CONNECTING VIRTUAL SPACES

Shadow Objects as key elements for weaving the cooperative space

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

Keywords: CSCW, MUD, virtual environment, distributed knowledge, area, multi-server, peer-to-peer

Abstract: Cooperative knowledge areas are a well-proved approach to support process of cooperative work and E-Learning. The Paderborn Open-Source system sTeam establishes cooperative knowledge spaces so far as a single server implementation. This paper presents our architecture of distributed cooperative knowledge areas. The main conceptual idea of the sTeam system is a combination of a document management system and a MUD. The goal of a distributed architecture of cooperative knowledge spaces is the ability to create one world of connected virtual knowledge spaces over various servers. Something, which is especially important, when thinking of new scenarios of integrating peer-to-peer clients into a multi-server architecture. Distributed knowledge spaces also have to cover concepts for a multi-server group and user management, which allow to move users transparently from one server to another. Materials should be structured independently of their location on a server. The following paper first discusses the idea of structuring a virtual world into zones or areas, which is also found in multi-user virtual environments. After that our architecture of distributed cooperative knowledge areas is presented. In the field of user management two different approaches of a peer-to-peer and master-server group and user-management are possible and discussed in detail. Our trial implementation will be a fusion of both concepts and prototypes.

1 INTRODUCTION

The WWW is currently the standard way of accessing documents on the Internet. However, it does not support any cooperative structuring or working with documents. Due to this there are some isolated applications like CSCW or CSCL systems, but there are mostly no standard interfaces to access the materials.

When there are multiple servers the situation often arises where a user wants to reference materials on a different server. On web servers this can be accomplished by using the URL of a document. While this is also possible for objects in a CSCW system, we might want to reference distant objects transparently and work cooperatively with users on other servers. Our goal is to create a distributed cooperative knowledge space (Hampel, 2003) by joining individual servers.

Since our current sTeam system is area based and avatars move through gates from one knowledge area to another, this idea should be extended to create gates between servers. Thus users can access materials on other servers and they do not even have to know about their location.

The following issues are addressed in this paper as key features of a distributed knowledge space:

- Consistent group- and user-management over a cluster of servers.
- Distribution of areas on a cluster of servers.

The most important part is the partitioning of areas. Users work cooperatively in groups inside group's work areas and can move between them through gates. Figure 1 shows a number of areas on a single server. One area is the private workplace of a user named "Carsten", which is connected to the group "Documentation" work area by a gate.
When there are lots of materials inside an area the content can be further structured using containers (folders).

Bopp T., Hampel T. and Eßmann B. (2004).
CONNECTING VIRTUAL SPACES - Shadow Objects as key elements for weaving the cooperative space.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 475-479
DOI: 10.5220/0002610504750479

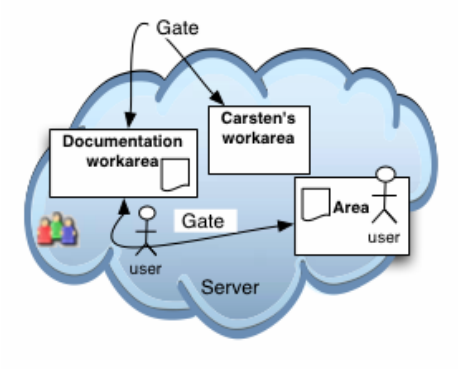

Figure 1: Knowledge Area

sTeam currently uses a client-server architecture where all clients are connected to one central server. Any action is executed inside this server and then distributed among clients using an event system. This mixture of document repository and synchronous communication with the integration of the latest standards and protocols has proven to be a sound approach to distributed knowledge management (Hampel, 2003). The underlying architecture is strictly object-oriented with every class derived from the class "Object" providing the basic functionality of any object. This includes access, events and attributes. Apart from that any object is persistently stored inside a database.

The open source sTeam server is designed as a rather small kernel with pluggable modules. However, the components are all written with a single monolithic server in mind and thus some parts have to be re-designed in order to fit the requirements of a distributed server system.

In order to find a flexible design, we first have to inspect all requirements of a distributed server architecture.

2 MULTI SERVER ISSUES

There are several reasons to create a Multi-Server cluster. As described above our main reason is to have a single world of knowledge areas with all users working cooperatively together through server boundaries. A more common reason in virtual environments is to improve the performance of the overall system. Thus the initial situation might be there is one server that should be split on different machines. Each machine could have a subset of objects of the initial server or each server could replicate the whole content. The latter case would require some sort of master or proxy server to balance the load between all available machines. On the other hand, if each server in the cluster contains native objects then a user would connect to that server where the requested resource is available. Of course the server cluster should be hidden to the user in order to be able to transparently access materials. The research about distribution of virtual environments mostly regards graphical 3D systems.

