A FRAMEWORK FOR DISTRIBUTED OBJECTS IN PEER-TO-PEER COOPERATION ENVIRONMENTS

Bernd Eßmann, Thorsten Hampel
Heinz Nixdorf Institute
University of Paderborn
Fürstenallee 11, 31102 Paderborn

Keywords: Mobile Computing, Spontaneous Collaboration, Distributed Knowledge Spaces, CSCW.

Abstract: Mobile forms of cooperative knowledge organization need system architectures that also allow spontaneous (ad hoc) networking and collaboration structures. An essential requirement here, besides establishing the peer-to-peer network, is the design of suitable framework architectures for object distribution. This article presents our approach to developing a basic architecture for distributed systems that support cooperation. This also involves creating interfaces to existing classical CSCW architectures and systems. The novel element of our approach, which is based on a JXTA network, is its use of JXTA services to distribute objects among peers, thus achieving the desired object distribution.

1 INTRODUCTION

Mobility is one of the most important of modern-day developments. Methods, tools and architectures for CSCW systems must take into account mobile, flexible forms of cooperation. Cooperation partners and learners spontaneously form groups, making use of technologies for spontaneous networking. At the same time, it is essential to take advantage of today's powerful mobile devices, such as PDAs, smartphones as well as laptops, to develop mechanisms that support interpersonal collaboration without existing fixed infrastructures. And it is also important to suitably integrate classical CSCW systems into such mobile forms of cooperation, where they are available.

One possible way to provide ubiquitous network environments without the need for centralized administration is to establish so-called *ad hoc networks* (Perkins, 2001). Ad hoc networks appear to be the infrastructure of choice for network support in mobile environments where no guaranteed connections exist. One of the main features of such networks is the ability to establish network structures spontaneously from scratch, though they are not reliable. The dynamic nature of ad hoc networks means that hosts may appear or disappear without warning; even complete networks may be disconnected (*network partitioning*) (Feeney et al., 2001).

Applications running on such networks need to be aware of the loss of connection to counterparts in the network. Strategies for network failures are especially important for systems providing CSCW environments.

Our approach to the challenges of mobile CSCW environments is to provide a basic architecture supporting small highly portable devices in ad hoc networks as well as powerful CSCW servers in existing network infrastructures. Major services such as network communication, a distributed object repository and user interfaces should be integrated in a flexible and modular manner. This will allow extension of the basic architecture in future developments – an important feature since many of the above-mentioned problems are the subject of current research activities.

As a first step toward designing CSCW systems that suitably integrate peer-to-peer and client-server-based architectural concepts, we begin by presenting a basic object distribution and administration mechanism. This allows the dynamic transmission of objects between ad hoc networked peers and the testing of flexible strategies for replication and object distribution. The approach involves flexibly integrating classical client-server-based CSCW architectures as so-called super-peers – depending on their availability.

The paper is organized as follows: First we introduce systems that address some of the requirements for mobile working environments. Then we look at the fundamental problems of object management in mobile CSCW environments. The following two sections present our conceptual approach and give a brief description of our framework's architecture. The paper ends with a brief conclusion and a look at future research prospects.

2 RELATED WORK

Peer-to-peer platforms are flexible depending in terms of the network infrastructure. Most peer-to-peer solutions are designed to distribute data, communication or computing resources over the network. Nevertheless, even when they are called peer-to-peer, many of these platforms still use certain client-server structures for indexing the distributed data, for connection management and other functions. Since they require Internet access, most of these tools do not allow data exchange in a dynamic ad hoc network. (Milojicic et al., 2002) give an overview of existing peer-to-peer systems.

A well-known peer-to-peer system designed for cooperative work is *Groove*[1]. Developed as a peer-to peer CSCW platform, it features distributed persistence. In view of the architecture for synchronizing the distributed workspaces, Groove still requires servers.

