A MICROKERNEL ARCHITECTURE FOR DISTRIBUTED MOBILE ENVIRONMENTS

Thomas Bopp, Thorsten Hampel
Heinz Nixdorf Institute,University of Paderborn, Fuerstenallee11, Paderborn, Germany

Keywords: Microkernel, Distributed Objects, Peer-to-Peer, CSCL.

Abstract: Microkernels are well known in the area of operating systems research. In this paper we adapted the concept of microkernel to the field of Computer Supported Cooperative Work and Learning (CSCW/L) to provide a basic underlying architecture for various collaborative systems. Such architecture serves well for the fields of mobile and distributed collaborative infrastructures with its new inclusion of small mobile devices and ad-hoc network structures. Our architecture provides a distributed object repository for an overlay network of CSCW/L peers. Nodes can dynamically join and leave this network and each peer is still autonomous. In this network different kinds of peers exist depending on the module configuration of a system. So-called super-peers with lots of storage and computing power provide gateways to the network (for example HTTP).

1 INTRODUCTION

While the World Wide Web is still the standard way of accessing documents in the Internet, various new applications are created for communication and collaboration. In particular, the emergence of Peer-to-Peer systems with the pioneers of Napster and Gnutella reduced the gap of consumers and producers. Everyone can bring their own content into the peer-to-peer network and share it with other users. Instead of the classic client/server infrastructure we now have a collaboration of equal partners.

Furthermore the Internet has grown from machines with a permanent connection at the office to users at home connecting temporarily to the Internet. The latest developments are wireless networks where users can connect with notebooks and personal digital assistant (PDA). Instead of a communication to a central server there is also the possibility to communicate with another device in close proximity.

As another development more and more smaller mobile devices come into the focus of the users. Mobile devices lack resources such as memory and computing power required for existing centralized architectures.

In this paper we present a microkernel architecture to support different kinds of environments. On one hand there are static environments with machines with good processing power and in general a lot of resources to be used by an application. On the other hand there are also networks without a permanent connection to the Internet and there are nodes joining and leaving that network dynamically. Each of these nodes might have varying processing power and storage capacity.

As a common underlying architecture for applications on different devices the microkernel at first offers functionality to load, exchange, and remove modules. Appropriate modules will provide all other necessary functionalities.

The core components of our architecture support interaction, offer persistence of objects and provide security. Using protocols like JXTA the different nodes build an overlay network where different peers are in contact with each other. Each node of this network can have a different configuration though and we can roughly distinguish between two types of peers:

- Devices on users' site connecting to the network
- Server machines with huge storage and processing capacity

To make all devices in the network work together it is necessary to meet the requirements of

Bopp T. and Hampel T. (2005).
A MICROKERNEL ARCHITECTURE FOR DISTRIBUTED MOBILE ENVIRONMENTS.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 151-156
DOI: 10.5220/0002545101510156

the ones with limited resources. The microkernel allows different module configurations and therefore the same software is able to run on the different devices.

The main reasons for microkernel architecture instead of a monolithic system can be identified as:

- Use of resources: Mobile devices have less processing power and disk capacity than large server systems.
- Extensibility: The modular architecture allows flexibility in adding new modules and replacing existing ones.
- Maintainability: The components (modules) of the system can be maintained separately. Each server administrator can maintain his own server.
- Configurability: On different platforms the system can be configured with different modules.
- Alternative communication infrastructure and mobility: The architecture does not rely on a central communication infrastructure. In particular, ad hoc communication as part of mobile scenarios is now possible.
- Security: The module concept allows different security modules to run on the same system and interact with each other. Each server can set its security policies.
- Platform independence: The system is independent from the operating system. This means the same software can be used in different environments.

Our recent conceptual approach in the field of Computer Supported Cooperative Work and Learning (CSCW/L) has been to build cooperative knowledge spaces. As one implementation the result is a monolithic client/server system called sTeam (Hampel and Bopp 2003). The idea of this system is to combine document management facilities with a room metaphor in collaborative knowledge spaces.

Although in our system there is some basic support for modules, it fulfils part of the criteria above and will not run on small devices like PDA. Each sTeam server uses a local database and is not able to use different persistence layers.