NPSNET (Zyda et al, 1992) and Massive (Greenhalgh et al, 2000) use Area of Interest Management because of performance reasons – in 3D environments something like the line of sight plays an important part and a lot of events take place in such a virtual world. Therefore events are filtered and only the relevant notifications are send to a client.

In our approach the performance reasons do not play an important role, nevertheless the semantic area concept can be used to improve performance, as well as getting informed about what is going on inside an area. Furthermore it allows distributing different areas on different servers. In this sense the sTeam server is already partitioned and might be distributed to multiple servers. The problem is how to deal with references to objects, which are on different servers and with groups and users, which exist on any server of the cluster.

When it comes to synchronous tools like chat and whiteboard we only have 2D environments, which do not demand full processing power. All interaction takes place on a central server, which notifies other applications about events. This means a lot of network traffic and the requirement of notifying a client only about events that are of interest.

Events can be seen as synchronisation objects that notify other clients, or objects in general, that something has happened. Any object can respond to such a notification in its own matter. A solid event model is required for a middleware and distributed objects (Lewandowski, 1998).

On a CSCW system it should be possible to reference materials on different servers. Supposing a scenario where several departments at a university host their own server and where many relationships between departments and the provided materials exist. So instead of performance scalability we need some kind of semantic scalability on a CSCW server, which means we need one world of cooperative knowledge spaces with a coherent group structure valid on all servers. Due to that a user just has to create a single account and is able to access all the materials provided within the cluster.

Taking into account all the issues mentioned above the server design for a cluster of CSCW systems needs to be very flexible and adaptable on different situations. In the next chapter some different approaches are described.

3 STEAM MULTI-SERVER ARCHITECTURE

The most central part of the architecture is the so-called shadow (see figure 2), which makes possible to reference objects on distant sTeam servers. To do so the shadow only keeps the object ID and the server ID. Apart from that it only functions as a wrapper object to transparently call methods on distant servers. The shadow works effectively as a middleware – a replicated object on the local server with minimal data. All function calls are handled by the COAL (Client Object Access Layer) protocol, which invokes remote methods. This is in a sense related to the proxy (or wrapper) design pattern, which is generally used in the sTeam core server. For each object in sTeam a proxy is used as a placeholder for the actual object containing the whole data. It must be assured that any stored reference points to the proxy instead of the object itself.

This mechanism allows the implementation of swapping algorithm and dynamically upgrading of an objects source code, because we can just drop any object and then create a new instance with the same data. Especially the upgrade functionality is a crucial feature of a virtual environment, since it allows adding features and fixing bugs on the fly, without having to reboot the whole system.

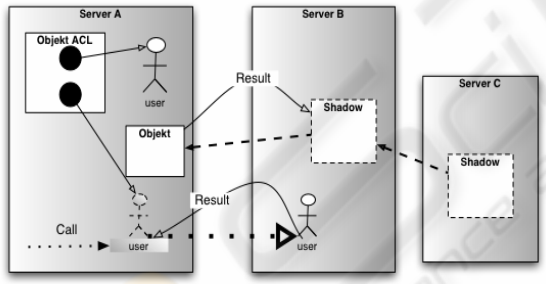

Figure 2: Shadows – References on distant objects

Every time a method is called inside the shadow object the call is forwarded to the distant server. The current action then waits for the result and so the latency between the servers is crucial for response times. Moreover the server needs to be multi-threaded in order to still be able to respond to other requests, as the active thread needs to wait for the data from the distant server and the server might get into some dead lock situation.

Apart from the integration of the COAL middleware, the only difference between a proxy and a shadow is that the shadow additionally needs to store the distant servers ID. Figure 2 shows how function calls on shadows take place.

Shadows can be references of every kind. So it is not only possible to have shadows inside an inventory listing (actually this is never the case, if objects are always on the same server as their environment), but also in an object ACL. Furthermore even a user could be a shadow, which points to a user on a different server. Here the object reference of being a member of a group comes to mind.

So whenever a shadow object is called, the server needs to create a connection to the distant server. In order to have an overview of all connections and limit the number of connections at a time, there has to be a special connection handler module keeping track of all connections. Also, there should be a pool of connections, since the number of connections should not grow out of bounds.