For communication in heterogeneous network environments with multiple devices and applications, the Park Labs developed *Speakeasy* (Edwards et al., 2002b). They call their approach *recombinant computing*. This means providing fixed domain-independent interfaces and mobile code. Speakeasy is designed to interconnect appliances such as PDAs and multimedia devices in the environment on a peer-to-peer basis. *Casca* is an application that uses the Speakeasy technology for cooperative work (Edwards et al., 2002a). It allows the detection and selection of potential partners for collaborative work. Users can share documents in shared spaces. These spaces may also contain devices like printers and beamers but they do not provide semantic or object-oriented structures like virtual knowledge spaces.

3 OBJECT DISTRIBUTION

Client-server architectures are no longer useful in dynamic network infrastructures. The dedicated server is a single point of failure. When the server is disconnected from the network, all clients are rendered useless. Figure 1 shows objects stored in a central repository in the network, as is common in client-server architectures. Clients access original objects

[1]http://www.groove.net

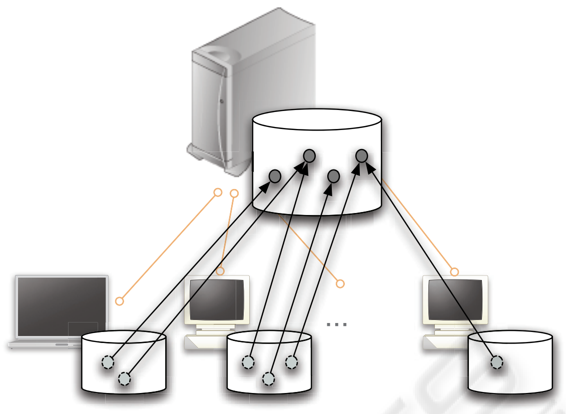

Figure 1: Objects stored in a centralized object repository. Clients reference the original objects by proxy objects.

through *proxy objects*. These are merely pointers to the original objects stored in the central server and its object repository. When disconnected from the server, clients are no longer able to access the objects.

The way to avoid these shortcomings of ad hoc networks is to use peer-to-peer architectures. They are designed to distribute the working environment over all involved devices. This strategy avoids a complete breakdown of the working environment as a result of network disconnections and ensures that resources on all connected devices remain available – the first key to supporting mobility.

There are several strategies for distributing objects over peer-to-peer networks. To get some idea of the strategies available, it might be useful to look at the various stages of a connected distributed object repository:

1. First, each peer is disconnected from the network, only retaining access to its own local repository. The user working on such a peer may only use these local objects (Figure 2a).
2. During connection to the network, the peer will hopefully discover some counterparts, which will themselves provide objects in their local object repositories. Now the peers can also access the remote objects via the peer-to-peer network layer. To enable the remote objects to be manipulated transparently to users, they are represented by so-called proxy objects (Figure 2b).
3. When disconnected from a node hosting a remote object, the proxy objects point to a no longer existing object. In this state, the peer's object structure is inconsistent (Figure 2c).

Most systems providing a distributed object repository try several strategies to deal with disconnection

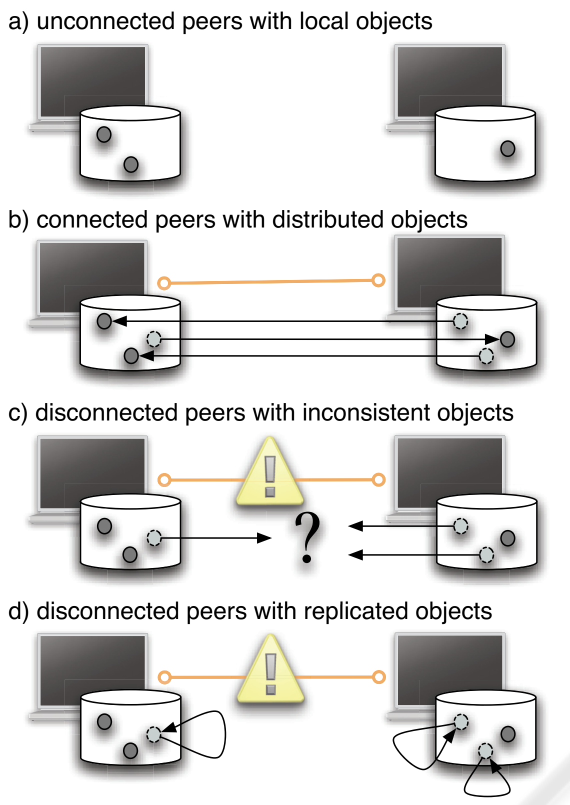
a) unconnected peers with local objects
b) connected peers with distributed objects
c) disconnected peers with inconsistent objects
?
d) disconnected peers with replicated objects