Apart from that it is our goal to connect rooms of different servers. Therefore our first solution was an enhancement of our monolithic server architecture by creating a new type of inter-server object called “shadow” (Bopp et al, 2004). Unfortunately the old server architecture needs a fundamental redesign to be able to transparently connect these rooms.

Due to the shortcomings of a monolithic system we developed a new underlying architecture. This architecture has been designed to be as simple as possible in order to make it understandable for a larger Open-Source development Community.

This paper will outline the main characteristics of this Microkernel-based architecture for both mobile peer-to-peer scenarios and multi-server approaches.

First we briefly present some related work. The Microkernel itself is described in section 3. At the end of the paper we present some important modules and how a distribution of object can be achieved.

2 RELATED WORK

A kernel is the essential part of a system. It is responsible for allocating resources, security and other crucial tasks.

A microkernel (Rashid et al, 1989) is a kernel that consists of the minimum required functions to run a system. All additional functionality is moved into modules, which are dynamically loaded by the kernel.

The concept of microkernel is common in the area of operating systems. The idea of this work is to adapt this concept to the area of computer supported cooperative learning and working.

The JADE project (Oliveira et al, 1999) has taken a quite similar approach for the area of graphical virtual environments. The Java-Implementation guarantees platform independency and makes it possible to run the software on different devices. The microkernel manages modules, which are executed in the context of their creators. Therefore each module can has different permissions.

Another important concept of the JADE microkernel is the communication design. It provides three possibilities:

- Passive Communication: Objects provide static methods for invoking the functionality.
- Local Events: Objects are subscribing to events inside modules and modules are also able to subscribe events inside any object or other modules.
- Central Events: There are central instances keeping track about any event. It is possible for modules to subscribe to central events and therefore be notified about everything.

Maverik (Hubbold et al, 1999) is a GNU Project similar to JADE, but is implemented using the C language. It is a platform for large-scale applications of virtual reality. Even though the system runs on different operating systems, there is no general platform independence. Another microkernel architecture for virtual environment is Bamboo (Watsen and Zyda, 1998), which is written in the C++ language.

All of the microkernels above are designed for graphical virtual environments. Therefore the kernel includes a lot of predefined functionality for this context. Due to that they are not working on mobile devices like PDA.

Also the MadKit agent infrastructure (Gutknecht et al, 2001) is based on a microkernel approach. It offers an underlying agent platform, which is able to integrate different agent architectures. At its core it offers some agent-specific functionality like the control of local groups and roles and agent-lifecycle-management. It launches agents and assigns them global unique identifiers. Apart from that the message passing between local agents is handled by the microkernel.

OceanStore (Kubiatowicz, 2000) is a peer-to-peer cooperative file system. It is object based and objects are replicated. This replicas are called floating replicas because they are independent from a server and might be moved from one server to another. The system creates an overlay network where users can access the content of all the nodes in the network.

Even though Eclipse is known as a JAVA development tool, it is also a microkernel-based architecture. It provides a plug-in management, which resolves dependencies between different plug-ins. The system is not suited for a CSCW-microkernel, because it loads some core modules by default. Those are responsible for the user-interface, which is not needed by all peers in a peer-to-peer network. Also it is too resource hungry to run on mobile devices like PDA, because it is a Java-AWT interface. Apart from that the eclipse system includes a lot of important concepts, which have been integrated into our microkernel architecture.

3 MICROKERNEL ARCHITECTURE

The microkernel is the core component of our room-based CSCW/L-System. Its functionality is minimized to load and configure modules. As the second main function it provides the object class as the core class of an object-oriented system. Apart from that there is basic network support to open ports and connections with different protocols to other hosts.

Figure 1 shows the kernel with different modules. Beside the module manager there is persistence and network support. The communication between the kernel and the modules is handled by events (a detailed description of the event system can be found in the next section).

The resulting application is defined by the set of modules configured. The minimum configuration

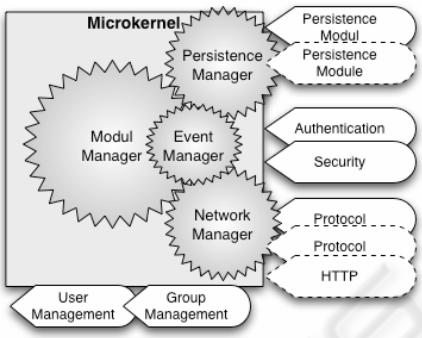

Figure 1: Microkernel

consists of a persistence module, at least one protocol and a security module.

