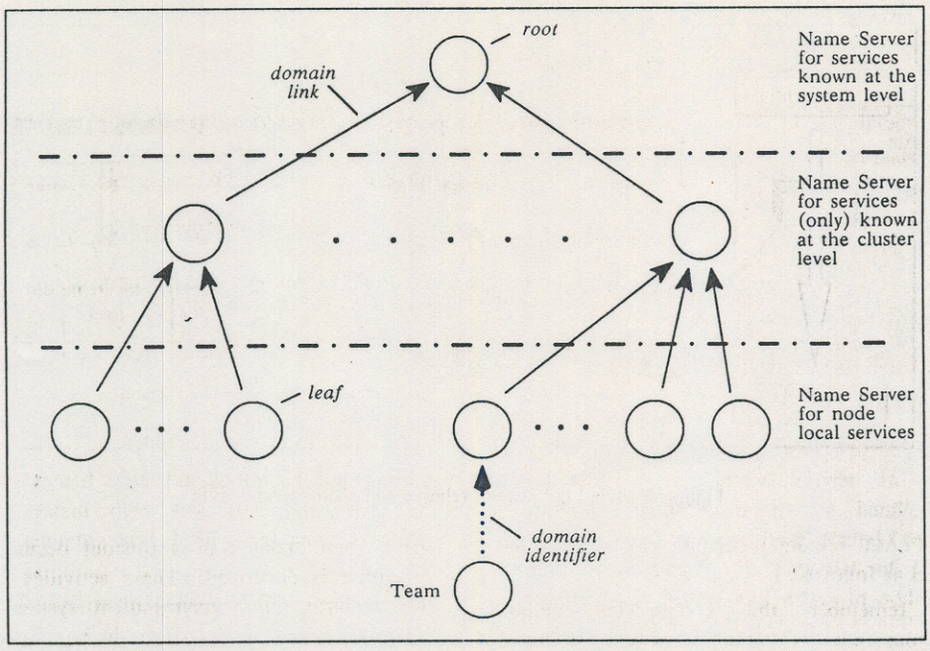

Figure 4: Modeling a name space for replicated system services

some cluster name server and at least one system name server. Each type of name server stores names of services which are available at the given level. In addition to local service names, the linkage to the next higher name server is remembered. This linkage, called *domain link*, is nothing else than a symbolic name designating another name server of the tree-structured name space.

Performed by the naming library (i.e., on behalf of the calling process), the name space is searched for a specific service name following the pattern of scope rules often found in block-structured programming languages. If the name is not yet locally cached by the requesting team, the search is directed to a name server. In the very first case, this name server represents the leaf of the name tree and is addressed by the domain identifier of the requesting team. In subsequent cases, the system-wide unique name server identification is obtained from the name space itself.

If a service name was found, it will be cached and associated user information will be returned. Otherwise, the currently selected name server is requested to resolve (i.e., dereference) the domain link. Note, the domain link may be different for different applications and will always be interpreted as a name server name. If the domain link lookup results in a match, the system-wide unique identification of the associated name server is delivered. To this name server the original service name lookup request is issued, again. If there is no domain link match, the root of the name tree is reached and the requested service name is declared as unknown, meaning that the associated service is not available.

This simple name lookup strategy allows for the dynamical integration of further name server processes. In addition to that, it enables the definition of name scopes which are not necessarily associated in one-to-one correspondence with the underlying hardware organization, such as required for SUPRENUM. Logical clusters of system services can be built, supporting the operating system family concept. Independently of the actual system representation, the remote procedure call layer applies the user information returned by a successful name space search to direct a service request to a server process. In PEACE, a "*third party connect facility*" will be used to model the clustering of system services, without effecting the original application programs.

