MAGNET: A Virtual Shared Tuplespace Resource Manager

Patty Kostkova and Tim Wilkinson
High Performance Extensible Systems Group (HiPeX)
Inter-Operability Research Centre
City University
London, UK
{patty,tim}@sarc.city.ac.uk

Abstract

Traditional operating systems limit flexibility, performance and utilization of hardware resources by forcing applications to use inappropriate high-level abstractions, uniform protection schemes and high-level static resource management. This forced use of inappropriate services results in poor application and operating system performance. A radical new approach to operating systems design and construction is needed to meet the requirements of modern applications. Within our Centre, we are building BITS: the Component Based Operating System, to address these issues.

To realize its full potential, BITS requires a radically new resource management strategy. The operating system design gives an environment for implementing extensions, but a resource manager module is responsible for making them available. It allows system services to be specialized, replaced or extended to better serve application-specific needs.

In this paper we propose the MAGNET Resource Manager enabling a free-market negotiation of application requests and server resources. It provides an additional level of flexibility for application participation in resource management. MAGNET also provide an ideal platform for an additional runtime level of extensibility: dynamic modification and replacement of its parts during execution.

1. Introduction

Operating systems form the interface between system resources and applications by providing abstraction of hardware resources, protection of application and resource management.

The role of operating systems has significantly changed, with the ever increasing use of heterogeneous, distributed systems offering enormous computational power and new application requirements - higher flexibility, extensibility and tailoring services to application specific demands. Traditional operating systems, including microkernels, limit flexibility, performance and utilization of hardware resources by forcing applications to use inappropriate high-level abstractions, uniform protection schemes and high-level static resource management. This forced use of inappropriate services results in poor application and operating system performance. Applications really require a platform where they can implement their own abstractions, tailor existing servers to their needs, utilize available hardware efficiently, define their own protection schemes and participate in resource management policies. A radical new approach to operating system design and construction is required to enable these new requirements to be met.

One recent approach to be introduced in all areas of software design is "Components" [4]. We believe that this methodology can be adapted for use as the basis for an operating system, enabling us to provide the desired resource management flexibility and dynamic adaptation of resource-application interfaces. Within our Centre, we are building the BITS Component Based Operating System, which aims to provide an environment that meets the above application requirements more precisely. By pushing the hardware abstractions to the lowest level and by binding trusted applications to servers through indirect calls, BITS ensures significantly better performance. Untrusted applications pay a performance penalty because of the additional protection overhead. By enabling application defined protection schemes, BITS provides an extra level of flexibility over classical extensible systems. Through its component structure and use of late binding, BITS allows a finer-grained interface to system services.

The operating system design provides an environment for implementing application requirements for extensibility, but a resource manager is the essential component enabling system services to be specialized, replaced or extended to better serve specific application requirements. This paper provides a detailed description of the MAGNET Resource Manager. MAGNET utilizes

1060-3425/98 $10.00 (c) 1998 IEEE

a "Virtual Shared Tuplespace" (physically distributed tuplespace presented by a Distributed Shared Memory Server (DSMS) as a shared memory tuplespace), to satisfy the resource requirements for the widest range of applications, maximizing hardware utilization and enabling application participation in resource allocation. This brings an additional level of flexibility over traditional resource managers. In addition to these features, MAGNET; cooperating with applications to ensure consistency; enables an additional level of runtime extensibility: dynamic modification and replacement of its parts during execution (e.g. plug and play, hot swap). This allows failed elements to be replaced with alternatives providing an equivalent service level.

In this paper we give a detailed description of the MAGNET approach, our motivations, the design issues and discuss its features and potential. In Section 2 we provide a more detailed case for MAGNET as an essential part of BITS. Section 3 is devoted to a detailed description of both BITS and MAGNET architectures. Section 3.1 briefly discusses the BITS Component Based Operating System approach, the system structure and its computing environment, to give a brief overview of the system MAGNET has been designed for. In Section 3.2. we describe the MAGNET design issues in detail. Next, Section 3.3. discuss our Virtual Shared Tuplespace design. Problems and future work are discussed in Sections 4. and 5, related work being presented in Section 6 and, conclusions in Section 7.

2. Motivations

Contemporary operating systems research has focused on the design of extensible kernels. These are designed to overcome the main microkernel limitations of poor performance, centralized, inflexible resource management, and the inability to specialize or extend high-level abstractions to application demands [3].

