Collaboration in the Virtual Laboratory for e-Sciences

Alexander de Ridder
adridder@science.uva.nl
October 2003 / October 2004

Supervisor
Adam Belloum
adam@science.uva.nl

Abstract
The Virtual Laboratory for e-Sciences seeks to provide users with a collaborative environment in which they will be able to work together across time and space while using Grid technology. In this paper we will define the requirements for collaboration in the VL-e. An in depth study of the Userlist, Instant Messenger and Telepointer has been done and a Grid Service based architecture has been designed for the first two. The architecture consists of three Grid Services and a Client. The Services have been built using the Globus Toolkit in combination with the Java Shared Data Toolkit. The architecture has been partially implemented and some test runs will be shown.

Index

1 Introduction....................4
2 Introduction to the Virtual Laboratory....................6
   2.1 Enhanced Science....................6
   2.2 Grid-based Virtual Laboratory of AMsterdam....................6
   2.3 The Virtual Laboratory for e-Sciences....................8
3 Collaboration in General....................9
   3.1 Collaboration....................9
   3.2 Failure of Collaborative Systems....................11
   3.3 Current work....................14
4 Requirements....................15
   4.1 Requirements....................15
   4.2 In Depth Analysis....................17
5 Toolkits....................31
   5.1 Toolkits....................31
   5.2 Toolkit for VL-e....................36
6 Architecture....................38
   6.1 Designing the Architecture....................38
   6.2 Using the Architecture....................46
7 Results....................50
   7.1 What has been implemented?....................50
   7.2 Results....................51
   7.3 Problems encountered....................54
8 Conclusion....................56
   8.1 Short Summary....................56
   8.2 Discussion....................57
   8.3 Future Work....................58

Bibliography....................59

Appendix A: Use Cases....................61

Chapter 1 Introduction

One of the most interesting developments in computer science must be Grid technology. By allowing users to share resources on a global scale it promises virtually unlimited processor power, infinite storage, sharing of large databases, sharing of expensive equipment, etc. In combination with the ever increasing available network bandwidth, networked R&D has interesting times ahead of it.

Grid technology is available through toolkits, that provide the user with a collection of low-level services. These can be used to develop Grid applications. To harness the strength of Grid technology for a variety of applications and to make the Grid available to a larger public, the Grid-based Virtual Laboratory of AMsterdam (VLAM-G) and its follow up the Virtual Laboratory for e-Sciences (VL-e) seek to provide a layer between the low-level Grid and the application layer (Figure 1). The Virtual Laboratory must be capable of handling large datasets coming from experimental devices regardless of the place and time of actual readings [26]. A workspace will be presented in which users can perform experiments. Furthermore users will be provided with tools that will allow them to process, analyze and manage data.

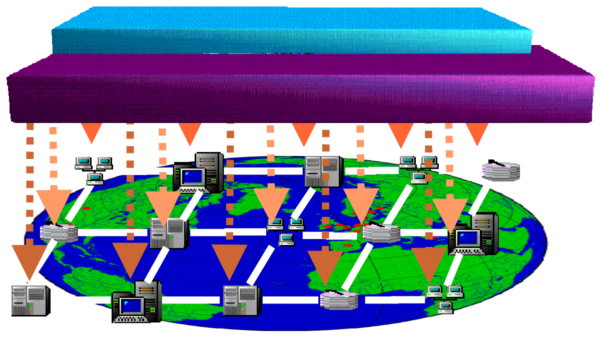


Figure 1: Virtual Laboratory Middle-Ware

The VL-e focuses on Enhanced Science, or e-Science, originally defined by Taylor [24]:

> *"e-Science is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it"*

The sharing of resources made possible by Grid technology is one form of such global collaboration. The VL-e additionally seeks to provide a more tightly coupled form of global collaboration by creating an environment allowing scientists to help each other by working together. Creating a computer supported collaborative environment is far from easy. Not only will such an application have to deal with distributed computing issues, such as data consistency management, it will also have to deal with issues specifically related to collaboration, such as awareness, GUI consistency, communication, etc.

Fortunately, the job of the software developer is made easier by the availability of collaborative toolkits. These toolkits provide the developer with a set of components that can be used to implement collaborative software.

In this report we will develop part of the VL-e’s collaborative environment. In Chapter 2 a short introduction to the Virtual Laboratory will be given. Chapter 3 will give a general introduction to collaboration. The requirements of the Collaborative System will be defined and an depth analysis of some particular components will be done in Chapter 4. Chapter 5 will explore collaborative toolkits found on the Internet. In Chapter 6 the requirements will be turned into an architecture and its results will be shown in Chapter 7. Finally we will draw a conclusion and shed some light on future work in Chapter 8.

Chapter 2 Introduction to the Virtual Laboratory

In this chapter we will give a short introduction to both the Grid-based Virtual Laboratory of AMsterdam (VLAM-G) and to its follow-up project the Virtual Laboratory of e-Science (VL-e).

2.1 Enhanced Science

Networked R&D has lead to a new paradigm in scientific research called (digitally) enhanced science or e-Science. In 2001 John Taylor [24] defined it as:

> *"e-Science is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it"*

This global collaboration can come in many forms. At its best people from all over the world share equipment, resources and databases while performing experiments, sharing results, discussing insights, and so forth. However, people may not be that keen on sharing their data and institutions may be unwilling to share expensive equipment or cutting edge technologies, especially if significant money is involved.

An infrastructure that allows global sharing of resources is the Grid. It allows, for example, users to submit CPU intensive jobs with the calculation being distributed over multiple processors. Grid technology is available through toolkits that offer a collection of services, such as data management, security, etc. These toolkits are, however, concerned with low-level details, making them unusable to the average scientist.

2.2 Grid-based Virtual Laboratory of Amsterdam

Based on the Globus toolkit the VLAM-G provides a layer between the applications level and the grid-service layer, thus harnessing the strength of Grid technology for a wide variety of applications and making it available to a wider public. The VLAM-G provides a scientific portal for remote experiment control and collaborative, Grid-based distributed analysis in applied sciences [1]. It differs from other work in this area in that it seeks to provide a solution for multiple classes of applications instead of for just one. For technical information about VLAM-G toolkit, the reader is referred to [1, 6]. Here we shall give a short overview of how a scientist can use the Virtual Lab.

A study in the Virtual Lab is a formalized series of steps (workflow), intended to solve a particular problem in a particular domain. This workflow is represented by a Process Flow Template (PFT). The content of a process step in the PFT represents an experiment executed either by human intervention or by an automatic process. PFTs are defined through the PFT editor and are stored in a database for later use.

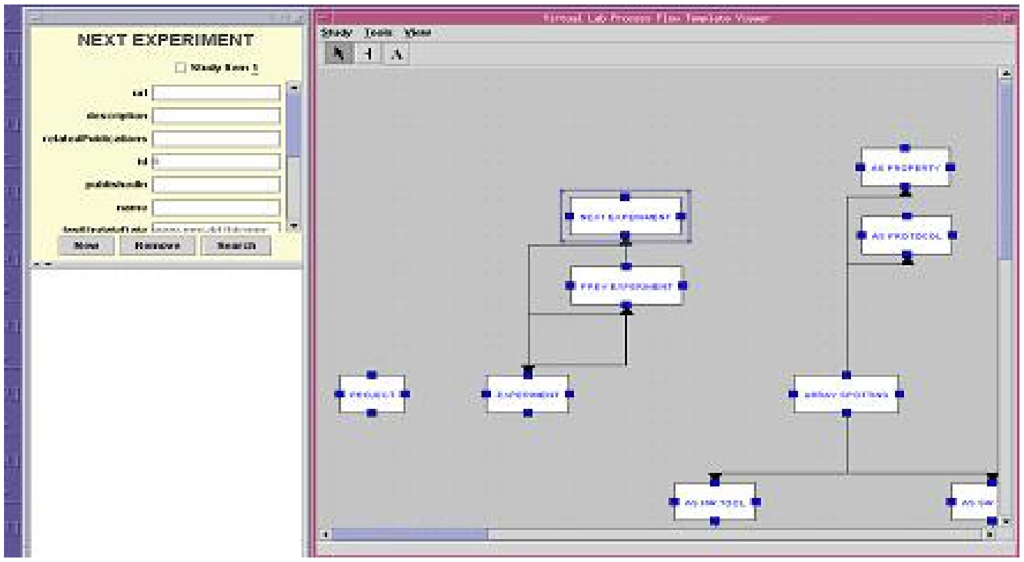
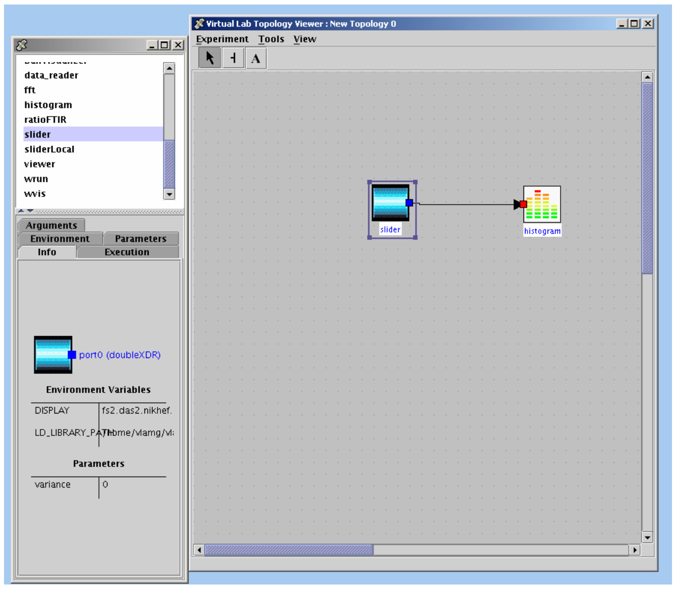


Figure 2: PFT Viewer

When a user wants to perform an experiment he selects a previously defined PFT from the database. As the template itself cannot be changed, an instance, the Process Flow Instance (PFI), is created for him to work with (Figure 2).



Environment Variables

DISPLAY	fs2.das2.nikhef.
LD_LIBRARY_PATH	/home/vlamg/vl:

Parameters

variance	0

Figure 3: Topology Editor

Each PFI contains at least one operational step, which is further defined by using the Topology Editor (Figure 3). Here the experiment topology is created by drag-and-dropping modules into the work area and connecting them together using directed data flows. These modules can range from input to operational to output and visualization components. Because of its modular architecture new components can easily be added, making it suitable for a vast amount of applications. An instance of the topology is sent to the Run-Time-System for execution.

2.3 The Virtual Laboratory for e-Sciences

The VLAM-G project officially ended in September 2003 (?). The number of interested parties had grown spectacularly over time and green light was given for the follow-up project, the Virtual Lab for e-Sciences (VL-e). It will extend the features of the VLAM-G where possible, but, unlike the VLAM-G, the VL-e will have a service-oriented architecture. This has some important consequences. For one, individual components can be addressed only via their published interface, which is made available to the network. This allows for decoupling between individual components, as the only connections between them is via these interfaces. Furthermore services and multiple instances of a single service can be located on different machines. When a machine goes down, the same service may still be available on another machine. This gives greater overall stability. To achieve location transparency, applications search for services in a directory after which they connect to them dynamically at run-time. This allows for code mobility, as the client does not care where the service is located [27].
Grid technology allows for a special kind of services: Grid Services. Grid Services are basically Web Services with enhanced capabilities, including notifications (events between client and server), lifecycle management, and statefulness. The latter allows storage of information on the server side. This has led to the availability of Factories that create instances, which can be accessed by multiple users at the same time, allowing them to work on the same data.

Where the VLAM-G supports global collaboration through resource sharing, the VL-e additionally seeks to provide more fine-grained global collaboration. It seeks to provide its users with a shared workspace environment, allowing them to conduct experiments together. Means must be provided for them to communicate. However, it is important to first understand collaboration.

Chapter 3 Collaboration in General

Before we can focus on collaboration for the VL-e, it is important to get a clear picture on collaboration in general. In this chapter we will explain the concept of collaboration and its importance and we will use research done by others before us to explore why collaborative applications have a tendency to fail.

3.1 Collaboration

Not a day in our lives goes by without us collaborating. Collaboration is about two or more people working together towards a joint goal. It differs from cooperation in that with collaboration all sides benefit from working together and that it is on an equal basis. Cooperation on the other hand can also benefit only one side and is often more connected to hierarchy. A police-officer would, for example, ask a criminal to cooperate and not to collaborate.

One important aspect of collaboration is communication. When two people engage in verbal communication they exchange words, but a lot more is involved: intonation, body-language, body odor, etc. Even simply being there and saying nothing is communicating something to the other person. All such elements give extra information about a person's character and mood and provide better insight on how to proceed in the discussion; the more information, the better. Furthermore, interaction with the artifacts in the environment, like someone tearing up a piece of paper, provides extra information and so are part of the collaborative process. Face-to-face communication is incredibly complex and effective.

Unfortunately, direct face-to-face collaboration is not always possible. When, for example, working on a project with people located at several different locations it may be possible to meet every now and then, but often not on a daily basis. The expenses are simply too high and it is much too time consuming. Luckily, technologies are available which allow us to collaborate over distances. One of the best known and widely used technologies is teleconferencing, where two or more people meet via the telephone. Though it works, it comes at the cost of information: it does provide intonation but removes any physical communication. Sending a message by mail has even more problems. It removes intonation, physical information and does not allow for instantaneous communication. With video conferencing video images of people in one room are transmitted to other participants in rooms at distant sites. In the simplest case, only an image of the entire room is transmitted. More complex scenarios would have a single screen for every participant. Video conferencing captures some physical information as well as audio. Though one would expect video to be an incredible advantage, it seems to fall far short of expectations [8].