When loading the modules of the application, it is important to determine an order for the load process, because of dependency. The Eclipse Framework offers a comprehensible plug-in mechanism: modules can require each other or extend others. This model has been adapted for our architecture. Each module has its own configuration file to define dependencies between different modules. For example there might be a file persistence module for storing objects in the local file system. Furthermore there could be a persistence module, which provides a database for storing the objects. Both modules would extend the basic persistence layer.

The persistence manager of the microkernel is able to deal with several persistence modules. Those modules can use a database or other locations to store objects. The system is also able to provide objects for external clients using protocols like JXTA. The result is a remote persistence module that works on the resources of another node. In this sense the other node provides a repository for objects. So there is a remote persistence module on the one side and a repository on the other. The repository is just another view on the objects of that node. This is called a network view and is derived from the basic view module. In a classical sense, a view represents a graphical display of the objects (see Krasner and Pope 1988).

The overlay network is created using this functionality. Each node can have its own repository of objects and provide them to other nodes, which access objects by a special kind of persistence layer. This remote persistence works in the same way as any other persistence module, but needs to keep connections to other nodes of the network. If every object is accessible within the network the nodes build a shared space. Any node can access any

object (of course permissions are checked) in the network and due to that it is possible to transparently move through the shared object space.

Figure 2 illustrates the idea of the shared object space with some peers and one super-peer (Mizrak et al, 2003), which contain more storage space and some additional protocols. Due to that it also functions as a gateway and provides access to the whole shared object space to external users (for example using a Browser to access objects).

Due to the different module configurations of the nodes, each node can provide varying functionality. In general we distinguish between two types of nodes:

- Static Nodes: typically on machine with static IP; they provide lots of functionality and consist of many objects since there have large capacity; they are similar to classic servers.
- Dynamic Nodes: with changing Internet address; they provide varying functionality depending on the device and application; the minimal functionality is to work as an object broker and provide communication with other nodes.

Although in a peer-to-peer network all peers are equal and should have the same functionality, the two classes above are acceptable. The first class of nodes offers the same functionality as the second class, but in addition provides more capabilities. Those nodes can be called super-peers. They offer a gateway into the network with different protocols. For example a device with limited resources, like a PDA, might not be able to offer a HTTP-Server. Super-peers provide this kind of functionality. Apart from that they have a static IP address and allow other nodes to build their initial connection to the network.

In order to make the microkernel run on all different devices, it needs to be platform independent. This can be accomplished by using JAVA (or any other platform independent language) as the programming language. With the possibility of different module configurations the kernel is scalable to use different resources and thus works on devices with small resources. On a PDA the persistence layer could use a memory card and provide basic functionality for storing and retrieving objects.

The speed of the JAVA language compared to compiler languages like C or C++ is not such an important issue, because the design of the network with sharing of processing power and storage capacity of the different nodes distributes the load on the nodes. Apart from that, a CSCW/L system uses fewer resources than a graphical virtual environment like Bamboo. The platform independence is much more important in the context of a CSCW-system. Moreover all critical code can

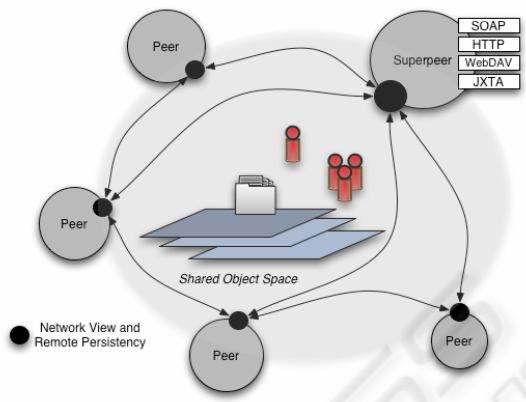

Figure 2: Shared Object Space.

be implemented as a library and imported to the microkernel as a module.

Due to the modular design of the system it is always possible to add new modules and thus enhance the system. On the other hand modules can be exchanged with different ones providing the same interface, but working in a different way.

Good examples for this are security plug-in that might work differently on each peer and define own security policies. Thus any peer in the network is autonomous and can be maintained by its local administrator.

Since the modules are the key components of the microkernel and define the application, the next section will describe the most important module types of our architecture.