One of the major factor limiting both system flexibility and performance is fixed high-level abstraction. It has become clear that requirements of all applications cannot be met by any operating system in advance [2]. It is for this reason current research explores better structured abstractions [3] to enable finer granularity in system services. [1] shows that presenting real hardware to applications that require low-level access allows them to tailor their specific abstractions enabling better performance. BITS' abstraction approach is to implement just the protection schemes necessary for system security, and leave the implementation of high-level abstraction to servers which applications can use, if appropriate. Applications can also implement their own abstraction at all levels including the raw-hardware level. This approach brings both flexibility and offers enhanced performance.

Another observation made by research on current microkernels is that cross-domain control transfers and protection checks hurt system efficiency. BITS supports application defined protection schemes. In this environment trusted applications can access services through indirect calls giving them shared library performance. Untrusted applications communicate through standard protection checks. We term this binding mechanism the **"Glue"**.

The goal of resource management in distributed environments has been extended from satisfying application demands and resource allocation to enabling sharing of resources and supporting reconfigurability and fault tolerance. The resource manager actually supports extensibility, such as tailoring policies to given hardware configurations and application specific requirements. This cannot be simply implemented by heavyweight user-level servers as demonstrated by microkernel research. The following two enhancements enable large improvements in flexibility and extensibility:

- Separation of kernel services from resource management policies. Microkernel research has shown that moving servers to the user level ensures a level of flexibility, but does not support extensibility.
- Separation of resource protection from resource management policies allows extensibility and application specific customization.

BITS implements these techniques providing an environment to enable the MAGNET Resource Manager to further improve flexibility by encouraging applications to influence resource allocation strategies. A Virtual Shared Tuplespace introduced above is used to implement a free-market for the efficient matching of application needs with available system resources. Although the tuplespace is distributed across system physical memory, BITS enables DSMS to tailor the implementation to MAGNET's requirements. An additional level of flexibility is added by presenting a framework for dynamic modification of services at run-time (hot swap, plug and play) and by enabling the run-time replacement of failed components by equivalent ones with application agreement.

3. BITS and MAGNET Architectures

In order to present the MAGNET architecture design we briefly discuss BITS' design as a framework for the resource manager.

3.1. BITS: the Component Based Operating System Approach

The BITS Component Based Operating System [4] consists of a set of independent **Components** representing system elements. From an application point of view, all components are treated equally, each providing some different level of service. Figure 1 shows a typical application view of the system.

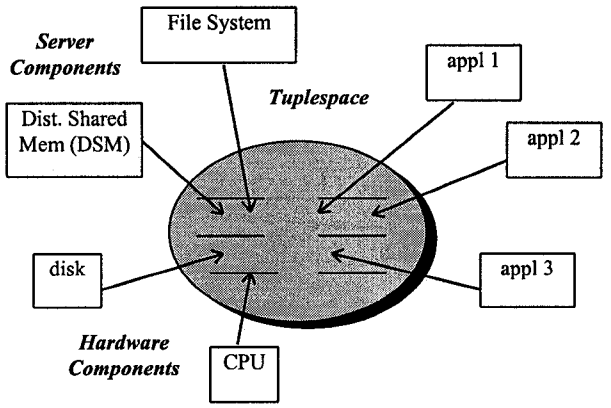

Figure 1. Component Based System Structure from an application point of view

From the operating system point of view, components can be divided into two significant groups:

- **Hardware Components** representing hardware devices, and
- **Server Components** representing user-level abstraction servers.

This restructuring of abstraction enables applications to choose the abstraction level and model they want to access the hardware through. The raw hardware devices in Figure 1 (CPU and Disk) provide the base level **Hardware Components**. These components implement their own protection schemes in order to control access and prevent starvation. We term the protected hardware elements **Virtual Hardware**.

The second group of components, the **Server Components**, represent diverse services such as file systems (FS), distributed shared memory (DSM), TCP/IP, Database Management System (DBMS), etc. providing high-level abstractions of the hardware components. Typically, user applications will use components which provide value added services (with an associated performance penalty) rather than the hardware components. However, when speed is essential or an application requires a specific service, these can be bypassed, replaced or customized.

The component system structure is illustrated in Figure 2.