Figure 2: A peer-to-peer-based distributed object repository. Every peer may reference remote objects and provide objects to foreign peers.

from peers hosting remote objects. Often, they replicate the objects over the network. Replication may be done using several strategies. File-sharing applications, for instance, replicate the objects to every peer requesting an object. Other systems use heuristics to distribute the data over a subset of the available peers. To achieve better distribution, it is also possible to distribute chunks of the data, instead of the complete file, over the peers. The heuristics used often include statistics about the online time and storage capacity for selection of the peers to save the objects.

File-sharing systems simply distribute objects without enabling them to be changed once they are stored in the network. In CSCW environments, it is vital to be able to change the objects for cooperation purposes. In the simplest replication strategy, replicated objects can be seen as proxy objects with an embodied copy of the remote object. When connected to the peer hosting the original objects, they point to the remote objects. In the case of disconnection, the replicated objects merely change the pointer to themselves; the user can work on the local replica (see Figure 2d). As soon as they are reconnected, the replicas of an object must be synchronized with the original and the proxy objects point to the originals again. Thus, when working on replicated objects, it is important to synchronize the distributed instances of an object. Concurrent changes on instances could put the distributed object in an inconsistent state.

In our approach, we have not yet tackled the problems of distributed object repositories, preferring to first establish a peer-to-peer communication network to connect peers in heterogeneous networks transparently to the user. The important thing here is that users need not know the network environment they are connected to in order to participate in the cooperation environment.

To establish the network communication, we use JXTA (Gong, 2001), an industry-standard peer-to-peer framework. JXTA allows peer-to-peer communication and service detection even in firewall- or NAT-protected networks. Aiming at an object-oriented, distributed cooperation environment, our implementation transforms the service-oriented philosophy of JXTA into our philosophy of flexible object distribution.

Future versions of our framework will use the flexibility of the framework's design to integrate different strategies of object distribution and replication. At present, however, the objects are stored locally on a single peer, allowing other peers to access them remotely. When disconnected, the peers lose access to the remote objects. To reduce the risk of disconnection from transient peers, we included the option of integrating dedicated CSCW servers into the peer-to-peer cooperation environment. For this purpose, a peer becomes a proxy to the classical client-server CSCW server. Peers can thus access the objects stored on the server, even though the server knows nothing about the peer-to-peer network. Users of peers can store their cooperation results from within the peer-to-peer environment on dedicated CSCW servers. This scenario is shown in Figure 3.

4 OBJECTS, TYPE CLASSES AND VIEWS

The distributed cooperation environment is based on the objects living in the distributed knowledge spaces. Peers building this environment run in a heterogeneous hardware and software environment, which means that many different implementations of the peers may exist. It is not a proper solution to update and restart all peers when new object types are introduced or existing ones are altered.

To tackle this problem, we provide *type classes* carrying the functionality for manipulating and creating objects of a certain type. Each object type is related to