Computers provide us with even more possibilities for collaboration. Computer Supported Cooperative Work (CSCW) is the field that focuses on the design, adoption, use and social implications of collaborative systems, or groupware. Though people use the term groupware for any multi-user piece of software which supports groupwork, there is ongoing debate on what the actual definition of groupware is [2]. One issue is whether it applies strictly to soft-, or hardware, or whether it also applies to group techniques. Another issue is whether the emphasis is on “group” or on “ware”. Those emphasizing “group” place emphasis on the group process, while those emphasizing “ware” are more focused on the technological part. One of the most widely used definitions of groupware is the original one by Peter and Trudy Johnson-Lenz [20]:

> *"The combination of intentionally chosen group processes and procedures plus the computer software to support them."*

Even though there is no definition on which all are agreed, the fact remains that computers can be used for collaborative purposes. For the purpose of this paper we will think of groupware as computer based systems which allow for collaboration.

Dimensions of Groupware

Groupware technologies are typically categorized along two dimensions: the dimensions “time” and “place”. “Time” is split into synchronous and asynchronous groupware. In the first, also often referred to as “realtime” groupware, people are working together at the same time. In the latter time can be different. Some well known asynchronous groupware applications include email, message boards, newsgroups and electronic calendars. Well known synchronous groupware includes instant messengers, video communications, shared whiteboards, etc.
The amount of distance between collaborators matters. The “place” dimension makes a distinction between whether people are working at the same place or at different places. Mark Brader’s research [8] has shown that an increased distance initially has a negative effect on cooperation, though after a longer period of interaction cooperation improves. Furthermore, people are more likely to give deceptive (positive) portrayals about themselves to, and are less easily persuaded by, a person whom they believe to be farther away. This study, as the authors themselves note, has several limitations, including the fact they conducted their research in the context of a laboratory rather than in a context of the workplace. Furthermore the individuals in the experiment were unacquainted, whereas in the workplace this is not necessarily the case. Nevertheless, this research has shown that distance does matter.

Shared Workspace Environments

One important class of systems, and the one we will be focusing one, is that of applications that support realtime, distant collaboration through a shared workspace. A group of users share the same workspace, with the same objects, allowing them to work synchronously on the same project. Analogously to WYSIWYG (What You See Is What You Get), shared workspaces have WYSIWIS (What You See Is What I See). WYSIWIS comes in two versions: relaxed and strict. In strict WYSIWIS, all users have the same viewport. It is easier to create than its strict counterpart and provides a better sense of awareness of what other users are doing, as everything happens in sight. Relaxed WYSIWIS provides greater freedom, allowing users to scroll around the workspace freely. Not only is it harder to create but, as users can work outside someone's view, workspace awareness is less.

3.2 Failure of Collaborative Systems

Collaborative systems tend to fail, or at least fall short far of expectations. There are many reasons for this high rate of failure including social differences between collaborators, difficulty of testing groupware and organizational culture. Grudin [18] points out that in order for an application to succeed it has to be beneficial to all its users. For example, a schedule in which a manager can look up the agendas of all other employees is destined to fail, as the manager is the only one who will benefit from it. Email on the other hand is beneficial to all users and is an incredible success. Therefore an application has to appear beneficial to all who have to use it. The VL-e is an application which should be beneficial to its users, as it allows them to make use of the power of the Grid.

Greenberg and Gutwin [4] believe that problems with groupware are often caused by a failure to support basic necessities. They introduced the Mechanics of Collaboration, the things groups have to do in shared workspaces, over and above what an individual has to do, in order to carry out a task. They have combined these Mechanics with Nielsen's Usability Heuristics to create heuristics for groupware [17]. They are meant for validating the quality of software but can also be used to aid in defining the requirements of our collaborative system, as we shall see in Chapter 4.

Heuristics based on The Mechanics of Collaboration

Heuristic 1: Provide the means for intentional and appropriate verbal communication

When two or more people communicate, words are exchanged. During such verbal exchanges there are three ways in which information is picked up [17]:

1. Explicit conversation between two people
2. Overhearing a conversation between others
3. Overhearing commentary that others produce alongside their actions

The first is relatively easy to support in groupware. Most of the communicative applications are meant to support explicit communication. The second item can be supported if the conversation between two people is taking place in a shared space to which the entire group has access, like a group message space. Overhearing commentary is harder, as people will generally not type the commentary they are producing alongside their actions. To support overhearing commentary would require for example a webcam or an audio device, always turned on. People generally dislike such monitoring as they feel it is an intrusion to their privacy.

Some typical examples of groupware supplying verbal communication are instant messengers, black/white boards, digital audio communication, video, etc.

Heuristic 2: Provide the means for intentional and appropriate gestural communication
A large portion of the actions during collaboration are intentional gestures. People use intentional gestures to support the conversation. Intentional gestures take many forms [17].

1. Speech can be used as illustration. For example, showing a gap between one’s hand and the floor to illustrate height.
2. Words can be replaced by actions. For example thumbs up, as a replacement for “okay”.
3. People can reference objects with a combination of intentional gestures and communication. For example requesting “that pencil” while pointing at it.

The simplest form of embodiment in a shared workspace is a telepointer. A telepointer is a normal pointer except that it is visible to everyone inside the workspace. Telepointers can be used to gesture and point at objects. They also provide gaze awareness, as it is likely a person is gazing near his pointer. If means for intentional communication is also provided, the third item is taken care off. The second item can be provided if the telepointer can take varying shapes, like changing into a pen when someone is writing. Illustration, the first item, is also possible by providing a white board. This allows users to make simple drawings to illustrate their meaning.

Heuristic 3: Provide consequential communication of an individual’s embodiment
Some signals are not intentionally given off by people, but can be picked up by the perceiver nonetheless. Consequential communication of an individual’s embodiment can be split into two categories [17].

1. Actions coupled with the workspace, such as seeing someone’s gaze
2. Actions coupled to conversation, such as facial expressions, eye contact, intonation, pauses

Capturing this in groupware is hard. A telepointer, for example, does give some indication of what the person is looking at, but it is no guarantee. Avatars can capture the gaze direction, but still many consequential gestures are not captured. Video should be able to provide conversational awareness, but so far is falling short of expectations.

Heuristic 4: Provide consequential communication of shared artifacts (feedthrough)
When an artifact is manipulated, it gives off information. For example, when typing on a keyboard, there is the sound of the keyboard itself. Other who hear this sound, know someone is typing. By seeing and hearing an artifact as it is manipulated, people can easily determine what others are doing with it.
Artifacts can provide two types of communication [17].

1. Feedback, which is the information the user receives when he is manipulating the artifact
2. Feedthrough, which is the information the others who are watching receive

Feedthrough has to be supported for other users to understand how the objects are changing. This can be provided by, for example, adding a history of how the artifact was manipulated. Feedback is often automatically provided, for example by displaying what a user is typing.

Heuristic 5: Provide protection

In a shared workspace users can access the same artifacts at the same time. If no protection is provided, it is possible for people to edit a single object simultaneously. This may cause conflicting situations with users committing changes which are immediately overwritten by others who are also committing the same object. If enough workspace awareness is provided, natural collaboration will remove many conflicts, as it is considered to be rude to start making changes to something while someone else is editing it. In some cases conflicts may even be acceptable, but for most applications protection must be provided to prevent people from altering or destroying work that others have done.

Groupware often provides access control, concurrency control, undo, version control, etc. to assist with social protocols.

Heuristic 6: Management of tightly and loosely-coupled collaboration

During group projects, people continuously shift between individual and groupwork. After a certain amount of individual work, coordination is necessary between the team members, after which the individual work will continue. The amount of work people have to do before they require contact with another person is known as coupling. To be able to switch between group and individual work requires the ability to focus on different parts of the workspace. Shifting focus can, however, lead in loss of awareness of the overall picture, especially in relaxed WYSIWIS environments. Therefore the entire workspace should be provided in a small secondary window. These can come in many forms, like overviews, radar views (overview with telepointers shown), detail views, etc.

Heuristic 7: Allow people to coordinate their actions

In group projects it is necessary for people to be able to coordinate their actions to make sure that actions happen in the right order, at the right time. Many of the previously described heuristics are required to allow for this. It requires awareness features, as coordination requires overall knowledge of a project. Furthermore, collaborators have to be able to communicate verbally.

Heuristic 8: Facilitate finding collaborators and establishing contact

Users of a groupware application can be distributed across time and space. In order to be able to collaborate with other users, it is necessary to know who is available. After that, many problems can still arise while trying to initiate contact, such as incompatible software, lack of hardware (e.g. a webcam), etc.

Checking availability and initiating contact must be possible with minimal effort. A simple solution is the one provided by instant messengers; a userlist shows the availability of people.

3.3 Current work

The CSCW field is rapidly evolving and many interesting applications are being developed. One interesting project has succeeded in the 3-D reconstruction of a remote collaborator into the scene of the user [23]. The user views the world via a head-mounted display with a small security camera attached to the front. To create a realistic model of the remote collaborator, he is surrounded by fifteen cameras. By calculating the geometrical relationship between the user's camera and a marker, the appropriate view of the distant collaborator is displayed in the user's scene. Due to its high performance the system gives the impression that the distant person really is part of the scene.

Another project is Microsoft Research's Sideshow, an application which desires to provide users with peripheral awareness of important information [9]. They provide a sidebar which can be filled with tickets. These tickets hold small up-to-date information about an application. For example, an e-mail ticket would display whether a new message has arrived, an e-bay ticket would keep track of an auction, etc. Users can add and remove tickets to and from their sidebars, change the size each tickets takes, etc. Microsoft has tested their sidebar and concluded that users were willing to sacrifice a small part of their workspace in exchange for awareness. The tickets have to be individually customizable.

Chapter 4 Requirements

In this chapter we will explore the requirements for collaboration in the Virtual Laboratory for e-Science. Furthermore we will explore the importance of awareness and data consistency algorithms and do an in depth analysis of the Instant Messenger, Userlist and Telepointer.

4.1 Requirements

As stated in Chapter 2 the VL-e will be a shared workspace environment. The Mechanics of Collaboration, described in Chapter 3, are therefore valid for our project. Based on these as well as on other sources, such as [25], our requirements can be defined. They have been split into groups, beginning with the ones crucial for the collaborative environment while ending with optional enhancements.

Requirements 1

The first set of requirements should allow for minimal collaboration in a single workspace. Minimal collaboration should allow users to discuss existing studies. This requires users to be able to enter a study, navigate through it, find other users, initiate contact and communicate with them. This gives the following set of requirements:

- Session control: decides who gets to go in. Includes finding out what sessions are available, determining who can enter and exit the session, and when and how.
- Exploration of the space or of a set of artifacts: this allows for the analysis of artifacts and for movement around the workspace.
- Means for intentional and appropriate gestural communication 1: a simple telepointer to allow for simple gestural communication.
- Finding collaborators and establishing contact: a simple user list to know who is active in the workspace and which allows establishing contact.
- Means for intentional and appropriate verbal communication 1: a simple messenger to allow for simple text communication.

These features allow limited collaboration on an existing project: starting a session, finding collaborators and communicating with them, analyzing artifacts and ending the session. As artifacts are merely read there is no need for data consistency algorithms yet.

Requirements 2

The next set of requirements should provide the means to allow for creation and alteration of objects in a collaborative session. As now the artifacts are not merely read, but also written, there is need for data consistency management.

- Creation of new artifacts: allows the creation of new artifacts in the shared workspace
- Organization of existing artifacts: allows for changing existing artifacts
- Provide protection 1: in order to prevent multiple users from changing an artifact at the same time, data consistency mechanisms have to be provided.

A natural next step is to allow for customization of access to artifacts to protect them from unwanted alteration. Additionally, critical information has to be secure even against aggressive attempts to obtain the information

- Provide protection 2: provide workspace control to limit access to components.
- Provide protection 3: protect critical information.

In a shared-workspace environment, especially in a relaxed WYSIWIS, it is hard to remain aware of the actions of other users. Awareness is crucial for the success of a collaborative system.

- Management of tightly and loosely-coupled collaboration: adding workspace awareness by providing a radar / birds-eye view of the workspace. The pointers of the other users are shown herein.

With multiple users altering the project simultaneously, there is now need for fault tolerance in case of an abnormal termination

- Provide protection 4: protect in case of abnormal termination by adding fault tolerance

A system which has all the previous features, should allow for users to work together in a convenient way.

Requirements 3

Additional protective features can be added, such as undo/redo and merging. Asynchronous communication will allow more convenient communication across time and has the additional benefit of storing previous communication. Feedback and feedthrough of artifacts provides additional awareness.

- Means for intentional and appropriate verbal communication 2: allow for communication across time by providing asynchronous communication, such as a message board.
- Provide consequential communication of shared artifacts: providing the user with artifact feedback, thus providing information on artifact evolution.

- Provide protection 3: add additional protective features such as undo/redo and merging.

Requirements 4
Adding audio as a communicative device can be an enormous improvement. It allows users to use their natural language, which, in many cases, is far more efficient than having to type a message. Also, the ability to convey emotion is important.