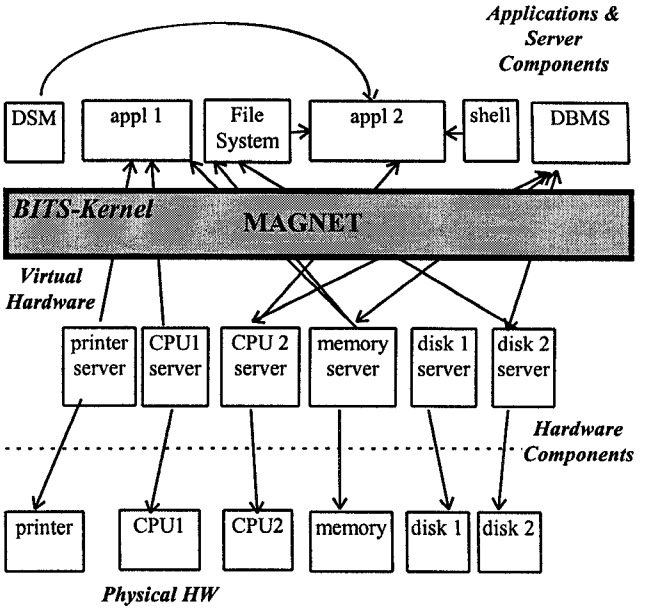

Figure 2. Component Based OS Structure

3.2. The MAGNET Resource Manager

MAGNET has the following goals:

- to satisfy resource requirements for the widest range of applications
- to support flexibility, extensibility (tailoring policies to given hardware configurations and to specific application requirements) and reconfigurability
- to utilize hardware resources efficiently.

The following list summarizes the design approaches used in MAGNET:

- separation of kernel services from resource management policies
- separation of resource protection from resource management policies
- encouraging applications to take part in the resource allocation negotiation process
- enabling dynamic modification of services at run-time (hot swap, plug and play)

- enabling a run-time replacement of failed components by equivalent ones with application agreement, and
- using late-binding to allow applications to form their requests without previous knowledge of current system service characteristics, configuration, or availability.

Physical distribution of MAGNET Virtual Shared Tuplespace improves reliability and ensures faster access. Although the original idea came from the Linda based tuplespace, MAGNET system does not stick to the Linda based system semantics (as used in a system such as OSPREY [7]) but offers a more flexible tuple matching environment.

3.2.1. MAGNET Structure. MAGNET provides dynamic component binding. It treats all BITS components (Hardware Components, Server Components and applications) equally. Each component acts as a black box providing services for, and requiring services from other components. Services are described as an interface. For component interaction we use Darwin, a well-defined language [13] supporting representation of these structures.

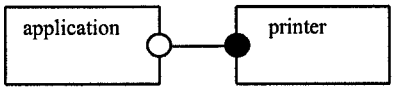

Figure 3. Component binding described in Darwin

In Figure 3, circles represent component interfaces, filled circles represent services provided by a component and empty circles represent services required by other components. Further formal description of component binding - not necessary for MAGNET presentation - is beyond the scope of this paper. It can be found in [13], [14] and [15].

Tuplespace and MAGNET Structure

The MAGNET tuplespace keeps all information on system components (defined by their interfaces) in the form of tuples. A simple example of an application and servers configuration is illustrated in Figure 4.

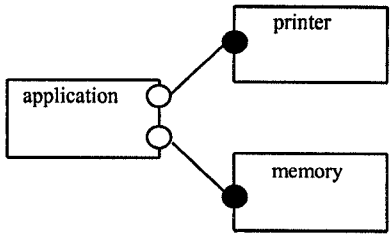

Figure 4. Example of application and servers interaction

As we are not dealing with a general binding situation, we have split the tuplespace into several smaller tuplespaces called **Resource Tuplespaces**, each one dedicated to a particular resource (e.g. printer, disk etc.) This approach enables logical parallelization of the binding process. Figure 5 illustrates corresponding simplified tuples placed into the relevant Resource Tuplespaces.

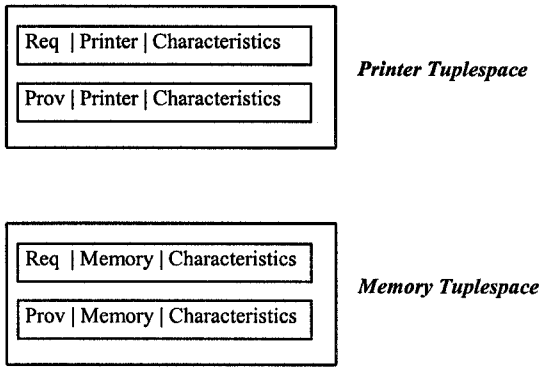

Figure 5. Example of relevant Resource Tuplespaces

Resource Tuplespaces are physically distributed, but to ensure the separation of the physical memory location from the binding algorithm, DSMS presents each Resource Tuplespace as a global namespace located in a virtual shared memory (for further discussion on distributed shared memory see section 3.3.)