4 MODULES

Modules are the key components of the CSCW/L application. Some of them are loaded at system start-up and some functions are called to initialize the modules. Other modules are dynamically added to and removed from to the system as and when necessary.

The communication between the modules takes place through events. The design of our event system is similar to the JADE implementation of events. There are *local events* inside any object in the system and there are also *global events*. For local events the subscriber has to specify the object and the type of event. When the event takes place the subscriber is notified.

The global events work quite similar, but instead of subscribing to a single object the subscriber calls one central event-instance, which keeps track of all events inside the system. The subscriber is notified

about any event-type that takes place for all objects. The subscription of events enables the modules to be notified about all actions within the server.

In a room-based CSCW/L system, the modules that provide persistence; offer communication using different network protocols; and provide security are core modules. Security modules handle permissions for users in the context of their environment (rooms). In particular, the possibility of using different security modules guarantees the autonomy of peers within the network.

The following types of modules can be found in a CSCW/L system based on our architecture:

- Security: Provide a security layer for the application. The security system is bound to the event system. Since each action triggers an event it is possible to block them and thus a security mechanism is provided.
- Persistence: Objects need to be stored in a database and must be retrieved at the point of access. After a reboot of the system all objects must be still present and no data is lost. The types of persistence are usual a database or a file system. A remote persistence module allows the access of objects on different nodes of the network.
- Views: A view displays the objects of the microkernel in a certain way. An application like a whiteboard offers a view of the objects by displaying them in spatial, synchronous view. There are also network views, which provide the objects by some object-oriented protocol.
- Controller: The application logic is part of these modules. They offer various functionalities and combine the components of the application.

Any loaded module is just available once in the server (a singleton). It is not possible to create several instances of a single module. Also unlike the objects the modules are not distributed through the nodes of the network.

5 DISTRIBUTED OBJECTS

Our microkernel-based CSCW/L architecture uses a fully object-oriented design with different classes. Each object in the server can be uniquely identified by its object id (OID). Since this id is only unique for the local persistence layer, the id must be extended by an id for each persistence layer. This namespace id (NID) describes the physical location of an object. When the persistence layer needs to locate an object it has to identify the place where to look for the object. Otherwise it is not possible to uniquely retrieve the object, because the same OID could be used in different persistence modules.

Apart from that for a globally unique identification of an object the server ID (SID) must also be included in the id of the object.

The attempt to generate a unique global identification for each object is common in the area of peer-to-peer systems. For example OceanStore uses a GUID (global unique identifier) (Kubiatowicz et al, 2000). It is composed from the objects name and its owners key.

The objects are distributed through the network and a shared object space is the result of the combination of view and remote persistence layers. The persistence managers retrieve objects from remote repositories and create local replicates of them. For the lifetime of those replicates the state is synchronised using the event system of the microkernel.

Any action of a user triggers events, which are distributed through the views to the remote persistence modules. Those events can be used for notification about modification of objects. The local replicates are updated accordingly.

Figure 3 shows the modification of an object as a result of a user's action. The persistence managers of

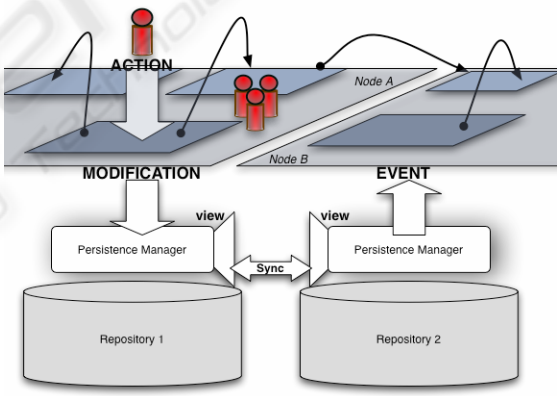

Figure 3: Network Views and Object Synchronisations

two servers communicate with each other using the event mechanism.

Apart from this the events need to be propagated between servers to allow cooperation of users. When synchronous cooperation takes place it is necessary to be notified about all changes within a user's environment.

In this context one important concept is the use of virtual rooms as a meeting place for users. Besides this room-class there are some more classes for all components of a CSCW/L system, e.g. for Documents, Users, Groups and so on. Each of these classes provides the same interface as the object class. Additionally there are some special methods

for each class. For example the class Document provides methods for storing and retrieving content.