4.2 Application-Oriented Name Space

The structured name space discussed so far implements location and replication transparency for the invocation of operating system services. In case of supporting the concurrent execution of distributed applications, name space isolation is required. This aspect is considered in PEACE to give distributed applications the opportunity for location transparent addressing of application processes. However, this addressing scheme must be application-oriented to avoid the potential of name clashes, if different applications apply the same name with a different meaning, and to maintain security, in that intruder processes are prohibited to join the application.

In PEACE, an application-oriented name space is implemented by one or more name server. Basically, the same naming strategy can be applied by the application processes as was explained with the lookup of service names. That is to say, the name space is built by a set of name server which are interconnected by a unique and application-dependent domain link name. Most importantly, a separation between user names and system names is to be made. For this purpose, a dedicated name server is used, the so called *domain server*. Figure 5 illustrates the integration of the domain server into the structured PEACE name space.

Following the concept of abstract data types (Ref. 10), the domain server provides the same interface as the name

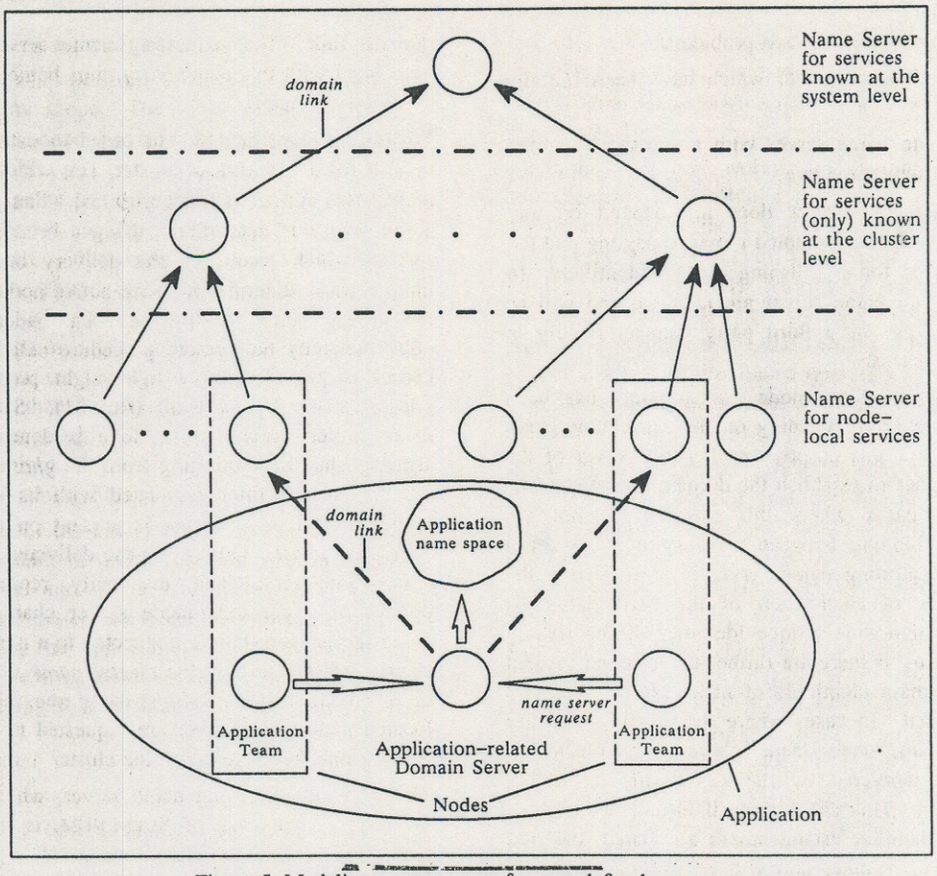

Figure 5: Modeling a name space for user-defined names

server, i.e., it actually is a more dedicated name server implementation, specifically designed for the demands of a specific distributed application. Its primary purpose is to merge user name space and system name space, thus building the application-oriented name space. Similar to the object inheritance approach (Ref. 8), the domain server inherits all the characteristics of a name server and introduces application-oriented naming functionalities.