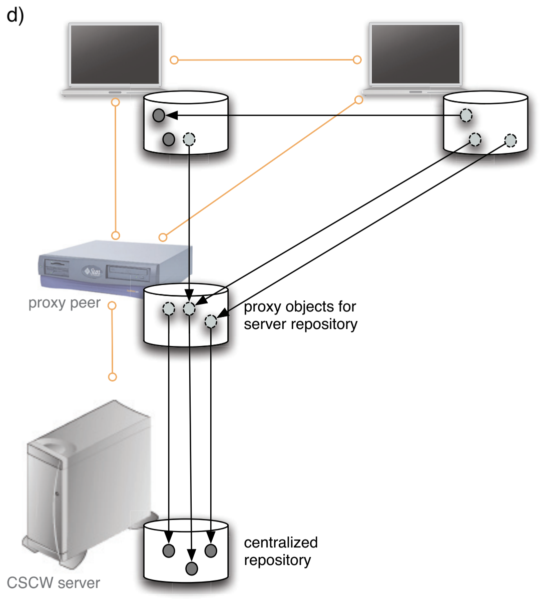

Figure 3: A peer-to-peer-based distributed object repository with a proxy peer acting as a gateway to a dedicated CSCW server.

exactly one type class. To distribute the type classes to the peer, they are stored in the same repository as the objects. New versions of a type class overwrite older ones as soon as they appear in the repository. When a peer does not know how to handle a new type of object, it must simply load the appropriate type class.

The concept for viewing an object's attributes and content is similar. Again, the objects of a specific type are related to so-called *views*. Unlike the type classes, each object type may be related to several views and users can choose the view that best matches their needs.

If participants in the peer-to-peer cooperation environment wish to provide a new object type, they must supply a type-class object, describing the object type's structure and functionality, and at least one view to show its content and attributes within the peer's *graphical user interface (GUI)*. Figure 4 shows an object type with its type class and view.

An object type can be introduced or altered at runtime without restarting any peers in the network. Indeed, views and management code are optional. Although provision of an adapted type class and view is recommended, all objects can be browsed and displayed using default attributes; type-dependent values can be ignored. The additional attributes featured by the type class merely provide further information useful for the associated object type.

An object's standard attributes include a reference list of associated objects. These references to other

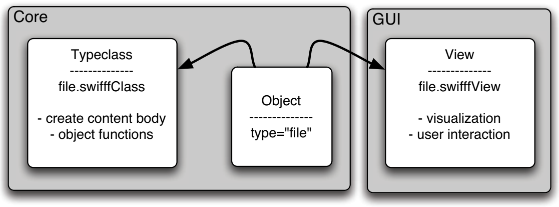

Figure 4: Every object type may provide some views and code for manipulation.

objects cause a graph-like object structure to evolve, which builds the semantics of the distributed knowledge space. The system does not check any constraints to on the references because they are provided by the users' actions. Again, this approach is highly flexible in terms of programming and expanding functionality in the future.

5 DISTRIBUTED KNOWLEDGE SPACES

For seamless cooperation in mobile scenarios, the cooperation environment must be independent of existing infrastructures. This is also true of the network environment and the organizational structures. Important requirements for the cooperation environment are *self-administration* and the option of *unrestricted structuring of knowledge*. For several years now, *cooperative knowledge spaces* have been the main conceptual goal in our efforts to establish collaborative structures between cooperation partners. Virtual knowledge spaces allow cooperation partners to establish groups and associated areas for their cooperation. Different cooperation areas may be connected by gates, allowing the user – represented by a virtual avatar – to move from one area to another. Objects and groups in virtual knowledge spaces are persistent, enabling them to be used in later cooperation sessions. The main idea behind the knowledge space metaphor is that all the different services, documents and objects can be interreferenced in an integrated manner. To transpose this concept to peer-to-peer environments, where objects are distributed over the network, the concept of virtual knowledge spaces must be enhanced. We call this new notion of knowledge spaces spread over the network *distributed knowledge spaces*. The cooperation area is saved in its entirety on a device and can be replicated by collaborators. Gates are marked as nonfunctional if the target area is not available.

For short-term cooperation, we have presented the concept of *temporary groups and knowledge areas* (Eßmann and Hampel, 2003). They are particularly useful in spontaneous face-to-face situations where fast establishment of a cooperation setup is needed. Once the cooperation session is closed, the group and the area are deleted. Optionally, they may be included in persistent cooperation structures.