6 CONCLUSION

This paper can only focus on some modules of our architecture. As the core architecture is designed around a strong object-oriented approach, all other classes and objects are shaped to the same design rules and architectural foundations as presented.

We have shown that the microkernel is a reasonable concept for a CSCW/L system. With protocols and object repository functionality it builds an additional network layer. It works as an underlying architecture for applications on static servers as well as mobile devices.

Due to the java implementation with modular functionality, it is independent of the platform and works on almost any operating system.

Moreover our microkernel architecture supports students with a very tight code base. All the functionality is moved into modules with well-defined interfaces. Due to this, it allows an open source community to collaboratively extend the overall architecture. E.g. it is possible to work on just a few modules without understanding the rest of it in detail.

Most importantly, the view/remote-persistency pairs in our approach establish a CSCW/L overlay network. All objects of every node in the system are available in the shared object space. Most of them are located at super-peers with huge storage capacity and processing power. Those peers also provide a gateway into the overlay network. Additionally they provide a View for external clients like Browsers and are also capable of accessing all objects of the shared object space.

The shared space is also an active repository of objects. All actions within the server are triggered and events are propagated from one server to another.

Current and future work focuses on the aspects of ad-hoc communication as basis on the above presented architecture. Here problems of replicated and distributed objects in distributed knowledge spaces have to be solved. Objects have to be moved transparently for the user from one to another peer when mobile peers are leaving or joining the network. Concepts of mobile knowledge spaces and persistent knowledge spaces apply. Our microkernel-based architecture proves to offer great flexibility in developing and evaluating these concepts and architectures.

REFERENCES

Bopp, T.; Hampel, T.; Eßmann, B.: Connecting Virtual Spaces. Proceedings of the ICEIS 2004, Sixth International Conference on Enterprise Information Systems, 475–479.

Eclipse Platform Technical Overview. http://www.eclipse.org/whitepapers/eclipse-overview.pdf, 2004.

Gutknecht, O.; Ferber, J.; Michel, F.: Integrating Tools and Infrastructures for Generic Multi-Agent Systems. Proceedings of the fifth international conference on Autonomous agents, 2001, 441–448.

Hampel T.,Bopp T. (2003): Combining Web Based Document Management and Event-Based Systems - Integrating MUDS and MOOS Together with DMS to Form a Cooperative Knowledge Space. ICEIS 2003, Proceedings of the 5th International Conference on Enterprise Information Systems, pages 218-223.

Hubbold, R., Cook, J., Keates, M., Gibson, S., Howard T., Murta, A.,West, A., Pettifer, S.: GNU/MAVERIK: A Micro-Kernel for Large-Scale Virtual Environments. Proceedings of the ACM Symposium on Virtual Reality Software and Technology 1999, 66–73.

Krasner, G.E., Pope, S.T.: A cookbook for using the Model-View-Controller interface paradigm. *Journal of Object-Oriented Programming* 1(3) 1988, 26–49.

Kubiatowicz, J.; Bindel, D.; Chen Y.; Eaton, P.; Geels, D.; Gummadi, R.; Rhea, S.; Weatherspoon, H.; Weimer, W.; Wells, C.; Zhao, B.: OceanStore: An Architecture for Global-scale Persistent Storage. Proceedings of ACM ASPLOS, 2000.

Mizrak, A.; Cheng, Y.; Kumar, V.; Savage, S.: Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. Proceedings of the Third IEEE Workshop on Internet Applications, 2003, 104–111.

Oliveira, M.; Crowcroft, J., Brutzman, D., Slater, M.: Components for distributed virtual environments. Proceedings of the ACM Symposium on Virtual Reality Software and Technology 1999, 176–177.

Rashid, R.; Baron, R.; Forin, A.; Golub, D.; Jones, M., Julin, D.; Orr,, D.; Sanzi, R.: Mach: A foundation for Open Systems. In Proceedings of the 34th Computer Society Ithe Second Workshop on Workstation Operating Systems(WWOS2), September 1989.

Watsen, K.; Zyda, M.: Bamboo - A Portable System for Dynamically Extensible, Real-Time, Networked Virtual Environments. Proceedings of the IEEE Virtual Reality Annual International Symposium 1998.