As explained in the previous subsection, name resolution starts with the name server addressed by the per-team domain identifier. If for a team an application-oriented name space is defined, because it shares a name space with other teams of the same distributed application, this identifier addresses a domain server, instead of a name server. Thus, transparently to the naming library, requests are directed to the domain server of the requesting team. Having received a request, the domain server executes it within a specific application-context. The requested name service function then is applied on an application-oriented name-tree, instead of a single name plane. For example, in case of a name lookup request, the application name space is observed. The result of this request is returned to the naming library. If a name match was indicated, name resolution terminates successfully. Otherwise, name resolution is continued with a name server which is addressed by a domain link. That is to say, the same name lookup strategy is performed at the naming library level, independently whether a name server or a domain server is requested.

Depending on the configuration, both domain identifier and domain link may refer either to a name server or to a domain server. This enables the construction of arbitrary name space structures. It also enables the transparent integration of dedicated naming strategies based on broadcasting or multicasting (Ref. 4). In addition to that, strategies may be introduced to maintain consistency of distributed name caches, because the domain server is related to a specific distributed application and, therefore, knows all the application teams where a name cache is present. In case of the necessity of name cache updates, the domain server raises a naming exception and, eventually, establishes forwarding identifiers as long as the update is in progress.

5. NAME SPACE CONSTRUCTION

The construction and definition of an application-oriented and hierarchically structured PEACE name space is dynamic. It will be performed at the time where either the entire system, a server or the distributed application is installed. In each case, an association between name server and teams is to be established. This will be done by a *third party*, transparently to the selected teams.

5.1 PEACE Domain Installation

At global system initialization time, i.e., after having finished the bootstrap procedure, the *PEACE domain* is to be installed, meaning to create a tree-structured name space which encompasses the names of initial PEACE system

services. Basically, PEACE domain installation is concerned with the following two problems:

- to find all name server which have been initially loaded;
- to associate name server with a specific hierarchy in the system name tree.

Note, a PEACE name server does not depend on any operating system service, excepted message passing and the kernel service used for establishing domain identifiers. In addition to that, all name server are identical and will be used at the "mercy" of a third party connect facility to configure name spaces.

5.1.1 Name server. Once node bootstrapping has been finished, all name server residing on the same node chain up each other. For this purpose, each name server of the particular node tries to establish the domain identifier of its own team. In cases where this identifier is not yet established, the binding between all existing teams of a node and the requesting name server is performed – the domain identifier of each team of the particular node becomes the system-wide unique identifier of the issuing name server. Thus, if there are further name server present on the node, domain identifiers of name server teams are established as well. In cases where the domain identifier of the issuing name server team is already established, it merely will be delivered by the kernel and no global changes are made. This will happen if further requests for global domain identifier establishment are stated; the first name server is the winner and will be used as the *node name server*.

In those cases where the domain identifier was already established, a request for name plane interconnection is directed to the name server which is addressed by that identifier, as shown in figure 6. Name plane interconnection is based on a domain link, i.e., it is given a symbolic name and is associated with a system-wide unique name server identifier. Each name server which receives a request for name plane interconnection tries to create a name entry which represents the domain link. If the corresponding symbolic name is known, the requesting server is directed to re-issue the original interconnection request to the name server which is associated with the name entry. This way, the request will be relaied to a name server which is able to create a name entry for the domain link. The requesting name server then will be associated with this name entry, thus being appended at the name server chain.

5.1.2 Tree construction. In order to establish a naming system for a SUPRENUM cluster, i.e., which spans a set of nodes, two functions are performed. First, the number of active nodes is determined, using a broadcast-like system service which results in the delivery of a system-wide unique *ghost* identifier for each active node. Note, at least the PEACE kernel must have been loaded on an active node, whereby the remote procedure call interface of the kernel is provided by a lightweight process, called the *ghost*, of the kernel team (Ref. 21). Second, the node name server, if any, of a node is determined. This is accomplished by requesting from the *ghost* the delivery of the domain identifier associated with its kernel team. If undefined, no name server is present on the node of the addressed *ghost*. Otherwise, the delivered identifier refers to a name server and, eventually, represents the head pointer of a per-node name server chain. Each located name server then will be connected to a name server which is selected to represent the *cluster name server*, leading to the installation of a two-level name tree. For this purpose, located node name server are requested to create a PEACE domain link which refers to the cluster name server.