- Means for intentional and appropriate verbal communication 3: adding audio to improve communication

Allowing for the construction of larger objects from component pieces not only reduces the chaos on the workspace, but may also make the entire component reusable.

- Construction of larger objects from component pieces
- The management of an autonomous system represented in the workspace

Requirements 5
The potential of video as a collaborative device is still uncertain. As Sellen [8] discovered, video does not appear to be an improvement compared to high quality audio and therefore should be considered ultimately.

- Means for intentional and appropriate verbal communication 4: adding video

Optional Requirements
The Mechanics of Collaboration described in the previous chapter suggest additionally adding:

- Means for intentional and appropriate gestural communication 2: adding avatars, video images, etc.
- Provide consequential communication of an individual's embodiment: adding for example talking heads, a mapping of your face to a 3D head.

Even though these features may help collaboration, it is questionable whether it is realistic to propose them as a requirement for the VL-e.

4.2 In Depth Analysis

Since we are extending an existing system, some features may already (partially) exist or may be closely related to work which will be done by other VL-e developers. Two of the five points described as first basic necessities are not solely collaborative features. The first one, Session control, is related to the VL-e’s access control, as those who are allowed to enter a specific study are also those who are allowed to participate in a collaborative activity. The second feature, navigating the workspace, is also a feature which is related to other VL-e components as the graphics are provided by the VL-e components themselves. In this report we shall focus on the three remaining requirements. Furthermore we will touch the subjects of data consistency management and management of loosely-coupled collaboration.

In Depth Analysis: Shared-Workspace Awareness

In shared workspace environments it is important to remain aware of other people's actions. According to Gutwin and Greenberg [15], awareness plays a key role in the fluidity and naturalness of collaboration. They define workspace awareness as:

> "The up-to-the-moment understanding of another person's interaction with the shared workspace. "

This emphasizes the importance of awareness of interaction between people and the workspace.

A shared workspace environment should provide the means to answer basic questions, such as:

- Is anyone here?
- Who is that?
- Who is manipulating that artifact?
- What is happening?
- Where is he working?
- Where is he looking?
- What can he see / manipulate?

Such questions can be answered in varying ways. A userlist, for example, can answer the first question. A telepointer can, if properly designed, answer all of them, but only if it remains inside the current view. By providing an overview of the entire workspace in a secondary window, showing Telepointers and objects, it is possible to expand the awareness beyond the point of focus. Adding view rectangles around the Telepointers in the secondary view will enhance the awareness even further, as users can see what other users can see and manipulate. However, when too many users are active awareness will decrease as it will be hard to get a clear view of what people are doing. The view rectangles will overlap, messing up the radar view. As Johnson noted in [21], requirements for architectures supporting "collaboration-in-the-small" are fundamentally different from those supporting "collaboration-in-the-large", such as the WWW.

In Depth Analysis: Data Consistency Management

Just like in normal distributed systems, collaborative systems require data to be kept consistent. However, with collaborative systems when events are handled out of order displays may become inconsistent, causing confusion amongst the collaborators. The activity of coordinating potentially interfering actions of processes that operate in parallel is known as concurrency control [13].

One such concurrency control method is synchronization. A non-optimistic synchronization policy ensures that all events are handled in order. The scheduler delays events until all its predecessors have arrived. This can make the policy slow, but it does guarantee consistency. Optimistic synchronization policies assume that conflicts rarely occur and therefore that events will rarely arrive out of order. Events are handled as soon as they arrive, making execution a lot more efficient. However, when a conflict does occur, it will have to be repaired. The algorithms needed to repair conflicts are often complex and even then not all actions are repairable.

Concurrency can also be managed through the use of locking mechanisms. Here, a user requests a lock to an object after which, if approved, he is granted privileged access to it for a certain amount of time. Non-optimistic locking forces a user to wait until the lock is granted before he is allowed to edit the desired object. Optimistic locking allows a user to immediately start altering the object; if the lock request is denied, the object has to be returned to its old state. Optimistic locking is divided into two main groups, fully-optimistic and semi-optimistic locking. In semi-optimistic locking the user cannot start editing another object until the lock request for the previous one has been answered (either denied or accepted). Fully-optimistic locking, on the other hand, allows users to start editing other objects, perhaps even based on the state of an object whose change has not yet been approved. When the first lock is denied, this can therefore require very complex undo mechanisms.
The grainsize of locking is also an important issue. For example, are the locks on entire objects, on a single property of an object, or on a single character which defines a property? Different grainsizes give a different feel.

The proper scheme for the VL-e will have to be considered carefully. Non-optimistic schemes will guarantee that no conflicts occur, but may cause irritating delays. Optimistic schemes may cause irrational behavior when an error is corrected, for example an object suddenly jumping back to an old position in the workspace. If sufficient awareness is provided, making it is clear that someone else is changing that object, natural collaboration should make conflicts rare though, as it is obviously rude to change alter and object while someone else is working on it. Another issue is the network. In high latency networks waiting times will increase and performance will suffer too much. In such a case an optimistic policy has to be provided. If latencies are low, there is less reason to implement optimistic policies.
A final consideration is resources. Optimistic policies require much more resources than their non-optimistic counterparts.

In Depth Analysis: Instant Messenger and Userlist

Background

Instant Messengers (IMs) are communicative applications which allow users to type and send messages to other users. Contacts of users are kept on buddy-, or userlists. By opening a message window to one of the contacts, text messages can be exchanged with that user. The message window provide a space to type a message and space in which messages are displayed. Given the popularity of MSN, ICQ, AOL, Trillian and other such applications, they have proven their worth as a communicative device, though this does not necessarily make them an effective collaborative device.

IMs are an interesting form of communication. They are extremely useful for informal communication and allow for complex collaboration, such as joint problem solving, coordination, social bonding, and social learning [22]. Since IMs are so informal, users have to spend less time on formalities, allowing for quick questions and equally informal and quick answers. Another big advantage of IMs is the message windows remaining open until closed by the user. If a user receives a message while not being around, on return to his computer he'll find the message, still allowing him to respond. This makes IMs a tool for synchronous as well as asynchronous communication. An additional effect is that IMs allows for plausible deniability about one's presence [22]. As a sender is unaware whether the receiver is actually their or not, the receiver can easily ignore the message and answer it when he sees fit. The sender will not be offended, as he presumes the receiver was simply away. IMs thus provide recipients with control in deciding whether and when to respond to a message.

On some occasions an IM will not suffice, as some problems are too hard to describe with text messages. Furthermore IMs' lack of emotional representation can lead to confusion on both sides. Though many messengers provide small icons to express emotion (emoticons), this is not nearly enough to simulate real emotions. On occasions where an IM does not suffice, switching media is necessary and the IM can be used to determine whether the other user is available and has time for, for example, a phone call.

Handel and Herbsleb [19] point out that IMs have many disadvantages. The often used pop-up window when a message is received is a distraction for the receiver. Lack of emotion and texts which have been given too little thought can spark online arguments. Instead of using an IM, they suggest using a public chat space. In a public chat space, everyone in the group reads and writes to the same persistent window. As this is by far more public than an IM, conversation will more focused on work and users will be less likely to gossip, flame, etc. An important advantages of group chat over IM, is that it is less intrusive. It is simply always there and people will respond when they feel like it. The main disadvantage is that someone may miss things addressed to him, simply because he is not paying attention. This can be fixed by, for example, allowing someone to produce a beep to alert someone else, but this will be distraction for the receiver. The balance between distraction and chance of missing an important message must be carefully considered with public chat spaces as well as with IMs.

As stated before, IMs make use of buddy-, or userlists. These contain the names of the people with whom the user wishes to keep in touch and also gives information about users' availability.

It can also be used for many other things, such as inviting people to join a conversation, starting a video conference, initiating an audio connection, sms-ing, etc. It can be the very basis of the communicative part of the collaborative system.

Design: Userlist

In designing the concept of the userlist, several questions require answering:

- Who are on the userlist?
- How can contacts be added?
- Who are the contacts?
- How are the contacts organized?
- Is a user available?
- What other functionalities does the userlist provide?

Who are on the userlist?

Studies in the VL-e will in general have a fixed group of people. As collaboration should take place primarily amongst those people, it makes sense to have those people on the userlist. When a user logs into the study, the study will provide him with the study's userlist[1], which will then be added to the user's userlist.

Users can be part of more than one study at a time, but are not necessarily active in all of them. For example, when a study is performing massive calculations the user may well decide to check back on them later. We believe that when a user logs into a study the accesslists of all his running studies (online studies in which he participates, but in which he is not necessarily active at the time) should be part of the userlist. This will allow him to communicate with all of the users of his running studies and vice versa. If something important comes up a user can easily be contacted, enhancing collaboration.

Collaboration can be further enhanced by allowing users to keep Personal Contacts. For example, when two brain surgeons are discussing a 3D model of a brain, if one of them has befriended another surgeon, it may be useful to invite him to join the study, even though he is not part of the accesslist. Having someone in a personal userlist is far better than having to look that person up. Not only does it save time, but it also allows the user to determine the other's availability. Furthermore is allows users to keep in touch. Even though this may prove to be distracting, it will improve collaboration; people are more willing to help someone they know.

How can contacts be added?

The content of the userlist is primarily defined by the studies in which the user participates. By joining a new study, the userlist is automatically updated with the study's accesslist. Personal

[1] From now on, we shall refer to a study's userlist as accesslist. This will decrease confusion with the user's userlist. Furthermore, since access to a study is indeed limited to the users on the accesslist, it is an accurate description.

contacts can be added by either searching a database for a user, or by adding them from the accesslist. Being able to search for contacts is also important when looking for experts to invite temporarily to a session. As this issue overlaps with adding users to a study's accesslist when a study is created, this issue will have to be discussed with the rest of the VL-e developers.

Who are the contacts?
In a highly secure workspace such as the VL-e, personal information must be available. As Grid credentials are used to identify a person, there is no doubt whether a person is who he claims to be. By making personal information available anonymity is removed, making everyone responsible for their actions and behavior. The amount of misuse should therefore decrease dramatically. A pop-up menu should provide users with the option to read a person's personal information.

How are contacts organized?
As stated previously, users should be able to communicate with the users of all their running studies. It does make sense to display the studies in which the user is currently active on top in the userlist, as communication will most likely take place within a user's active sessions.

Is a user available?
By providing states which reflect a user's availability and displaying this on the userlist, availability can easily be determined. A user will be able to be in several states: "online", "online in other", "offline", "do not disturb" and "away". When a user logs into a study, he will be "online" that study. In his running studies, he will appear as "online in other". When he logs out again, his state will revert to "offline". A user who does not want to be disturbed can set his state to "do not disturb". The "away" state is slightly more complex. Not only should the user be able to change his state to this mode, but it should also automatically go into this mode if a user does not use his mouse or keyboard for some time.
Unfortunately states are often misused. For example, setting a status to "away", while simply sitting behind one's desk. A solution would be not to allow the user to change his state to "away", but to let the system handle it. If, for example, the user does not use his mouse for 30 seconds, the mode changes to "inactive" and after another 30 seconds to "away". A user trying to send a message will receive a warning that the receiver is away after which he may decide whether to send the message or cancel it. Unfortunately there is no such solution for "do not disturb", as it is impossible for the system to determine whether someone truly does not wish to be disturbed. A solution is partially presented by social protocols. By providing the sender with a warning that the user does not want to be disturbed unless it is urgent a user may think twice about sending the message. Still, if the mode is too often misused (e.g. someone always being in "do not disturb" mode but always answering and chatting), such warnings will lose potential. The effectiveness is therefore largely dependent on the reactions on the receiver side. If someone

truly does not want to be disturbed when he is in that mode, users contacting that person for idle chatter will learn that quickly.
Another matter is that managers, project leaders, etc. want some way to entirely block incoming messages. Since such mechanisms disturb collaboration, it is questionable whether such an option should be provided to all the users, or should be limited to specific groups. We are currently exploring the possibilities of the latter by providing users with a collaborative level based on which they have more (or less) options available to them.

What other functionalities does the userlist provide?
Besides providing awareness, the userlist can be integrated with the instant messenger. Furthermore it can be used to invite people to join sessions, start audio conferences and video conferences, and to send sms messages.

Visual Representation
We have something in mind which looks like Figure 4.

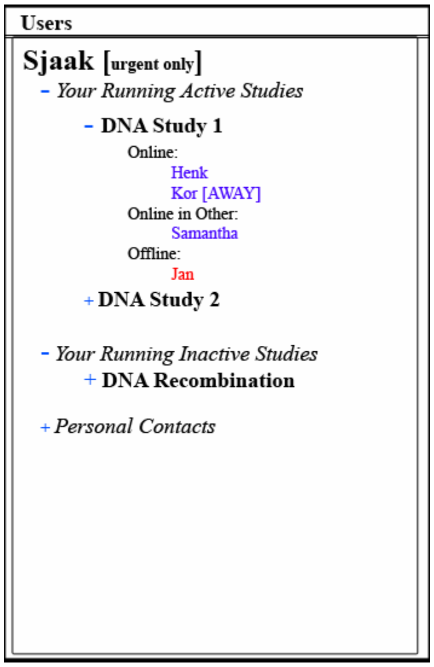


Figure 4: Userlist Concept

Discussion
The suggested userlist has advantages as well as disadvantages. We expect that the way it is presented will give users a clear sight of who is available and who is not, thus effectively providing some awareness. Furthermore it is possible to expand and contract the accesslists, allowing users to focus on the ones that are of interest. This may also reduce network traffic, as

it is unnecessary to receive updates on states in which the user is not interested. A minor disadvantage is that when a user logs into a study, it will have to be moved in the userlist.