3.2.2. Matching. Application demands and component offers, described as their interfaces, are placed into the relevant Resource Tuplespace. At this level, the cost to a binding process is very high but, apart from a few exceptions (component crashes and handling of faults), this matching process takes place only once, before the application starts executing. MAGNET is distributed across the system nodes so that the matching can be

performed in parallel to improve reliability and prevent MAGNET being a system bottleneck.

Once all components are selected and "plugged together", fast indirect calls provide efficient inter-component communication ensuring good overall performance.

For rebinding purposes all changes are stateful. (see the field "State" below).

A general tuple format defining the component interface consists of the following items:

Prov/ Req	ID	State	Time-stamp	Fea-tures	Min. match	Time-out

- **Prov/Req** - indicates if the service is provided or required.
- **ID** - component type ID (e.g. printer, memory etc.)
- **Timestamp** - the time the tuple was inserted in the tuplespace.
- **Features** - provided/required resource characteristics (e.g. amount of memory provided/required, printer type, CPU performance, etc.).
- **Min. match** - Minimum requirements the application can work with. This is used when negotiating for a resource when the ideal is not available (not applicable to server components).
- **Timeout** - The time period ($0 < \text{Timeout} < \infty$) the application is willing to wait for the tuple to be satisfied. If a match cannot be found in this time, the application is notified (not applicable to server components).
- **State** - The resource's state (e.g. available, failed, disconnecting, bound to, etc.).

The tuple formats are designed to provide additional flexibility, to support the negotiation process, and to best utilize available resources.

The matching process is initialized when a tuple is placed in the tuplespace. MAGNET runs its **Matching** algorithm concurrently. If the required binding according to the "Features" description is found, then the communication is established using the Glue. If the required component is unavailable MAGNET can negotiate less optimal tuples, and/or notify the component about non-availability when its "Timeout" expires. The combination of "Timeout" and "Min. match" provides an additional level of flexibility impossible in traditional resource management schemes. A full description of the matching algorithm and negotiating procedure is beyond the scope of this paper.

3.3. Virtual Shared Tuplespace

Communication in a distributed environment can be based on either message passing or virtual shared memory. Implementations of virtual shared memory systems such as in Angel [12] showed better flexibility and performance than message passing systems.

BITS does not hide its physical memory distribution from all applications enabling the DSMS to tailor the implementation to MAGNET tuplespace requirements. This approach supports a separation of the physical memory distribution from the binding algorithm. The alternative solution would be to make the distribution of the tuples visible but this is more complex, causes implementation difficulties and provides no functional advantages [5].

3.3.1. DSM Design. The major decision when designing DSM is to choose the appropriate consistency model, which suits the character of application (although BITS can tailor this to specific applications), otherwise performance may be poor.

Generally consistency models can be divided into two groups [16]:

1. consistency models not using synchronization operations (strict, sequential, causal, processor, PRAM)
2. consistency models with synchronization operations (weak, release, lazy release, entry)

All consistency models not using synchronization operations are too strict for MAGNET's purposes. This is also discussed in [17] which argues, that namespaces usually require only weak consistency model.

The **lazy release consistency** model [19] (which differs from the normal consistency model by delaying the propagation until the next "acquire" requests it) has been chosen for MAGNET because it offers:

1. significant performance advantages over the *weak consistency* model (does not distinguishes if the critical region is entered or left)
2. performance advantages over the *normal (eager) release consistency* model (propagates after every "release" operation even when the same data is to be accessed by the same processor again)
3. programming simplicity over the *entry consistency* model (gives good performance results, but management of the list of synchronization variables is extremely complex). For a large number of components this model will also be considered.

A DSMS runs on each CPU, connected to others to form a single circle enabling easy communication among all DSMS. In our system we have a small number of processors so this arrangement is adequate. For large systems other communication structures could be used.

This DSM enables the sharing of the tuplespace between the physical nodes. However, due to the structure and distribution of MAGNET there is no need to provide uniform shared memory among all Resource Tuplespaces.

In addition to the flexibility provided by the DSMS, MAGNET itself can choose between blocking and nonblocking access to tuples which provides another level of flexibility for waiting applications.