The next encompassing name server, which is the *system name server* in case of SUPRENUM, is installed by first requesting any node name server of a cluster for the resolution of the PEACE domain link. As was explained above, this link refers to the cluster name server. The selected cluster name server then is requested to create a PEACE domain link which refers to the system name server. This procedure is repeated to establish further name tree hierarchies accordingly.

5.2 Installation of Name Scopes

In case of replicated system services, the *name scope* of each service is to be manifested. A service might have local or global relevance, leading to different scopes of a particular service. For example, based on SUPRENUM, a distinction is to be made between services related to a node, a cluster or the entire system. For each of these

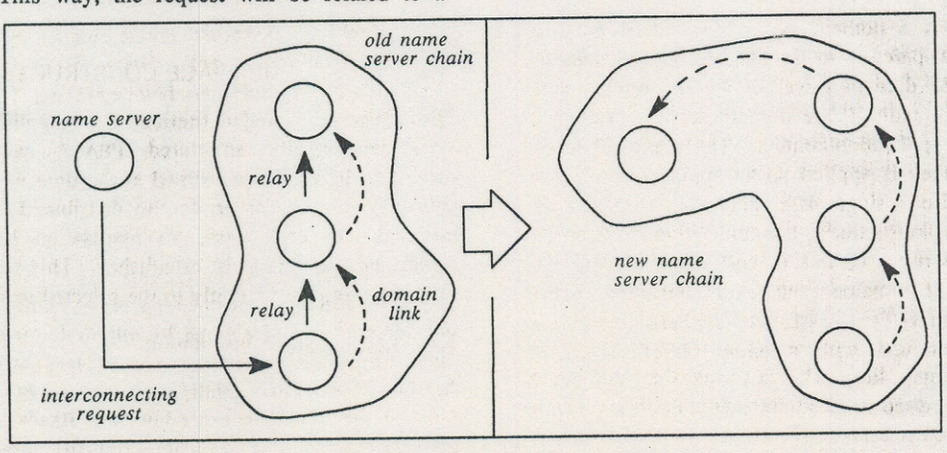

Figure 6: Chain up of per-node name server

contexts, a name server is used to keep track of context-relative services names. The scope of a particular service then is represented by a proper name server.

Once a server team has been installed, its services are to be bound to a specific scope. The scope associated with a server team is managed by a dedicated PEACE system process, the *scope server*. For each server team, the scope server selects a name server which is to be used for requesting the creation of service names. Both, the new server team as well as the name server are identified by symbolic names. Thus, the scope server maps the symbolic name of the server team, usually the corresponding program name, onto a symbolic name used to designate a name server, i.e., name-tree hierarchy. The name server name then is located within the name domain of the new server team. As explained previously, this follows the default PEACE name lookup strategy being performed by the naming library.

The name domain of the new server team may correspond to the PEACE domain, in which case a new global PEACE system service is introduced. The PEACE domain is extended by a new system service. In contrast to that, the name domain may be application-oriented, meaning the installation of a new application-oriented system service. The PEACE operating system family is extended by a new system member.

5.3 The Third Party

The installation of PEACE domain and name scopes is controlled by dedicated system processes, representing the third party connect facility. These processes, so called *conductor*, are loaded at system bootstrap time. Dependent on the types of conductor processes, different name space configurations are established. Thus, for SUPRENUM there is a dedicated conductor used to establish the cluster name space, while another conductor establishes the system name space.