First Version

The first version will be a very basic userlist. It will allow only one accesslist and will support a variety of states, though all of them will be handled manually. Blocking incoming messages will not yet possible.

Design: Instant Messenger

The IM has the user list as a basis. A number of issues have to be taken care of:

- How can users send messages?
- How are users made aware of received messages?
- How do user states (defined by the userlist) effect the messenger?
- How can users chat with multiple users?
- What happens when a user destroys his message window?
- What other functionalities does the Instant Messenger provide?

How can users send messages?

By clicking on a user name from the userlist a message window will open. Each conversation has its own message window. The user can type a message, followed by pressing either enter or clicking the “send” button. As soon as the user has sent his message, the message will be displayed on the sending as well as on the receiving side’s message window. If no message window for the conversation exists at the receiving side (it is either the first message, or the receiver has killed the previous window) a new message window will be created. The receiving side can thereafter send messages in the same fashion as the sender.

How are users made aware of received messages?

As stated previously, the balance between awareness and disturbance has to be considered. For example, playing a sound when a message is received will make the person aware of the message, but it is highly disturbing and often annoying. On the other hand, if no feedback at all is provided, users will not be aware of messages. Tolerance to disturbance is user dependent and the disturbance/awareness balance should therefore be made customizable. Naturally a minimum level of awareness should be provided, perhaps something similar to Microsoft’s Sideshow, displaying a small part of the message near the username on the userlist.

How do user states affect messages?

The states a user can be in should have some effect on the messenger. Someone who is “online” will be able to receive messages normally. Both “do not disturb” and “away” should provide the sender with a warning. Users who are “offline” are, of course, unable to receive messages. This will be changed once offline messaging is added for asynchronous communication. Offline messaging will simply store a message somewhere and when the user logs in, he will receive the message.

How can users chat with multiple users?
Users can address messages to multiple users in two ways. The first will allow them to drag a username into an existing message window, adding the user to the conversation.
The second way will be from inside the message window. A menu item will be provided which will allow a user to add someone to the conversation by selecting someone from his userlist or searching a database. The selected user is added to the conversation.
Additionally it is possible to add a feature which will allow users to send a message to all users in the study. This would allow for easily talking to all within the study.

What happens when a user destroys his message window?
The message window will be terminated and the user leaves the conversation. In a one-on-one conversation, if the user who did not close his window sends another message, a new message window will open at the receiver side. If the user who destroyed his window opens a new window to the same user sends a message, it should arrive in the other user’s old window.

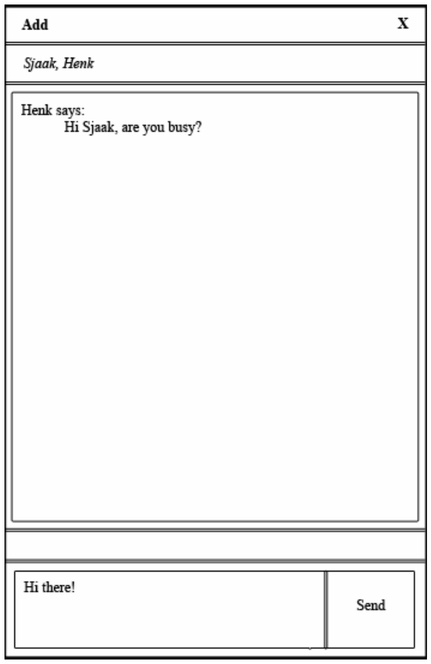


Figure 5: Message Window Concept

What other functionalities does the Instant Messenger provide?

File transfer may be an interesting feature for the message window as it would allow multiple users to receive the same file.

Visual Representation
The Instant Messenger should look something like Figure 5.

Discussion
The one point of discussion concerns our choice for IM instead of group chat. The main reason for this is that with group chat all communication would take place inside the study's group chat space. If a user is active in multiple studies, he would have to keep an eye on all the separate chat spaces. Furthermore contacting someone outside of the session would in this case require adding that user to the group chat space. For what we need for VL-e, IM is better suited.

First Version
The first version of the Instant Messenger should simply allow two users to send messages to each other.

In Depth Analysis: Telepointer

Background

Telepointers are mouse pointers which are visible to all users within the shared workspace. They are simple and computationally inexpensive. They provide embodiment, gestures and a means of coordination.
In strict WYSIWIS constructing a telepointer is fairly easy: simply map the coordinates of the pointers to the same coordinates on all displays. Relaxing WYSIWIS makes matters more complex. In workspaces which provide a simple form of relaxed WYSIWIS telepointer mapping can be done by using world coordinates. As these are the same for every user, the mapping itself poses no problem. However, due to the relaxed WYSIWIS it is possible for a user's telepointer to reside outside another user's view, thereby decreasing workspace awareness. Another problem is that textual reference of objects can refer to different objects. For example, "the object at the top" can refer to two different objects for two different views.
Greenberg, Gutwin and Roseman describe two more relaxed-WYSIWIS styles [14]. In the first style the view size itself is customizable, for example by resizing the window. As they point out, a problem arises if the application reorders the objects to fit in the resized space. In this situation the world coordinates are no longer similar and cannot be used to map the telepointers. Mapping occurs by mapping pointer position to the underlying application objects. Though the mapping itself occurs correctly, telepointer movement may become erratic. Moving the pointer from the new line to the above line will be a fluent motion in the new window, but will seem jump from the end of the line to the beginning in the original window. Similar oddities will happen with gestures which span more than one line.

The second style makes matters more complex by allowing objects to alter the views. As an example consider an expandable tree, which expands on the view of the user who desires it, but not on all others. This may lead to a situation where someone is working on, talking about, and gesturing at some object which is not even visible to the others. Furthermore mapping becomes problematic, as the object to which the telepointer is mapped, may not be there in a different view.

Mouse pointers are often overloaded, providing information of the current state of the system, the user or help messages. For example, in windows when the system is busy, the user is made aware of this by transforming the pointer into an hourglass. This overloading can also be used in groupware to help provide understanding of what a user is doing, for instance by changing a cursor into a flywheel when a user is editing an object. Another important use of overloading telepointers is to provide awareness of the identity of a user. Examples include color coding, displaying a name or picture below the pointer, and changing the pointer into a symbol which represents the user.

Telepointers are a streaming media. A constant flow of x and y coordinates is required to achieve the necessary performance. A big enough disruption in the flow can cause stuttering or jumps. This not only looks bad, but is devastating for gestural communication. Three network effects can cause problems [11]:

- Latency, which is the delay between the sending and the receiving of a message. The user might not notice it, as all movement will be delayed equally (assuming no jitter). However, when a second media is used to support the telepointer, they will be out of synch.
- Jitter, which is the variance in transmission time. Messages arrive irregularly, some on time, some too late, due to congestion of the network. This will cause stuttery movement.
- Loss, which is complete loss of a message during the transmission. As losses often come in bursts, this will cause jumps.

According to Dyck, Gutwin, Subramanian, and Fedak [11], most of the currently existing telepointer implementations can only function on high-bandwidth networks, but fail on slower ones, which have far more problems with network performance. They have developed a telepointer which can function under bad circumstances. A number of techniques were used to improve performance, the most important ones being compression and adaptive rate control. Compression decreases the message size and therefore the amount of data to be sent, thus achieving better performance on slower networks. Adaptive rate control sends messages on a timer instead of using an event based system. The adaptation ensures that adjusting the send rate based on network conditions. Another interesting conclusion is the unsuitability of TCP/IP. Instead the more light-weight UDP protocol should be used.

The effects of jitter can be decreased by adding telepointer traces. A telepointer trace is a visual effect which provides feedback on the previous positions of the pointer. The visual effect must be carefully designed, lest it be distracting. Gutwin and Penner [16] discovered that "relatively short, low-contrast, fading motion lines showed motion well, but did not add undue clutter to the display". Adding a trace decreases many negative effects of jitter. By allowing people to customize the time to fade, contrast, etc. of the trace, users limit cluttering to an extent which they find acceptable.

Design: Telepointer

Telepointers are less customizable than an IM and userlist. Performance is an important aspect and the suggestions given above should be taken into account during its implementation. Due to a lack of time the Telepointer has not been fully designed. There are a number of issues which still require answering and some which have already been answered.

- How can the owner of the Telepointer be recognized?
- How will the Telepointer be mapped?
  - What will happen when a user edits an object?
  - What will happen when a user moves his pointer to a different layer?
  - What will happen when a user switches to a different study?

How can the owner of the Telepointer be recognized?

By providing names with the pointers, it should nearly always be clear to whom the specific pointer belongs. These named pointers should also be shown in the secondary view.

How will the Telepointer be mapped?

Since the VL-e will not reorder objects when a window is resized and will not have expandable items, world coordinates should allow proper telepointer mapping.

When a Telepointer moves to a different layer (e.g. a pop-up window, the drag and drop list of the Topology Editor) there are two possibilities. Either map the Telepointer to the different layer, or don't map it. In case of a pop-up window, mapping it to that layer will not work unless the pop-up is visible for every user. Showing a full-scale pop-up on every screen every time someone triggers one is obviously a bad idea. Not showing anything may be a bad idea as well as it may provide awareness. The best solution in case of a pop-up would be to show a miniature version of it on all other workspaces, located at the position of the Telepointer of the user who triggered it, and map the user's Telepointer to it. In case of the drag and drop list, the Telepointer leaves the shared workspace. Furthermore, as can be seen in Figure 3 on page 7, the position of the list is a separate window and its position is therefore customizable. Not mapping the Telepointer to that layer will either freeze it on the position where it was prior to switching to the other layer or removed it from the workspace entirely since the Telepointer is no longer on the

workspace. Mapping the Telepointer may cause erratic jumps, as the layer's position is customizable. Once again, providing a miniature on the place where the Telepointer changed to a different layer might be the best solution, as this will provide feedback but prevents erratic Telepointer movement.
When a user edits an object, he will be editing its properties. As can be seen in Figure 3 on page 7 these properties are located on the down-left side of the screen. This may be a problem, as altering these properties will require the mouse moving to that portion of the screen and then switch between layers. Not only are the same problems present as with the drag and drop list, but now the position of the Telepointer no longer properly reflects the user's action. A solution might be to visually freeze the Telepointer to an object while it has been selected, while letting the user move his mouse around freely. Unfortunately, this would require a user to deselect the object before his Telepointer becomes visible again. It is more likely that people will select an object, read its properties, navigate around the workspace and select a different object. Deselecting is an extra action people will generally not do.
When a user switches to a different study, the most obvious option would be to let the Telepointer simply disappear from the workspace. This should work fine in most cases. Color coding for objects which are locked may have to be provided though, as it may otherwise be unclear why a user may not alter an object even though no pointer is near it.

Different Telepointer

One option we are considering is to limit the amount of Telepointers available in a workspace. Instead of giving every user a Telepointer, one can be requested by clicking on a button. Once the user has received the pointer he can use it for a certain period of time or can release it beforehand. Furthermore only Telepointers are able to use the drag and drop list and to alter objects.
The advantage of this scheme is that it will lessen the chaos in the workspace. Data consistency management will become easier, as conflicts are less likely to occur. The downside is that awareness will drop as it is uncertain where other users are gazing. Furthermore the effectiveness is largely dependent on whether users will collaborate to discuss results or to build a study. If a lot has to be built, limiting the amount of Telepointers will slow down the progress. If results have to be discussed, limiting the amount of Telepointers will improve the discussion as users have to focus on fewer pointers.
By providing a customizable timer as well as a button-based release mechanism and by allowing the project leader to customize the amount of Telepointers the system should reflect the needs of any group. By making the release mechanism customizable, project leaders can make a tradeoff between starvation (someone refuses to push the release button) and annoyance (someone loses the telepointer in mid-discussion). By making the amount of Telepointers customizable the project leader can better customize his study.

Chapter 5 Toolkits

A huge number of collaborative toolkits exist which should improve the life of the implementer. In this chapter we shall first give a list with many of the toolkits we found on the internet. We shall give short descriptions of them, as given by the sites from which they were taken, and we shall give the status of the toolkits. We will conclude this chapter by explaining which toolkit we selected for our work and illuminate some of its advantages and disadvantages.