Furthermore it is required to keep the area partitioning of a server and retain any object inside an area locally. It is also required to move objects between servers. Any object inside sTeam has a unique environment. Moving an object from one area to another might move the object to another server, if the area is located on a different one. Such a movement might also involve the creation of a new shadow on that server as replacement of the object just leaving this server. Due to that it is possible that there are chains of shadows. Following such a chain should resolve the chain and directly store the target location and thus update the shadows.

A quite different behaviour is required when a user is moved, since we don't want to move user objects from one server to another in general. The solution is to create a temporary shadow for the users current environment and the objects inside of it. So instead of moving the user we create a new virtual environment for this user locally.

The architecture described above has already been implemented as a prototype and has been successfully tested. Inspecting the issues more closely we can distinguish between two main approaches at this point. The peer server approach and the master server approach:

3.1 Peer-Server Approach

The Peer-Server approach keeps user data on each server. A user's home area is located on that server on which the user was initially created. Although all groups are spread through the server cluster, there is one virtual group of all users on each sTeam server. Due to the transparency of the whole cluster, the „sTeam" group might also just have the meaning of giving any sTeam user of the cluster the appropriate

permissions. However, this would involve giving access permissions to a group of people, which may change unpredictable. The concept of self-administration (Hampel, 2003) might apply on some servers of the cluster in terms of allowing everyone to freely join a server. On the other hand some other servers might be much more restrictive. Also technical difficulties apply because there is already a work area of the "sTeam" group on every server. This area would then be distributed between a lot of servers and needs to be synchronized. This does not fit into the concept of having areas distributed between servers and not distributing the content inside one area (by keeping the inventory of an area locally). Due to that a "sTeam" group should always be a local group of each server. Additionally the shadow of a "sTeam" group can be used on a server in order to give users of specific servers access permissions.

The Peer-Server approach can be described as a couple of previously independent servers being loosely connected by shadows (see figure 3). Each of these servers keeps its own user- and group-base and is still under a local administration. By giving access to group-shadows users are able to allow distant groups to work cooperatively with local groups.

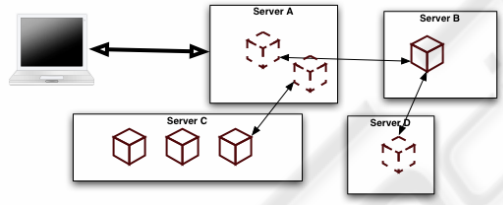

Figure 3: Peer-Server approach

Any object reference stored inside one server could be the direct reference to the object itself or to a shadow. The location of the real object addressed remains invisible to the client.

Unfortunately this causes problems with the naming of groups, because a group named „users" on server A is not distinguishable from a group with the same name on server B. This is especially harmful since it affects security. When a user gives access permissions to some groups, identification takes place just by the groups' name. A solution would be to add the servers name to the name of the group. So something like serverA.<group> could be an identifier. Of course this is not an ideal solution, as it leads to less transparency and might also confuse the user. This whole matter also applies for user names.

3.2 Master-Server Approach

A different approach would be to move all user data to one master server, which also handles the initial connection of a client (see figure 4). The Master-Server updates it's peer connection depending on the environment of the user. That is when a user moves from one area to another this might mean the user is also moved to a different server and thus also connected to that server.

To meet load-balancing issues areas should also be moved from one server to another. For example if an area on server A, which already has the most load, is accessed very often, then it should be moved to some other server with less load.

We need to keep this in mind when it comes to dynamic server clusters, because in general a server administrator don't want any of the server's data moved to some unknown server.

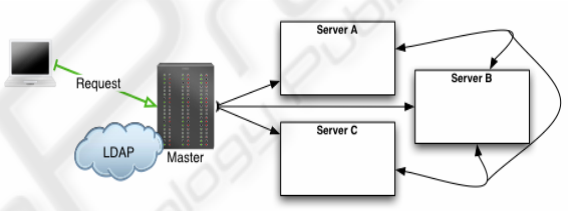

Figure 4: Master-Server approach

Moving an area actively by a user is a different matter though, because then the MOVE bit of the areas access control list is used. As described above an ACL could also contain shadows of groups of other servers, enabling members of distant groups to move an area from one server to another.