Name space configuration is directed by information obtained from a *configuration specification* written in a dedicated system configuration language. This specification is interpreted by a conductor and executed accordingly. A low-level *bootstrap description* specifies what teams are loaded on what nodes in a distributed computing environment. This way, conductor processes are placed on specific nodes. Relative to these nodes, they construct the name space according to the configuration specification.

At run time, the *loader* is not only responsible for placing teams (i.e., programs) onto nodes, but also for the establishment of the domain identifier of the teams. In case of extending a distributed application, a new team will be bound to the domain identifier of that application, enabling name space sharing. For this purpose, in addition to the program file name, the domain identifier of the new team is passed to the loader too.

6. RELATED WORKS

In addition to introduce certain levels of transparency in a distributed computing environment, naming is also a mechanism for synchronization (Ref. 17). For example, this aspect is of importance for installing the PEACE domain. The domain identifier state of a kernel team specifies if a node name server has been installed. In a similar way, the domain link state in a node name server specifies if a cluster name server has been installed, and so on. That is to say, these identifiers are used in PEACE to synchronize on external events, e.g., caused by conductor processes.

Recently, several works on naming and resource management in distributed systems have been published. However, most of these works are concerned with naming at a very high level of abstraction. For example, in (Ref. 1, 5, 14, 16, 22, 24) naming services are described which rely on a complete operating system environment. This is in contrast to PEACE naming, which is designed to model a family of distributed operating systems and, hence, will be used to implement fundamental operating system services.

From its level of abstraction, naming in PEACE will be comparable to naming required for the construction of object oriented operating systems (Ref. 19). However, PEACE does not claim to be an object-oriented operating system, although it is constructed following object-oriented design principles.

Concerning the switching between name domains, PEACE follows an approach as discussed in (Ref. 23). That is, from its functionality, the global name service of HCS is comparable to the PEACE domain server. However, the naming service introduced there, again, is layered on top of a complete operating system interface and is specifically designed to interconnect heterogeneous computer systems.

As an example for a naming approach which supports the restructuring of operating systems is explained in (Ref. 4). This approach is based on broadcast techniques and takes advantage of V group communication. Multicasts are used to query system services. In case of replicated services, this results in the delivery of multiple responses and the client is burdened to choose between one of them. For performance reasons, this approach is not followed in PEACE because the SUPRENUM network provides no broadcast service. Because of the limited number of fixed multicast addresses (i.e., group identifiers), the construction of arbitrary structured and application-oriented name spaces is not well supported.

7. CONCLUDING REMARKS

Aim of the naming approach explained in the paper is to support an operating system family where members are represented by server processes. An evaluation of this approach is in progress, now. Presently, the PEACE naming system is not used to support a dynamically alterable operating system structure because system services which provide team migration, checkpointing and recovery are still under development. However, PEACE domain installation as well as the dynamical construction of application-oriented name spaces works.

In the near future, PEACE activities will focus on mechanisms for dynamical configuration and reconfiguration of distributed applications. Especially, a message passing kernel family is to be supported, which is much more ambitious than providing support for a process-structured operating system family. The members of the kernel family range from a single-process to a multi-team mode of operation, each one offering different

communication performance parameters. Depending on distributed SUPRENUM applications, the most efficient message passing kernel will be loaded along with the application. This also includes the dynamical restructuring of a system in cases where a single-process kernel is to be replaced by a multi-team kernel.

8. ACKNOWLEDGMENTS

The authors wish to express their gratitude to all members of the SUPRENUM project, above all Peter Behr. Concerning PEACE, we especially would like to thank Ralph Berg, Lutz Eichler, Jörg Nolte, Bernd Oestmann and Friedrich Schön for their very high qualified contributions related to all steps of design and implementation of a distributed operating system.

This work was supported by the Ministry of Research and Technology (BMFT) of the German Federal Government under grant no. ITR 8502 A 2.