While the above concepts allow seamless cooperation in mobile environments without functional constraints, systems implementing these concepts must take into account different network structures. Basic requirements for an application implementing a mobility-supporting cooperation system are *open architecture*, *automatic configuration* and *spontaneous networking* (Eßmann et al., 2004a). The first step was to implement a basic framework with simple but powerful modules to support mobile cooperation. This architecture is described in the next section.

6 FRAMEWORK ARCHITECTURE

The objective of creating a mobility-aware cooperation platform thus involves implementing the concept of distributed knowledge spaces. The technology is based on a peer-to-peer architecture supporting multiple types of static and mobile devices. The application must, then, be platform-independent and use standardized interfaces and protocols.

To achieve maximum flexibility in terms of providing different device-specific user interfaces, our prototype implementation is split into a core and a user interface part. The communication interfaces are implemented as modules, which can be extended or exchanged. The graphical user interfaces visualize the objects in the knowledge space maintained by the core. While each core instance represents exactly one user of the cooperation environment, it is possible to connect more than one user interface to the core. Additionally, it is possible to run a core instance without a user interface. From the user interface point of view, it may run on a mobile device without a local core instance by using a remote one. Figure 5 shows a possible user scenario, where each user interface is connected to exactly one core, but using several or no connected user interfaces.

While the core is responsible for persistence and object management, the modules are mainly responsible for the network communication. To ensure adaptability to future changes, the design of the core is similar to that of microkernel architectures, allowing individual parts to be exchanged if needed. For a detailed description of the core's implementation in our prototype, please refer to (Eßmann et al., 2004b). Further information on the user interface concepts and implementation can be found in (Slawik et al., 2004).

We now go on to present architectural parts vital

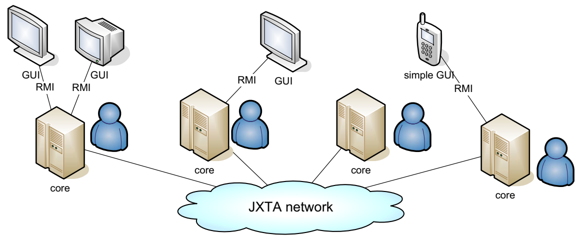

Figure 5: Peer-to-peer-connected cores, each assigned to exactly one user. Optionally, the core can be used by one or more user interfaces.

to our framework. Given the flexible concept of *objects, type classes and views* presented in Section 4, new object types can be added to the system without affecting the core's implementation. An important element in this concept is a flexible *object loader*, including a *persistence module*. While the object loader manages all objects handled by the application, the persistence module manages access to the local data repository. The job of the loader/persistence module is to manage the remote/local storage of objects, respectively. Here, a replication scheme would plug in. Local objects are accessed via a *repository controller* from the local repository; remote objects are accessed via the *network interface module* from the remote peer, which hosts the respective object. This process is transparent to the peer itself: it does not have to take into consideration where the objects are located.

An *access control module* checks the users' permissions concerning a requested object. For this purpose, objects provide an *access control list (ACL)*. The access rights are checked on every access to an object.

External communication is provided via the *GUI interface* and *network interface* modules. The GUI interface provides access for one or more user interfaces, while the network interface is responsible for all network communication between the peers. In our prototype, this is an interface to the JXTA network but it may be replaced or supplemented by a different communication layer in the future. We use the network layer abstraction of JXTA to obtain adequate peer-to-peer communication. A *discovery manager* is closely coupled with the JXTA interface and handles service discovery. If new potential cooperation partners appear, they are instantly made known in the knowledge space by the creation of a new user object. All known users are monitored according to their online status.

Internal object communication is based on XML. Thus, a *parser module* provides XSLT as a query language to receive and filter objects. This feature is used for minimizing communication overhead. The user interfaces only receive the object properties they really need to present to the user. On the assumption that data are more often read than written, this form of optimization is only done for read access.

These central modules enable the whole functionality of the framework to be maintained. While the *object loader*, *persistence module* and *access control* handle internal object access, the *(JXTA) network interface* and the *GUI interface* allow transparent communication with external parts of the peer's core.