5.1 Toolkits

Toolkit Name & Info URL	Short Description (taken from the respective sites)	Status
COAST www.opencoast.org	The COAST framework reduces the load of development of groupware applications to a level of single user application. The COAST framework offers data description mechanisms to specify the (shared) domain model, provides mechanisms for synchronous manipulation of the shared domain model by a set of users, keeps the shared model and their visualizations at a consistent state, supports the provision of clues about activities performed by other users (group awareness), includes a pre-defined extensible model of users and their work environments, and assists the developers in modeling user interaction and collaborative sessions.	Last Update: 2003
Collabrary grouplab.cpsc.ucalgary.ca/ collabrary/	The Collabrary is a library of COM objects for rapidly prototyping collaborative multimedia applications. Example applications include: media spaces, synchronous real-time groupware, community information spaces, computer vision applications, scripted image and video manipulation.	Alive
DistView www.eecs.umich.edu/distview	DistView is a prototype multicast middleware service for building collaborative applications. Written entirely in the Java programming language, DistView provides group communication services that meet the various shared state management needs of collaborative environments. It provides a rich interface for group and session management, the ability to ensure totally ordered message delivery, a lock-based distributed synchronization mechanism, and support for selective window sharing.	Alive

Collaborative Toolkit For Diverse www.sv.vt.edu/future/cave/ software/D_collabtools	There are two main parts to this collaborative toolkit. The first part is providing avatar support to your virtual world. The second part is item manipulation in the virtual world. Diverse is a cross-platform, open source, API for developing virtual reality applications that can run almost anywhere. The goal of Diverse is to enable developers to quickly build applications that will run on the desktop as well as various immersive systems.	Alive
DreamTeam dreamteam.fernuni-hagen.de/ dreamteam/	DreamTeam is an environment for developing, synchronous shared applications. The DreamTeam environment allows the developer co-operative applications like single user applications, without struggling with network details, synchronization algorithms etc. A DreamTeam add-on, called DreamObjects, simplifies the management of shared data. It extends the DreamTeam runtime and development environment and offers a variety of shared data services. DreamTeam is entirely written in Java, thus runnable on many systems	Alive
Egret www2.ics.hawaii.edu/ftp/ pub/csdl/egret	Egret is a groupwork environment that defines both a data and a process model along with supporting analysis techniques for exploratory collaboration, such as software development and document generation. Egret is an Emac-based toolkit which provides both low and high level storage and communication facilities for the development of (primarily textual) cooperative work applications.	Alive
GroupKit www.groupkit.org	With GroupKit programmers build applications for real-time, distributed computer-based conferencing. Groupkit is based on the Berkeley's Tcl/Tk language. GroupKit facilities include shared data structures, flexible session management, remote procedure calls, concurrency controls and multi-user shared interfaces. It is possible to integrate computer-based media space systems with GroupKit, so that starting a GroupKit conference also starts the media-space system.	Alive

Habanero sunsite.bilkent.edu.tr/pub/SDG/Software/Habanero/	Habanero is a collaborative framework and set of applications. Using Habanero you can create and work in shared applications from remote locations over the Internet. The Habanero framework, or API, enables developers of groupware applications to build powerful collaborative software in a reduced amount of time. The Habanero framework provides the necessary methods developers can use to create or convert existing applications into collaborative applications. Habanero is written in Java, it will run under any operating system that supports JDK 1.1.6. The Habanero environment consists of a client, a server and a variety of tools.	Last Update: 1998
Jabber www.jabber.org	Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in close to real time. The first Jabber application is an instant messaging (IM) network that offers functionality similar to legacy IM services such as AIM, ICQ, MSN, and Yahoo.	Alive
Java Shared Data Toolkit java.sun.com/products/java-media/jsdt/	The Java Shared Data Toolkit software is a development library that allows developers to easily add collaboration features to applets and applications written in the Java programming language. Enterprise developers can use the Java Shared Data Toolkit software to create network-centric applications, such as shared whiteboards or chat environments. It can also be used for remote presentations, shared simulations, and to easily distribute data for enhanced group workflow. The Java Shared Data Toolkit complements Sun's rich suite of Java multimedia technologies by allowing them to be incorporated into "sessions" created and managed with the Java Shared Data Toolkit.	Alive
MAUI (MUG) hci.usask.ca/projects/maui.xml	This toolkit provides the first ever set of UI widgets that are truly collaborative-aware. The MAUI toolkit includes groupware versions of standard UI widgets, and also provides specialized groupware components such as telepointers and participant lists.	Alive

OVAL www.ickn.org/elements/hyper/ cyb64.htm	Oval uses ideas from artificial intelligence and user interface design to represent information in a way that can be processed intelligently both by human beings and by computational agents. It integrates ideas from the fields of hypertext, object-oriented databases, electronic messaging and rule-based intelligent agents. Its particular focus is on the development of applications for cooperative work, but most of the ideas can easily be applied to other domains.	Unknown
PREP eserver.org/software/prep/	PREP is a Macintosh application that encourages and facilitates collaboration in writing. PREP provides a column-based interface where related information is linked across columns. One use for PREP is as a word processor that provides more than print margins -- it provides an unlimited number of "virtual margins" that collaborators can write in. PREP supports text, drawings and voice within the document as content or annotation.	Dead
Prospero (Neuman) www.isi.edu/gost/gost-group /products/prospero/	Prospero is a collection of protocols and embedded software providing distributed directory services, file access services, naming, maintenance of attributes, indexing, caching, storage, for other network applications.	Last Update: 1998
Prospero (Dourish) www.cs.ucl.ac.uk/external/ p.dourish/thesis.html	Prospero embodies a model of its own behavior (a reflective model) which is causally-connected to the behavior it describes. So the model can be used, not only as a way for the system to introspect and reason about its own configuration and action, but as a way for it to reach in to the implementational structures which support it, and modify them. The application developer gains control over the strategies used for data distribution, conflict resolution and interface linkage. These are areas where the toolkit developer would normally have to make implementation decisions which would limit the applicability of the toolkit and impact patterns of collaborative interaction for end-users. The open implementation approach used in Prospero allows these decisions to be revised later, when the needs of particular application areas or groups become clear.	Dead

Single Display Groupware Toolkit www..cpsc.ucalgary.ca/grouplab/software/SDGT/	The SDG Toolkit is a framework for designing single display groupware applications. SDGT is a COM object designed for managing multiple mice and keyboards.	Alive
Sonexis Audio and Web Conferencing www.sonexis.com	Sonexis, Inc. delivers an in-house, secure, integrated audio and web conferencing system that helps businesses improve business processes and communications while significantly reducing the cost of conferencing.	Alive
Suite ftp.cs.unc.edu/pub/users/dewan/suite	Suite Distributed Object Model is a Unix/X groupware toolkit that can be used to extend the original Suite system and offers a better support for building new classes of applications in distributed fashion. It has been used to investigate the issues of sharing abstractions, flexible coupling, access control, merging and inheritance. Suite is one example that indicates the benefit of the distributed objects approach.	Last Update: 1998
TeamWave Workplace www.markroseman.com/teamwave/	In real life, teams use rooms to discuss matters, brainstorm ideas, scribble diagrams on a whiteboard etc. They also use rooms to store/display things (draft reports, drawings, references etc.) for other members to look at/modify/add to at any time. TeamWave Workplace is a cross platform Internet-based software package which allows these sorts of collaborations to take place over the Internet in a set of virtual meeting rooms. Not only does this mean that geographically isolated teams can work together but also groups of school students and/or teachers can use the virtual rooms as places to permanently store shared resources.	Dead
Virtual Network Computing www.realvnc.com	VNC is remote control software which allows you to view and interact with one computer (the "server") using a simple program (the "viewer") on another computer anywhere on the Internet. The two computers don't even have to be the same type, so for example you can use VNC to view an office Linux machine on your Windows PC at home.	Alive

5.2 Toolkit for VL-e

Many of the toolkits described above have a variety of problems which make them unsuitable for our project. For one, some of them force the designers to use a specific architecture (e.g. COAST, DistView). Since the VL-e itself has specific architectural requirements, using such a toolkit will give an awkward end-product with an architecture inside an architecture. Furthermore it is uncertain whether it is actually possible to use such a toolkit for the VL-e. A telepointer mechanism may well be incompatible with the workspace which already exists in the virtual lab. Another problem with using a tailored toolkit is that the components often cannot be changed. Other toolkits are too limited to suit our needs (e.g. VNC), simply unsuitable for our intentions (e.g. Suite), or dead (e.g. TeamWave).

We selected the Java Shared Data Toolkit to aid in our work. It has many advantages, the most important one being its high customizability. Unlike many of the other toolkits which provide entire components, JSDT provides small building blocks to create a component. Furthermore it uses Java, which is what most of the VL is written in. By using Java functionality provided by the Globus toolkit can be directly used, as the Globus has an API for Java. Furthermore, the JSDT is alive, so it is still being improved.

JSDT: A Short Overview

The choice of the toolkit will have major influence on the design. Therefore it is necessary to understand some of the functionality it provides [7].

Session

In the JSDT the initial meeting place is called a Session. A Session has a specific URL of the form:

jsdt://<server>:<port>/<protocol>/Session/<name>

Here, <server> is the name or IP address of the server computer, <port> is the port number, <protocol> can be socket, http, or lrmp, and <name> is the name of the Session. A Factory is provided which can be used to create, join and destroy a Session. The first invocation of the create method of the SessionFactory while providing a URL string causes the Session to be created. Preceding calls with the same URL will give a reference to the existing Session.

Client

Any object that is going to participate in a Session has to implement the Client interface. One important method of the Client is its authentication method. This method is used for authentication purposes when the client tries to perform a privileged operation like, for example, joining a Session. A special kind of Client can be created by using the ClientFactory. This Factory assigns the Client with a URL, which can be used for invitation purposes. A Client object can be the source and destination of the data which is being exchanged.

Channel

A Channel is a communications path between Clients within a Session. Clients can add Consumers to a Channel to receive data from it. Clients can send data over a Channel to all Clients including themselves, excluding themselves, or to a single Client. The data sent over a Channel is a specific object, the Data object. It includes the data, a priority, the name of the sender and the Channel over which the data was sent.

ByteArray

The JSDT allows the creation of a shared object that is permanently available to Clients within a Session. This object is a ByteArray and it can contain an array of bytes, a String object, or a serializable Java object. Clients joining a ByteArray will be notified when someone changes the object.

Manager

Access to Sessions, ByteArrays, Channels and Tokens can be controlled by adding a Manager to it at creation time. The Manager issues a challenge to the Client who is trying a privileged operation. The Client responds to the challenge and the Manager compares the response with the response it was expecting.

Using the JSDT will allow us to create a Collaborative environment. As it works with Session URLs, these Sessions can be located anywhere. By giving the Client a URL as well, it is possible to use invitation mechanisms. Communication between Clients will have to take place via a server. It is possible to create a Session on the Client side, but that will not provide a usable solution. If the Client who created the Session decides to turn off his PC, the Session and everything with it (Channels, ByteArrays, etc.) will go down with it. Obviously that is a bad idea, as this will break off any ongoing conversation. Solutions which do not have these problems include using predefined channels over which users can communicate and requesting the server to create a channel. Since both solutions have the channels located on server-side, the user’s availability is of no concern.

Chapter 6 Architecture

In this chapter we will use the requirements defined in Chapter 4 to design our system by using parts of the UML as described by [3] and [10]. We will present the architecture which supports the Instant Messenger and userlist. As stated before, the Telepointer's design has been postponed due to a lack of time.
It is important to note, that the development process itself did not take place in the same order as the items are presented here. Nor were they created as linearly.

6.1 Designing the Architecture

Requirements Revised

Besides the truly collaborative requirements described previously, the VL-e has additional requirements, which influence the system design.

- Grid Service: as stated in Chapter 2, the VL-e is service oriented. This influences our design by forcing a client-server architecture. Furthermore services have to remain up and running, requiring persistence.
- Light-weight client: most of the processing has to be done at the server side.
- Linking of studies: this links collaborative environments together. It opens up a new range of possibilities, such as communication outside of a study, inviting someone to join, etc.
- Security: only people with access to a study are allowed to communicate over a channel dedicated to the study.

If users from different studies are to communicate, the means has to be provided to allow them to find each other. The Global Collaborative Manager (GM) is a Grid service which connects the individual studies. Per VL-e setup there is only one instance of this service.
In order to create the light-weight client, nearly all processing will have to be done at the server side. The Grid Service responsible for this is the Cross-Session Collaborative Manager (CSM). Each client simply communicates with his CSM, who acts on his behalf on the server side. All the client-side has to do is provide a GUI.
For communications to take place between two people, the above allows for two options. Users can either communicate directly from one CSM to another CSM, or they can communicate via the GM. As described in Chapter 5, the first one is not an option, due to problems when the creator of a channel decides to leave. The second approach has other problems. All communications between users would have to go via the central server. On slower networks with many users (and perhaps in the future streaming media), performance will probably suffer greatly. Furthermore, the Collaborative System will be greatly dependent on the GM. If it goes down, communications are dead. To solve these and other problems, we decided on using a more

distributed approach. A third Grid service, the Inner-Session Collaborative Manager (ISM), will give each session its own collaborative environment. Users join channels made available by the study's ISM. The ISM knows how to contact the GM if a message needs to go outside a session. As it is to be expected that nearly all communications will take place inside a session and very little outside a session, using the distributed approach will decrease network load. Also, if the GM goes down, users will simply be unable to send messages outside their running sessions. A disadvantage of the distributed approach is that it is more complicated.

In conclusion, our system will have four main components: the Client, the Global Manager, the Inner-Session Manager and the Cross-Session Manager.

Use Case Diagram

Use case diagrams help to determine the functionalities a system must offer to the outside world. For this it is necessary to know who will interact with our system; the actors. Actors can include persons, but also other components, like databases. For our system we found the following actors.

- User: a person who uses the Collaborative System.
- Session Manager: the VL component responsible for the creation of a collaborative session.
- Accesslist Provider: since it is currently unclear which actual VL component will provide the access list, which we need to build the userlists, we will use this actor for it.
- VIMCO: the VL component which can provide us with a collaborative level for a user.