® PEACE is a registered symbol of GMD.

9. REFERENCES

1. Birrell A D, Levin R, Needham R M & Schroeder M D 1982, Grapevine: An Exercise in Distributed Computing, *Comm. ACM*, 25, 4, 260-274
2. Cheriton D R 1979, Multi-Process Structuring and the Thoth Operating System, *UBC Technical Report* 79-5, University of Waterloo
3. Cheriton D R 1984, The V Kernel: A Software Base for Distributed Systems, *IEEE Software* 1, 2, 19-43
4. Cheriton D R & Mann T P 1986, A Decentralized Naming Facility, *Technical Report* STAN-CS-86-1098, Department of Computer Science, Stanford University
5. Dyer S P 1988, The Hesiod Name Server, *Proc. USENIX Conference*, Dallas, Texas
6. Giloi W K 1988, The SUPRENUM Architecture, *Proc. CONPAR 88*, Manchester, UK.,
7. Habermann A N, Flon L & Cooprider L 1976, Modularization and Hierarchy in a Family of Operating Systems, *Comm. ACM*, 19, 5, 266-272
8. Halbert D C & O'Brien P D 1987, Using Types and Inheritance in Object-Oriented Languages, *IEEE Software*, 9, 71-79
9. Kolp O & Mierendorff H 1987, Performance estimations for SUPRENUM systems, *Parallel Computing* 7, 357-366
10. Liskov B H & Zilles S 1974, Programming with Abstract Data Types, *SIGPLAN Notices*, 9, 4
11. Müller J, Habermann A N & Löhr K P 1980, Address Space Management in the DAS Operating System, *Technical Report* 80-35, TU Berlin
12. Mullender S J & Tanenbaum A S 1986, The Design of a Capability-Based Distributed Operating System, *The Computer Journal*, Vol. 29, No. 4
13. Nelson B J 1982, Remote Procedure Call, *Report* CMU-CS-81-119, Carnegie-Mellon University
14. Oppen D & Dalal Y 1981, The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment, *Tech. Rep.* OPD-T8103, Xerox Office Products Div., Palo Alto, Calif.
15. Parnas D L 1975, On the Design and Development of Program Families, *Report* BS I 75/2, TH Darmstadt
16. Peterson L L 1988, The Profile Naming Service, *ACM Transactions on Computer Systems*, 6, 4, 341-364
17. Reed D P 1978, Naming and Synchronization in a Decentralized Computer System, *Technical Report* MIT/LCS/TR-205, MIT, Cambridge, Massachusetts
18. Rozier M et al 1988, CHORUS Distributed Operating Systems, *Computing Systems Journal*, Vol. 1, No. 4
19. Schantz R, Schroeder K & Neves P 1987, Resource Management in the Cronus Distributed Operating System, *ACM Computer Communication Review*, 17, 5, 243-244
20. Schröder W 1986, Ein Familie von UNIX-ähnlichen Betriebssystemen – Anwendung von Prozessen und des Nachrichtenübermittlungskonzeptes beim strukturierten Betriebssystementwurf, *Ph.D. Thesis*, TU Berlin
21. Schröder-Preikschat W 1989, PEACE – A Distributed Operating System for High-Performance Multicomputer Systems, to be published in *Lecture Notes in Computer Science*, Springer-Verlag,
22. Schroeder M D, Birrell A D & Needham R M 1984, Experience with Grapevine: The Growth of a Distributed System, *ACM Transactions on Computer Systems*, 2, 1, 3-23
23. Schwartz M F 1987, Naming in Large, Heterogeneous Systems, *Technical Report* 87-08-01, Department of Computer Science, University of Washington, Seattle
24. Skinner G, Wrabetz J M & Schreier L 1987, Resource Management in a Distributed Internetwork Environment, *ACM Computer Communication Review*, 17, 5, 254-258
25. Young M et al 1987, The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System, *ACM Operating Systems Review*, 21, 5