Example

To show the performance advantage of the lazy release model, we start our description at the moment when a component is inserting its tuples into the appropriate Resource Tuplespaces. It then initiates a "release" operation. As yet, nothing is propagated. MAGNET then "acquires" and the DSMS must determine if it has the most recent data. If not, the propagation is performed. MAGNET then matches the tuple with the most appropriate one and selects the suitable Glue to connect those components and changes the "State" field of their tuples. Once these matching and updating operations are performed, a "release" is initialized to resume the application. Again nothing is propagated until the data is "acquired".

Lazy release consistency is efficient not only because it combines more tuple updates into a single operation, but also because it delays the propagation until the updated data is requested. A sequence of operations performed on the same processor cost nothing (assuming no competition).

4. Discussion

The MAGNET system has the potential to provide more than just flexible resource management. Because of its ability to bind components "on-demand" we can provide a framework for dynamic rebinding and crash recovery services such as hot swap and plug and play. Because components act as black boxes, MAGNET cannot be responsible for ensuring consistency. In the case of application initiated rebind (not forced by a crash), the rebinding is not performed until both components involved are in a safe state to do so. MAGNET does not perform rebinding transparently.

When a device crash is detected, it is reported to MAGNET which determines whether the device needs to be restarted or if it must be replaced. In the former case the application will only be delayed, but in the latter a new device must be found to replace the failed one. To do this the application tuple has its "State" changed and a new negotiation is initiated. The resulting component is then "swapped" for the faulty one in the application. The application can then be restarted. MAGNET supports a hot swap and easy recovery environment with post crash application participation strategies. We believe we have the ideal platform to support such a facility, especially in distributed systems designed for 24x7 operation.

5. Problems

Not all the problems with the MAGNET system have been resolved. One problem still to be addressed is one of trust. MAGNET has to cope with problems of untrustworthy tuples in the tuplespaces, either by restricting who can insert, modify and delete tuples or by providing a mechanism to check the reliability of the tuple information. Without this, it would be possible for a malicious application to place misleading tuples in the tuplespace.

An enhancement would be to add a priority scheme to the tuples to provide extra flexibility to applications and components.

Another problem that needs resolving is that of resource revocation, which requires MAGNET and the BITS Virtual Hardware layer to cooperate. Our current approach is that MAGNET allocates the resource and "glues" the application to the server. BITS Virtual Hardware layer is responsible for preventing starvation, taking a resource back from an application which ignores the revocation request by force.

Another potential problem is MAGNET's performance for a system with a large number of components. For a small number of components, the implementation of the global Resource Tuplespaces is suitable, presenting the best component available at the time with satisfactory performance. For large systems, a more complex system of tuplespaces may be required. This will be addressed in our future research.

6. Related Work

The construction of systems using components, and the use of flexible communications mechanisms are approaches that are being examined by much contemporary work.

A lot of work has been done in the area of dynamic component binding formalism in distributed systems. As discussed, Darwin [13] together with Regis [11] give an

excellent environment for the dynamic building of distributed systems.

Standard communications schemes such as message passing and the popular remote procedure calls (RPC) provide a method of binding components, but are too inflexible, and do not support parallelism. Interface Definition Language (IDL) is a standard solution to the binding problem and is implemented in various systems, including CORBA [10]. It describes the communication interfaces using a standard language from which the relevant interface code is generated.

The alternative solution is to use a tuplespace scheme such as that found in Linda. Linda was the first system to support a generative communication model [8], but its fixed tuple format and semantics do not provide the flexibility we require. An alternative system called OSPREY [7], a question based system, uses the Linda approach applied to application-server level coupling. It adds a level of flexibility by a result-based tuple naming scheme and by replicating tuples to many nodes.

24x7 operation support for dynamic data object communication has been explored in "The Information Bus Architecture" [9] which is based on principles such as self-describing, anonymous communications and minimal semantic communication protocols.

Dynamic binding and rebinding have also been explored in Kea [18], based on RPC generated entirely by the Kea kernel. It allows the destination of the RPC to be changed independently of the caller.

7. Conclusions

In this paper we have presented a flexible, adaptable resource manager called MAGNET designed for use with the Component Base Operating System BITS. MAGNET is based around a Virtual Shared Tuplespace using a lazy release consistency model for implementing DSM. MAGNET administers the tuplespace having a goal of finding the best available application-server match. This satisfies the demands of a wide range of applications and provides good resource utilization in a flexible and dynamic way. In addition, the presented MAGNET design shows how the system provides support for dynamic component selection, reconfiguration and exchange at run-time, such as plug and play and hot swap, in order to support 24x7 operation.