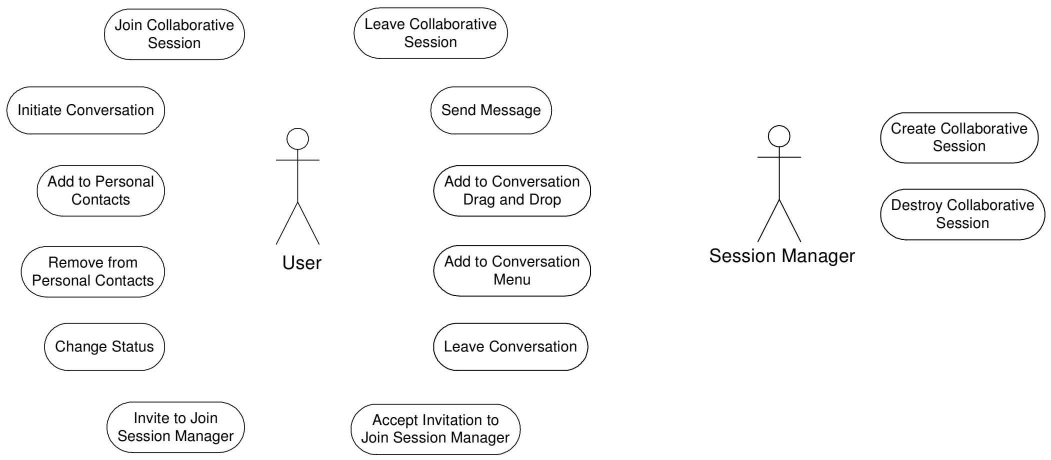

Join Collaborative Session
Leave Collaborative Session
Initiate Conversation
Send Message
Create Collaborative Session
Add to Personal Contacts
Add to Conversation Drag and Drop
Destroy Collaborative Session
Remove from Personal Contacts
User
Add to Conversation Menu
Session Manager
Change Status
Leave Conversation
Invite to Join Session Manager
Accept Invitation to Join Session Manager

Figure 6: Use Case Diagram

In the first set of use cases the Collaborative System will interact with the User, the Session Manager and the Accesslist Provider. Using the requirements defined previously use cases (Appendix A) and a use case diagram (Figure 6) were developed.

Conceptual Model

The conceptual model gives better insight on how the objects that emerge from the use cases are related. As it is part of the analysis phase, implementation issues such as GUIs, network communication, etc. are not represented.

In the use case “Create Collaborative Session”, a short description is given of how a collaborative session is created. It is created by the Session Manager and only one will be created per study. The object which is created matches the Inner-Session Manager. From the use cases we can thus already determine some of the connections that the ISM is likely to have with other components: the ISM will at least communicate with the Client, the SM and the Authenticator. Furthermore, we already know that the ISM needs to connect to the Global Manager to be able to send messages across sessions. Since the Client’s representative is the Cross-Session Manager, the ISM will have to communicate with the CSM as well. The GM will only communicate with ISMs so there are no further connections to represent for it.

Where the CSM is concerned, we already know that it will have to provide functionality for the Userlist and MessageWindow, but we have not yet included these connections in the schema as they need to be analyzed further; the class the user will interact with is not the same as the one the CSM will interact with.

This leads to the conceptual model in Figure 7.

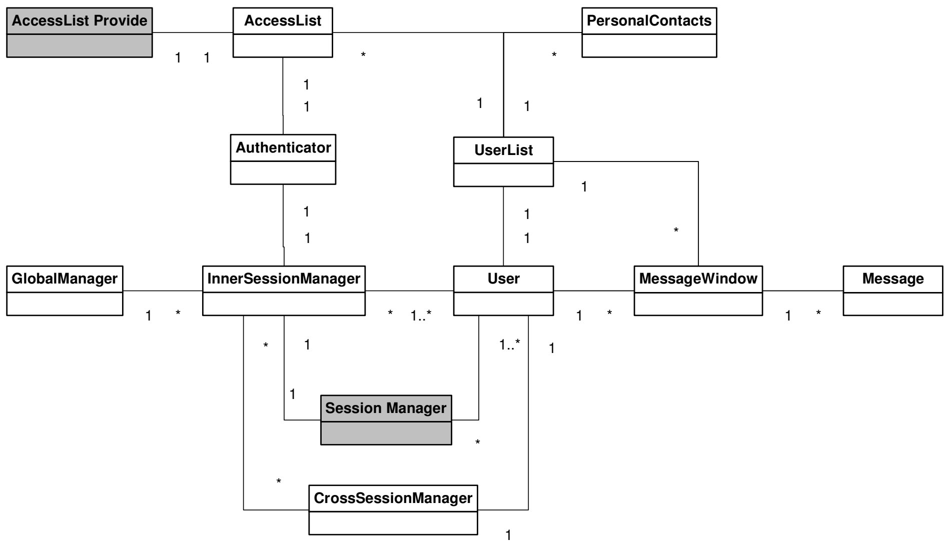


Figure 7: Conceptual Model

Finding Classes

Client Side

From the conceptual model we can derive several classes which will be located on the client side. The resulting architecture is shown in Figure 8.

- User: responsible for information about the user
- MessageWindow: the window in which to type messages
- UserList: the list containing the contacts of this user

Using the Control pattern from the “General Responsibility Assignment Software Patterns” (GRASP) [3], we learn that we should decouple the GUI from its functionalities. This implies that both the MessageWindow and UserList class should be split into two separate classes.

- MessageWindowHandler: ensures that messages are displayed in the right GUI
- MessageWindowGUI: the graphical representation of the message window
- UserListHandler: the functional part of the userlist
- UserListGUI: the graphical part of the userlist

Each user coming from an accesslist has a name and a state which will have to be represented on the userlist. Usernames can appear more than once on a userlist, simply because users can be in more than one study together. To make this easily manageable and to have to store a user only once per userlist on the userside a class should be created to keep track of users and their information.

- UserInfo: stores user information, like a user’s state and his name

Since most of the actual work will be done by the CSM, we need some way to communicate with this service. We will also need to be able communicate with the ISM.

- CommunicationsHandler: responsible for communications with the CSM
- UserManager: communication with ISMs

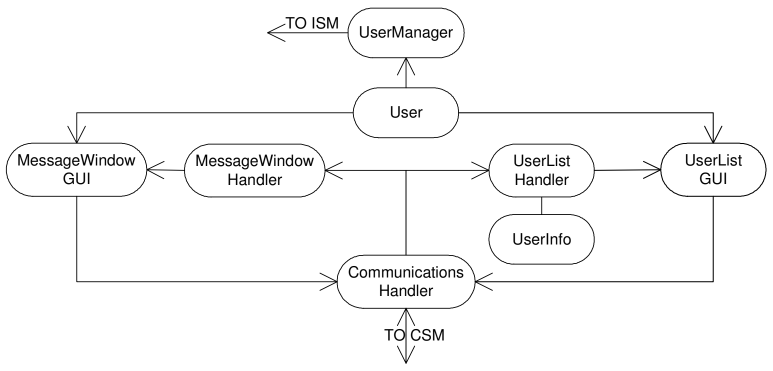


Figure 8: Architecture Client Side

Server-Side

Cross-Session Collaborative Manager

Some of the classes of the CSM are easily located, as they are reflecting the client side. Another class easily found is the actual service.

- Cross-Session Collaborative Manager: the service, simply waiting for incoming requests, which it forwards to the other classes. Also sends messages back to the client (through notifications).
- MessageWindow: provides the functionality for the message window. For example, when a user receives a message, this class ensures that the client side receives the proper information to display the message in the right window.
- UserList: provides the functionality for the userlist. For example, when a user clicks on a user's name, this is the class responsible for finding out whether a new message window has to be opened, or whether one already exists.

As can be seen in these last two classes, some administration has to be provided for the message windows. Both UserList and MessageWindow have a need for such a class. Furthermore text messages as well as userlist messages (messages concerning the change in state) are necessary.

- MessageWindowAdministration: responsible for keeping track of the message windows; which users are part of them, where these people are located, over which Channel to send the message, the ID of the window, etc.
- TextMessage: is the text-message to be sent.
- UserListMessage: userlist-message, concerning a change in state.

Since the JSDT is used to communicate between users, some other classes are necessary which allow the sending and receiving of messages. To reduce coupling between components with and those without JSDT a specific class between them was created.

- SenderReceiver: responsible for setting up the JSDT environment. Creates channels, listeners, etc.
- MessageSender: sends messages over channels
- MessageReceiver: receives messages from channels by adding consumers to them
- MessageHandler: decides the actions to take with incoming messages and outgoing. Reduces coupling.

As stated in Chapter 5, any object that is going to participate in a Session has to implement the Client interface. Furthermore for a user to be able to be invited to a JSDT session and channel it is required that the client is a specific kind of client: one with a URL.

- ClientCrossSession: special kind of client, required to allow for invitations.

In Chapter 4 it was mentioned that a possible way to provide users with varying availability modes, would be to provide them a collaborative level. This level would have to be stored in a database. Accessing this database gives us another class.

- VIMCOHandler: accesses the database where the collaborative levels are located

Finally, it turned out that the messages arriving via channels arrive faster than the CSM can forward them to the client. The solution is introducing buffers at the server side.

- MessageWindowBuffer: buffer for incoming text-messages

- UserListBuffer: buffer for incoming user-list messages

With all these components we get the architecture shown in Figure 9.

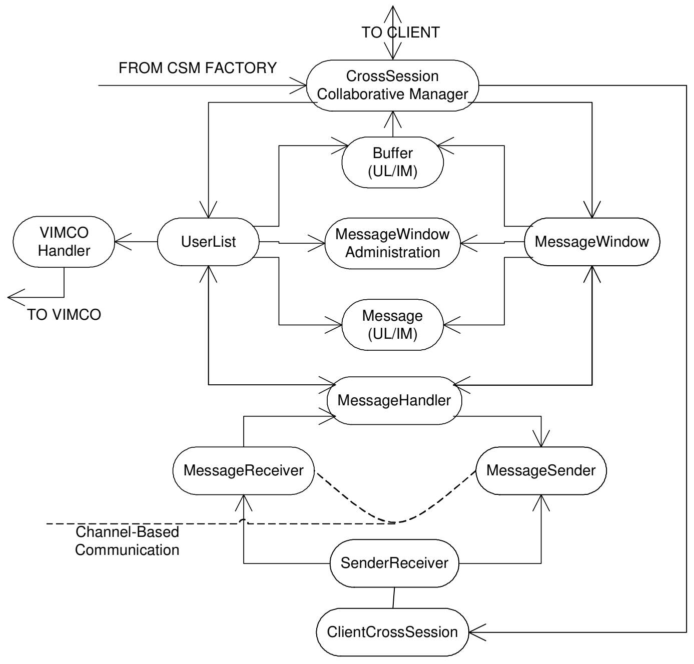


Figure 9: Architecture Cross-Session Collaborative Manager

Global Collaborative Manager

Like the CSM, the GM needs classes which allow the JSDT to function, as well as a service.

- Global Collaborative Manager: the service, waiting for incoming requests
- SenderReceiver / MessageSender / MessageReceiver / MessageHandler are where functions are concerned similar to the ones described above.
- Client is a minimal client, required for the JSDT

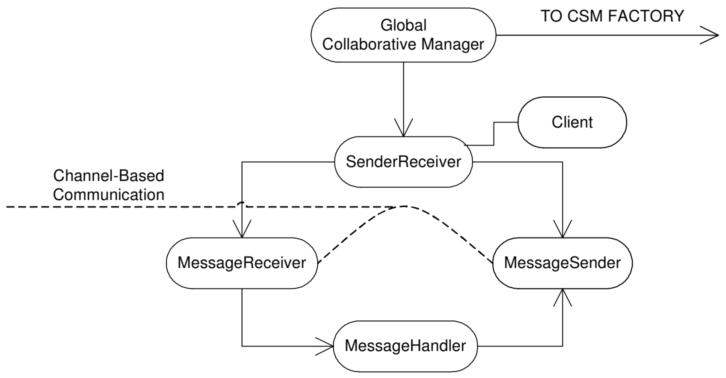


Figure 10: Architecture Global Collaborative Manager

At this time there are no other classes which need to be described here. Later versions may require other classes, for example if we would want to store offline messages. The GM's architecture can be seen in Figure 10.

Inner-Session Collaborative Manager

Once again some classes are required which allow the JSDT to function and an actual service is required.

- Inner-Session Collaborative Manager: the service
- SenderReceiver / MessageSender / MessageReceiver / MessageHandler are similar to the ones described before
- Client again is a simple client to allow the JSDT to function properly

The conceptual model reveals some other classes.

- Authenticator: access control for the ISM. Actually, as stated in Chapter xxx, the JSDT allows us to create three types (Session, Channel and ByteArray), but for the architecture we will use Authenticator.
- AccessList: contains the list with the users allowed to access the session

This results in the architecture of Figure 11.

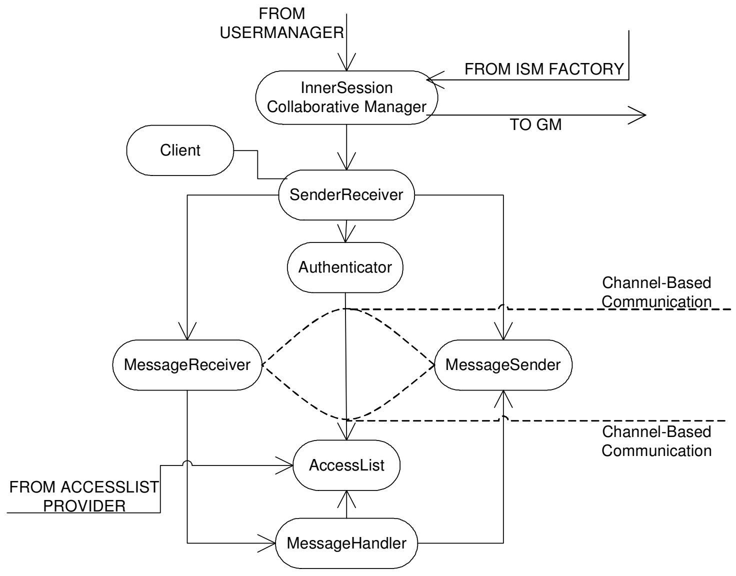


Figure 11: Architecture Inner-Session Collaborative Manager

Combining the architectures of Figures 8, 9, 10 and 11 and adding the factories for the services, gives us the architecture shown in Figure 12.