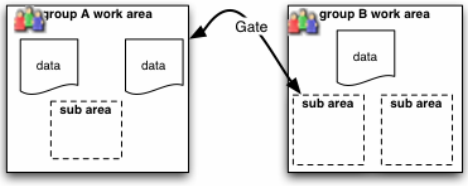

Figure 5: Group Work Areas

Observing the whole structure including the gates, we get a quite complex graph. Areas can have sub-areas in their inventory and gates connecting different areas (see figure 5). Also the graph contains a couple of trees with root nodes being the groups' work areas, as those work areas do not have an environment and are just set to be the private area of a certain group – they can be seen as starting

points into the sTeam virtual world. Apparently this leads to a partitioning of group work areas if we want to keep the locality criterion, which means keeping any object in an areas' inventory on the same server as the area (see figure 6).

Furthermore since gates connect one area with another, the distribution of areas through different servers leads to gates, which will transparently move a user to a distant server. Obviously this can be accomplished by just using a shadow as the gates destination.

This approach has some similarities to NetEffect (Tapas, 1997), which handles the partitioning of a server using communities. On each server of a cluster are a couple of usually independent communities. If a user is part of more than one community he has to specify the community he wants to use. A switch of the active community might include a connection to a different server.

Due to the fact that there is one master server keeping all the user and group data, this architecture cannot dynamically expand to new servers unless some hybrid solution of the Peer- and Master-Server approach is applied. In that case new peer servers have to be handled differently, as they have their own user base.

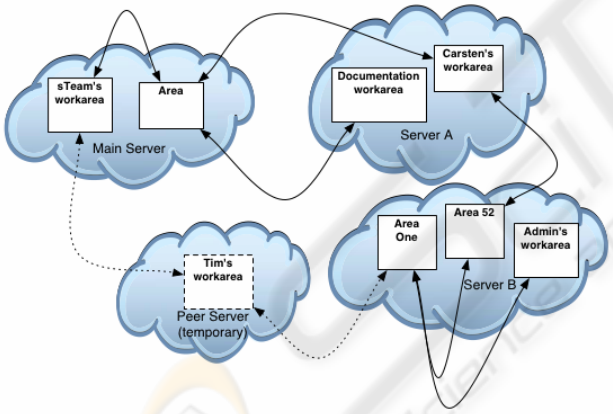

Figure 6: Area Partitioning of sTeam Server Cluster

4 CONCLUSION

This paper described the issues and different approaches to distribute areas between previously independent servers. Our goal is to connect those servers in a cluster where transparent access on distant areas is possible. Performance is less important here, so multi-server solutions using replication of data is of less interest to create a sTeam server cluster.

The most challenging problem in such a virtual world of areas is to keep a coherent access and group structure for all groups and users on any server. A central solution with a sophisticated user/group server is not sufficient for mobile environment where peer-servers on mobile devices join and leave the cluster regularly. However, we want to create a coherent solution, which should work in most scenarios. Thus hybrid architecture is the only possible approach.

Apart from that the sTeam cooperative knowledge area is an ideal base for partitioning a server into a cluster of servers, since it provides independent connected areas.

REFERENCES

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.

Hampel, T., Keil-Slawik, R. (2003): *Experience With Teaching and Learning in Cooperative Knowledge Areas*. Proceedings of the Twelfth International World Wide Web Conference, CDROM 1-8.

Hampel, T. (2003): *Our Experience With Web-Based Computer-Supported Cooperative Learning.Self-Administered Virtual Knowledge Spaces in Higher Education.* In: Proc. of Site 2003. Society for Information Technology and Teacher Education - International Conference, pages 1443-1450.

Zyda M., Pratt D., Monahan, J., Wilson K.(1992): *NPSNET: constructing a 3D virtual world* Proceedings of the 1992 symposium on Interactive 3D graphics, pages 147-156.

Greenhalgh C., Purbrick J., Snowdon D.(2000): *Inside MASSIVE-3: flexible support for data consistency and world structuring.* Proceedings of the third international conference on Collaborative virtual environments, pages 119-127.

Tapas D., Singh G.,Mitchell A.,Kumar S., McGee K. (1997). *NetEffect: a network architecture for large-scale multi-user virtual worlds.* Proceedings of the ACM symposium on Virtual reality software and technology, pages 157-163.

Vellon, M., Marple, K., Mitchell, D., Drucker, S. (1998): *The Architecture of a distributed Virtual Worlds System*. Microsoft Research: http://www.research.microsoft.com/vwg/#papers.

Lewandowski S. (1998): *Frameworks for Component-Based Client/Server Computing.* ACM Computing Surveys, Vol. 30, No. 1, pages 3-27.