7 CONCLUSION

Given the wide range of relevant questions pertaining to mobile (e.g. peer-to-peer-based) forms of cooperative work, the first requirement here is the creation of an open and flexible framework that allows us both to study different research issues relating to object distribution and replication strategies and to develop and test practicable systems in conjunction with existing classical CSCW systems. Another important aspect is the separation of user interface and system core in a network-transparent manner. Devices that are unable to compute the whole core's functionality can utilize other network-connected devices to provide the required core. The system is thus scalable from low-end devices like smart-phones to powerful devices like high-end laptops or even workstations and servers. The flexible object design includes – but is not limited to – user interface design. Our concept allows developers to provide new object types with related views and functionality at runtime. A node that does not know how to handle an object type can load the views and functionality from other peers. By using Java, XML and JXTA, we support a wide range of devices and operating systems. This also ensures that our solution is based on open interfaces and standards. Of course, later versions of our system may exchange or extend the communication protocols because of our open module architecture. Although it fails to provide a comprehensive functionality, our approach offers a mobility-supporting cooperative environment based on open standards and open interfaces. Our framework is therefore a first step toward the development of an open peer-to-peer architecture for distributed knowledge spaces. With its object distribution mechanisms and support for ad hoc networks, it allows research into new forms of mobility in cooperative knowledge management.

ACKNOWLEDGMENTS

Bernd Eßmann is a member of the postgraduate program 776 ”Automatic Configuration in Open Systems”, which is funded by the German Research Foundation (DFG) and the Heinz Nixdorf Institute.

REFERENCES

Edwards, W. K., Newman, M. W., Sedivy, J. Z., Smith, T. F., Balfanz, D., Smetters, D. K., Wong, H. C., and Izadi, S. (2002a). Using speakeasy for ad hoc peer-to-peer collaboration. In *conference on Computer Supported Cooperative Work (CSCW'02)*, pages 256–265, New Orlando, Louisiana, USA. ACM Press, New York, NY, USA.

Edwards, W. K., Newman, M. W., Sedivy, J. Z., Smith, T. F., and Izadi, S. (2002b). Challenge: Recombinant computing and the speakeasy approach. In *International Conference on Mobile Computing and Networking (MOBICOM'02)*, pages 279–286, Atlanta, Georgia, USA. ACM Press, New York, NY, USA.

Eßmann, B. and Hampel, T. (2003). Integrating cooperative knowledge spaces into mobile environments. In Rossett, A., editor, *E-Learn 2003*, pages 225–232, Phoenix, AZ, USA.

Eßmann, B., Hampel, T., and Bopp, T. (2004a). A network component architecture for collaboration in mobile settings. In Seruca, I., Filipe, J., Hammoudi, S., and Cordeiro, J., editors, *6th International Conference on Enterprise Information Systems 2004 (ICEIS'04)*, volume 4, pages 337–343, Porto, Portugal.

Eßmann, B., Hampel, T., and Slawik, J. (2004b). A jxta-based framework for mobile cooperation in distributed knowledge spaces. In Karagianis, D. and Reimer, U., editors, *5th International Conference on Practical Aspects of Knowledge Management (PAKM'04)*, Lecture Notes in Artificial Intelligence, pages 11–22, Vienna, Austria. Springer Verlag.

Feeney, L. M., Ahlgren, B., and Westerlund, A. (2001). Spontaneous networking: An application-oriented approach to ad hoc networking. *IEEE Communications Magazin*, 39:176–181.

Gong, L. (2001). Jxta: A network programming environment. *IEEE Internet Computing*, 5:88–95.

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. (2002). Peer-to-peer computing. Technical report, HP Labs.

Perkins, C. E., editor (2001). *Ad Hoc Networking*. Addison Wesley, Boston, USA.

Slawik, J., Eßmann, B., and Hampel, T. (2004). Shared views on mobile knowledge – a concept of a graphical user interface. In Karagianis, D. and Reimer, U., editors, *5th International Conference on Practical Aspects of Knowledge Management (PAKM'04)*, Lecture Notes in Artificial Intelligence, pages 82–93, Vienna, Austria. Springer Verlag.