MAGNET is an essential component specially designed for the BITS operating system. BITS' component based design ensures good performance by moving abstraction closer to the hardware level and by binding trusted applications to servers through indirect calls. Its component structure and late binding ensures flexibility and a finer granularity interface to system services. BITS separates kernel services and protection from resource policies providing additional flexibility and extensibility.

MAGNET is being implemented as a part of the BITS project. The BITS environment together with MAGNET offer a radically new approach to achieve good performance, increased flexibility, extensibility of system services and dynamic resource allocation fulfilling the high demands of current applications.

Acknowledgments

We thank Kevin Murray for his insight in the subject area and contributions to the BITS project. We thank Steve Crane for his help on formalizing our approach using the Darwin language and for many stimulating discussions. We also thank Nick Plumb and Andy Whitcroft for helpful suggestions and careful reading of previous versions of this paper.

References

[1] D.R. Engler, M.F. Kaashoek, J.W. O'Toole Jr. *Exokernel: An Operating System Architecture for Application-Level Resource Management.* MIT Laboratory for Computer Science. Proceedings of the 15th ACM SOSP, Colorado, USA, December 1995.

[2] S. Savage and B.N. Bershad. *Some Issues in the Design of an Extensible Operating System.* Department of Computer Science and Engineering, University of Washington, Seattle, USA. 1st Symposium on Operating systems Design and Implementation, Monterey, California, USA, November 1994.

[3] W.H. Cheung, A.H.S. Loong. *Exploring Issues of Operating systems Structuring: from Microkernel to Extensible Systems.* Department of Computer Science, The University of Hong Kong. Operating Systems Review, Vol. 29, No. 4, October 1995

[4] P. Kostkova, K. Murray, T. Wilkinson. *Component Based Operating System.* SARC, City University, London. 2nd Symposium on Operating systems Design and Implementation, October 1996.

[5] P. Ciancarini, N. Guerrini *Linda meets Minix.* University of Bologna and University of Pisa. Operating Systems Review, Vol.27, No.4, October 1993.

[6] A. Messer, T. Wilkinson. *Components for Operating system Design.* SARC, City University, London. 5th IWOOOS '96, October 1996.

[7] D. Bolton, D. Gilbert, K. Murray, P. Osmon, A. Whitcroft, T. Wilkinson, N. Williams. *A Question*

based approach to Open systems: OSPREY. SARC, City University, London. March 199

[8] N. Carriero, D. Gelernter. *Linda in Context.* communication of the ACM 32, 4, April 1989.

[9] B. Oki, M. Pfluegl, A. Siegel, D. Skeen. *The Information Bus.* Teknekron Software Systems, Inc, Palo Alto, California. SIGOPS '93, December 1993.

[10] *The Common Object Request Broker: Architecture and Specification.* Object Management Group. Doc. No. 91.12.1

[11] J. Magee, J. Kramer, M. Sloman. *REGIS: A Constructive Development Environment for Distributed Programs.* Distributed Systems Engineering 1(5), 1994.

[12] System Architecture Research Centre. *Experiences with Distributed Shared Memory.* SARC, Department of Computer Science, City University, London, UK, 1993.

[13] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. *Specifying Distributed Software Architectures.* Fifth European Software Engineering Conference, Barcelona, September 1995.

[14] S. Crane. *Dynamic Binding for Distributed Systems.* PhD Thesis, Imperial College, University of London, March 1997.

[15] S. Crane. *A Framework for Distributed Interaction.* International workshop on development and Evolution of Software Architectures for Product Families, Madrid, November 1996.

[16] A.S. Tanenbaum. *Distributed Operating Systems.* Prentice Hall, Inc., 1995

[17] S. Mullender. *Distributed Systems.* Addison-Wesley, 1993.

[18] A.C. Veitch, N.C. Hutchinson. *Kea - A Dynamically Extensible and Configurable Operating System Kernel.* Third International Conference on Configurable Distributed Systems, Annapolis, Maryland, USA, May 1996.

[19] P. Keheler, A.L. Cox, W. Zwaenepoel. *Lazy Release Consistency.* 19th Annual International Symposium on Computer Architecture, ACM, 1992