Figure 12: Architecture Complete

6.2 Using the Architecture

With the architecture in place, we shall now show how several of the more interesting actions can take place. Not all the details are shown.

Create Session

The creation of a session is initiated by the Session Manager. As can be seen in Figure 13, the environment is set up, creating the necessities for the collaborative session. First, the SM contacts the ISM Factory (1), which spawns a new instance (2). The SenderReceiver is created (3), which creates the Accesslist (4), the Authenticator (6), the MessageSender (7), the MessageReceiver and the MessageHandler (9). The Accesslist itself is retrieved from the Accesslist provider. Finally the SenderReceiver joins the Global Manager (10).

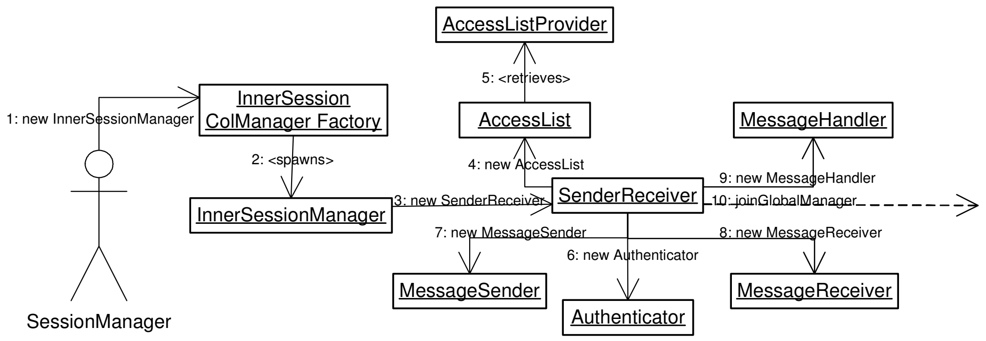


Figure 13: Create Session

Get CSM Reference

In order to be able to contact a Service, either a URL or a Grid Service Handle to it is required. Therefore if a user is to communicate with his Cross-Session Manager he has to acquire such a reference. Figure 14 shows the current way to obtain it. The UserManager contacts the ISM and requests a CSM reference (1). The ISM contacts the GM and request a CSM reference (2). The GM contacts the CSM Factory (3), which spawns a new CSM (4). The reference is returned to the user.

As can be seen the CSM is created via the ISM and GM. The GM creates the CSM, but since users are not allowed to access the GM, obtaining the reference has to go via the ISM.

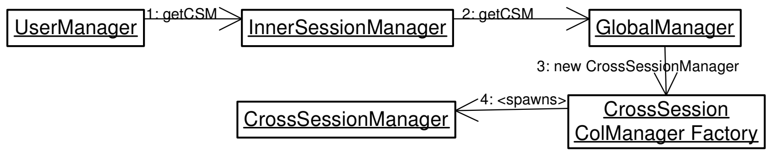


Figure 14: Get CrossSessionManager Reference

Join Session

This is more complex. When a user wants to join the Session Manager, he has to be able to locate it (cf. the above). For now, let’s assume the user already has a reference to the SM. He contacts this SM and requests the reference to this study’s ISM. This reference can now be used to communicate with the ISM. In order to actually join the ISM, the client’s CSM has to join the ISM.
As can be seen in Figure 15, the UserManager thus provides the ISM with the ClientCrossSession URL (1). The ISM then asks the SenderReceiver to invite the CrossSessionClient located at that URL (2). The CrossSession Client receives the invitation (3) and calls one of the (CrossSession) SenderReceiver’s functions to notify him of the invitation (4), as the SenderReceiver is responsible for setting up the connections. The SenderReceiver tries to join (5) and is authenticated by the Authenticator (6), located at the ISM. Once he is authenticated he continues the rest of the setup (7, 8, 9).

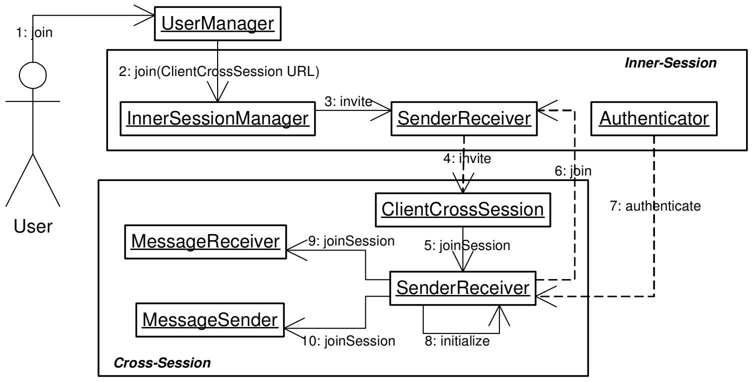


Figure 15: Join Session

Send Message
As can be seen in Figure 16, sending a message goes via a lot of steps. First the user types the message and pushes send (1). The GUI then gives the message to the CommunicationsHandler (2) which sends the message to the Cross-Session Manager (3). The functionality offered by the message window is handled by the MessageWindow class, so the CSM sends the message to this class (6). At this point we require additional information about this conversation. We know the originating WindowID, which is sent by the Client side along with the text to send, but do not know who are part of this conversation. Nor do we know over which channel we have to send the message. Such information is in the MessageWindowAdminstration. We retrieve the necessary information (5). Next we create the TextMessage (6) and give it to the MessageHandler (7). The MessageHandler transforms it into a Data object, necessary for JSDT, and tells the MessageSender to send the message (8). The message is sent over the correct Channel.
The MessageReceiver’s consumer receives the incoming data and sends it on to the MessageHandler (9). The MessageHandler transforms the data back to a message and after

determining it is a TextMessage, sends it on to the MessageWindow (10). The MessageWindow checks the MessageWindowAdministration whether this conversation already exists and if it doesn't the administration is created (11). Next, the message is added to the MessageWindowBuffer (12). From the buffer it is sent to the CSM, which in turn notifies the CommunicationsHandler and sends it on to the MessageWindowHandler (13, 14, 15). This class checks whether a new GUI has to be created and does so if necessary. The message is given to the appropriate GUI (16), which displays the message to the user (17).

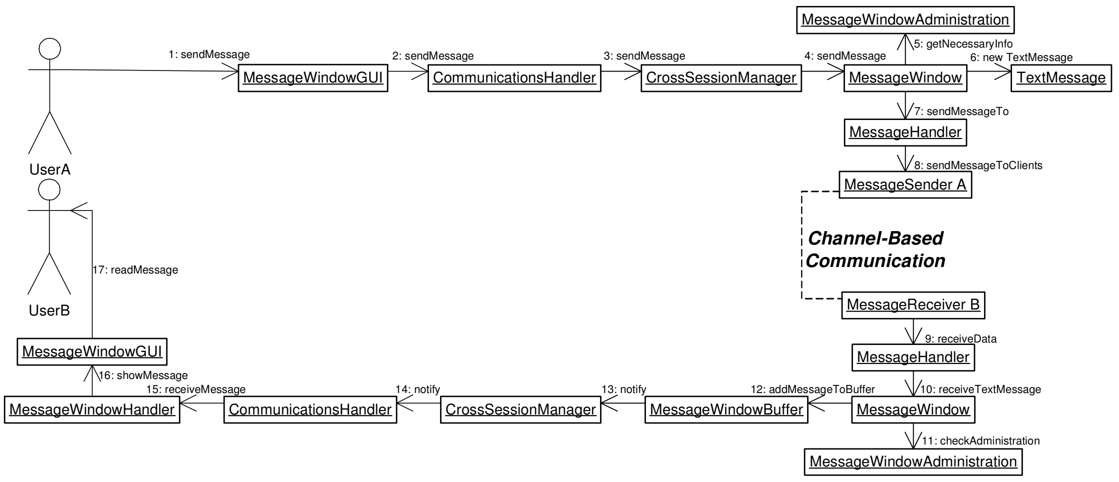


Figure 16: Send Message

Invite
We will not provide a diagram for invitations, as it hasn't been thought through yet properly. As its principle is quite easy to understand we will provide a sketchy description.
User A right-clicks on User B's name in the userlist. A pop-up window opens and UserA selects "invite user to join session…". Next he is given a choice to which session User B should be invited. User A makes the choice and the invitation is sent to User B, who can either accept or decline the invitation. He accepts and User A provides him with a reference to the Session Manager.
From here joining the Session Manager and joining the session ISM occurs normally.

Other interesting things
In Chapter 4 we stated that a user should join all his studies as soon as he logs into one. This should be possible without too much trouble. It simply requires a broadcast via the GM that the user has joined, after which the user's other ISMs can invite him.
As we stated in the beginning of this Chapter, one of the main purposes of using the ISMs is that communication will nearly never need to go via the GM. The GM is only needed for communication purposes when

a) A user logs into his first ISM. Besides getting him invited, it will make other users aware of the user's availability.
b) A user tries to contact a user who is not on his userlist or on his Personal Userlist. All that is required is to obtain the user's ClientCrossSession URL. As soon as that is available, the user can simply be invited to join a specific channel, thus not requiring the use of the GM. Even if the URL has to be received from the GM (at this time this is unclear), the accesses to the GM will be few.

Another interesting aspect is that users from completely separate VL-e distributions can be invited to join. All that is required is the ClientCrossSession URL.

Chapter 7 Results

At present part of the system has indeed been implemented successfully. In this chapter we shall describe what has been implemented, give some results and describe some of the problems encountered.

7.1 What has been implemented?

As most functionality is offered by the CSM, most of the work has been done there. Nearly all of the components at the CSM side have been implemented to some extent. The ISM is less complete. Not all of the components have been implemented, as not all were required yet. The invitation mechanism has been implemented and it is capable of inviting users to join its session if the user's JSDT URL is provided. The GM merely sets up the environment to allow ISMs to join. Similar to the ISM, some of the components still have to be implemented.
As has been noted in the previous chapter there is need for the Session Manager as it is responsible for creating the ISM. Since no operational Grid Service Session Manager currently exists, we created one which suits our needs. Furthermore the accesslist of a study at this point is statically created; it is simply a list of names.

The Collaborative System currently allows

- the SM to create an ISM (cf. Figure 13 in Chapter 6)
- the user to obtain his CSM reference (cf. Figure 14 in Chapter 6)
- the user to log into a ISM (cf. Figure 15 in Chapter 6)
- the user to send a message to another user (cf. Figure 16 in Chapter 6)
- the user to change his availability (very similar to sending a message)

And does not (yet) allow

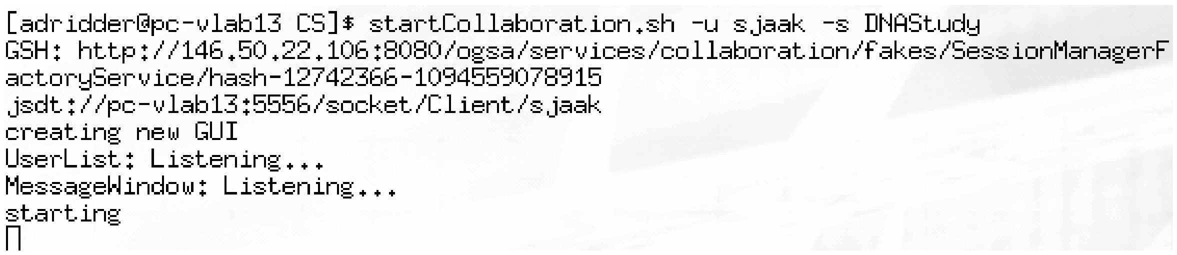
- authentication
- the user to log into multiple sessions
- the user to communication outside of a Session
- the user to have a personal userlist
- the user to invite another user to join a Session
- multi-user chat
- the user to close a message window and reopen it to the same user

Obviously, the basis has been laid, but there is still a lot of work to be done. Some of the items require little work, others are harder. The last point, for example, simply requires an update to the MessageWindowAdminstration. The first point, authentication, is harder, though the mechanisms have already been included.

7.2 Results

The best way to give an impression of the system is to provide screenshots belonging to the use cases for which the system was developed.

Test Case	New Session
Initial Operation	Create new Session Manager
Will Result in	User joining the Session
Remark	This is not one of the use cases as it appears in the Appendix, but a combination of two, namely Create Collaborative Session (invoked by the SM, which is invoked by the User) and Join Collaborative Session Manager (invoked by the User). Furthermore not all of the actions described by the use case take place. Since users currently cannot log into more than one session, checking whether the user is logged into more than one session does not happen.



```
[adridder@pc-vlab13 CS]$ startCollaboration.sh -u sjaak -s DNAStudy
GSH: http://146.50.22.106:8080/ogsa/services/collaboration/fakes/SessionManagerF
actoryService/hash-12742366-1094559078915
jsdt://pc-vlab13:5556/socket/Client/sjaak
creating new GUI
UserList: Listening...
MessageWindow: Listening...
starting
```

Figure 17: Sjaak Creats New Session

The user, in this case Sjaak, creates a new session, DNAStudy. As can be seen in Figure 17, the CS prints the Grid Service Handle of the Session Manager that has just been created by the user. The second output line is the User's JSDT ClientCrossSession URL, which is located at the CSM and is used for invitations by the ISM. The third line shows that a GUI is being started, which is the userlist. This occurs after the user has (automatically) provided his JSDT client handle to the ISM. As stated before, the accesslist is provided by the system. As soon as the CSM has successfully joined the ISM, the accesslist is received and transferred to the Client side. As can be seen in Figure 18, the user has indeed joined the session. No other users are currently online.

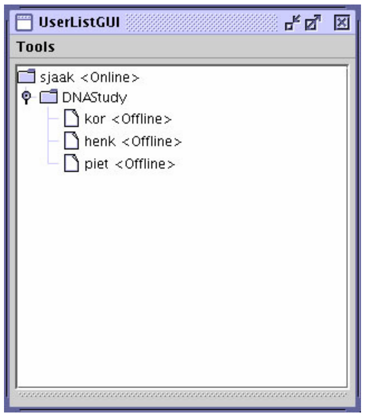
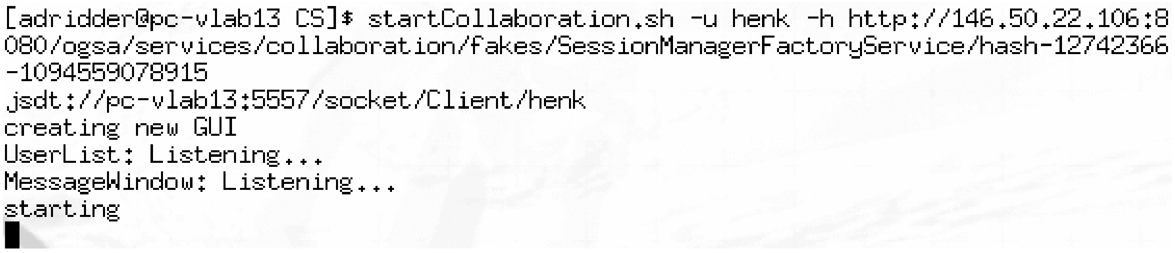


Figure 18: UserListGUI, Sjaak has joined

Test Case	User Joins
Initial Operation	Join Session Manager
Will Result in	User Joining the Session
Remark	This is similar to the Join Collaborative Session from the previous test case, but with a different initial operation. This time the user uses the reference to the SM to receive the reference to the ISM. He then tries to join the ISM by providing his JSDT URL.



```
[adridder@pc-vlab13 CS]$ startCollaboration.sh -u henk -h http://146.50.22.106:8
080/ogsa/services/collaboration/fakes/SessionManagerFactoryService/hash-12742366
-1094559078915
jsdt://pc-vlab13:5557/socket/Client/henk
creating new GUI
UserList: Listening...
MessageWindow: Listening...
starting
```

Figure 19: Henk Joins

The user, in this case Henk, has received the reference to the Session Manager he wants to join. He uses this to join the Session Manager (Figure 19). Similarly to the previous test case, the userlist is started and after a successful join, the accesslist is added to the userlist. As can be seen in Figures 20 and 21, both users are now aware of each other.

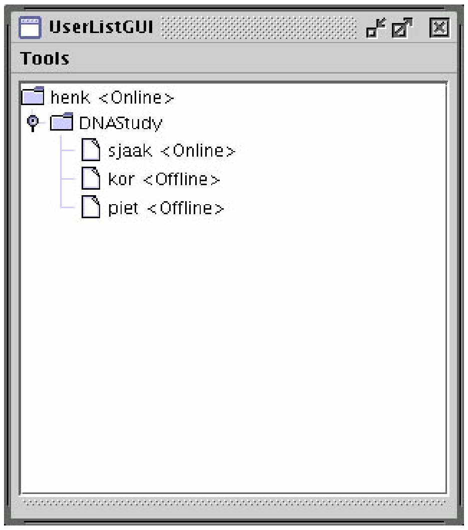


Figure 20: Henk sees Sjaak as online

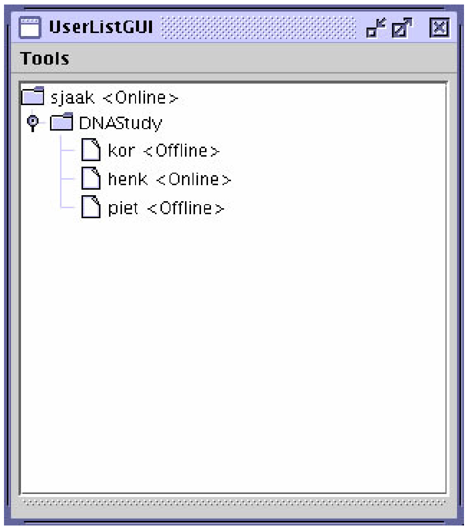
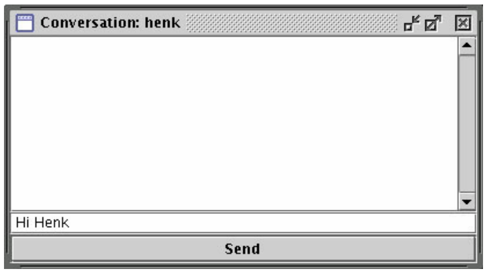


Figure 21: Sjaak sees Henk as online

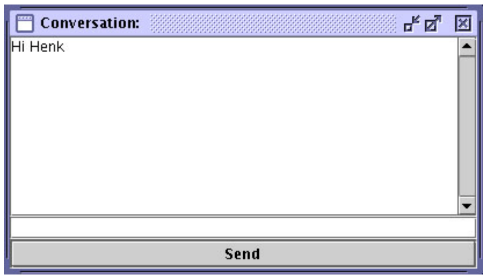
Test Case	Send a Message
Initial Operation	Click on user name
Will result in	1) Opening a new message window on the receiver side
	2) Displaying the message in the corresponding message window
Remark	This belongs to the use cases Initiate Conversation and Send Message

If one of the users, in this case Sjaak, clicks on the username of the other user, a message window opens, as can be seen in Figure 22. Sjaak can type a message and send it to Henk. Henk receives the message in either a new window or in the old one (Figure 23).



Hi Henk

Figure 22: Sjaak Types Message



Hi Henk

Figure 23: Henk Receives Message

Test Case	Change Status
Initial Operation	Select the status in the menu
Will result in	All userlist updating the status
Remark	This belongs to the Change Status use case.

One of the users, in this case Sjaak, changes his status to away. He can do this by using the menu, as can be seen in Figure 24. His status is changed accordingly and broadcasted to the other users (Figure 25).

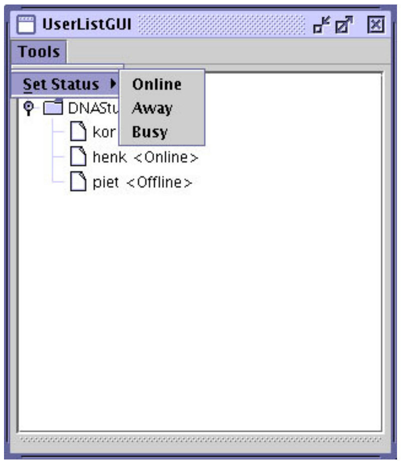


Figure 24: Sjaak Changes Status

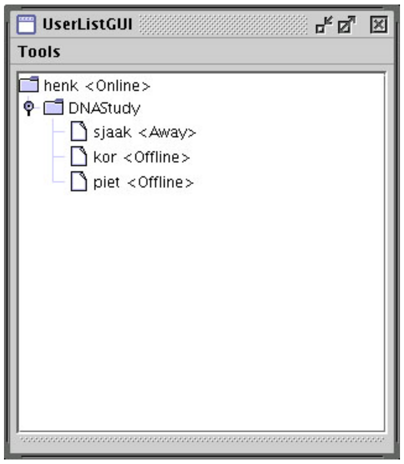


Figure 25: UserLists are Updated

7.3 Problems encountered

During the implementation several problems were encountered and two of them are worth mentioning.

Message lost without error

This was a rather weird problem. When a new user joins a new session, references are created at the CSM side. Amongst those, references to channels which belong to the particular session are created and, to be able to receive messages from these channels, consumers are added to them. When a new consumer is added to a channel, all users already on the channel are immediately aware of this. We had them respond by immediately sending their status to the new user. Unfortunately the user was unable to receive the messages, even though no error occurred. More interestingly, if we added a delay of perhaps one millisecond before sending their status, everything went fine. We have no clear idea why the error occurred though perhaps it happened because the Cross-Session side was still being set up. We corrected this problem by using an entirely different method. The new user now sends a welcome message, after which the other users respond with their status.

Grid-service problem

The second problem worth mentioning was already solved in the architecture displayed in the previous chapter. The problem that occurred was that messages would arrive faster at the CSM than that it could deliver them to the user. The Grid Service notification system notifies a client that some data has changed, after which the client retrieves this data. This is slower than the channel based JSDT communications. To solve this problem we added buffers.

Chapter 8 Conclusion

8.1 Short Summary

Grid technology allows its users to share resources, such as CPU power, storage and expensive equipment. The Virtual Laboratory projects, both the VLAM-G and its follow-up the VL-e, seek to provide a layer between the low-level services and the application level, harnessing the strength of the Grid for a wide variety of applications and making the Grid available to a broader public. Furthermore, conform the e-science paradigm the VL-e has to provide a collaborative environment. It seeks to do so not just by global sharing, but also by providing a shared workspace environment. However, creating usable collaborative applications is far from easy and many projects tend fall far short of expectations or fail entirely. Reasons for this high rate of failure include creating the wrong types of applications, social differences in the user-group and failure to support the basic necessities of a collaborative environment.
We included these basic necessities in the requirements for the Collaborative System of the VL-e and grouped our requirements by priority. As a first set to be developed we believe it is necessary to:

- Provide session control
- Allow users to explore the workspace
- Provide a Userlist
- Provide an Instant Messenger
- Provide a Telepointer

The first two items are not solely collaborative features. Session control is closely related to access control in the VL-e in general, as only the users allowed to enter a study may participate in its collaborative activities. Exploring the shared workspace is related to the components which provide the graphical output for the workspace. Our focus is therefore on the latter three items.
The Userlist plays an important role as it is the first step in finding other users and establishing contact with them. It can be used as a bridge to many other applications, such as video- and audio-conferencing. The first version of our Userlist should be able to show the list of users belonging to a study, to respond to a click on a username by opening a message window and to allow for a status change.
The Instant Messenger is a basic form of communication, allowing users to send simple text messages to each other. Plausible deniability of availability and message windows remaining open are important advantages of instant messengers, as they allow users to decide for themselves when they wish to answer. The first version of the Instant Messenger should simply allow two users to exchange messages.
A Telepointer is the simplest form of gestural representation. By making users' mouse pointers visible to everyone in the shared workspace, the pointers can be used to gesture and point at

objects. Efficiency plays a crucial role with telepointers, lest their movement become erratic. Due to a lack of time, the Telepointer's development is postponed.
Two other very important aspects of shared workspace environments are workspace awareness and data consistency management. Providing awareness is vital for collaborative work, as it is crucial to know what team-members are doing. By providing a small version of the workspace in a secondary window important awareness is provided. Data consistency management plays an important role when it comes to multiple users accessing the same object in the shared workspace. Finding a good consistency policy is far from easy, as care must be taken to provide a sensible yet efficient method. Not allowing users to access objects simultaneously may prevent conflicts, but the waiting times may be annoying for the users. Allowing users to access and change everything simultaneously on the other hand, will require complex error-correcting mechanisms. The proper policy for the VL-e will have to be considered carefully.
Many toolkits nowadays exist on the Internet to aid in the creation of collaborative applications. Most of the toolkits are, unfortunately, unsuitable for our application. Some force its users to use specific architectures, other are too limited or simply unsuitable. We decided to use the Java Shared Data Toolkit, which gives more architectural freedom and does not provide complete components, but provides the tools to create components. It is therefore highly customizable and allows for a range of applications to be developed.
Combining the requirements of the Userlist and Instant Messenger with requirements specific for the VL-e, such as having to be a Grid-service and providing a light-weight client, we used parts of the UML to develop our architecture. It contains a client and three Grid Services: a Global Collaborative Manager, a Cross-Session Collaborative Manager and an Inner-Session Collaborative Manager. The first is the glue between separate studies. The second is the client, but located on the server side, in order to provide a light-weight client. The third belongs to a study and its distributed nature should make the system more fault-tolerant and should allow for a decrease in network congestion.
Though it is far from finished, part of the architecture has been implemented successfully.

8.2 Discussion

Coupling

Using services decouples individual components, as communication can only take place via interfaces. By using the JSDT's channel based communication we reintroduce coupling. This component dependence is only inside the system. The system itself is still a component which is just another building block of the larger VL-e.

The necessity of the ISM

We introduced the ISM as it allows a distributed approach. We described many advantages to this implementation. However, a version with only a GM and a CSM will also work and it may be easier to implement and maintain. It remains open to discussion whether having the ISM is the better solution.

Expandability

We believe that the architecture we created can easily be expanded to include other media. It is possible to use UDP to send data over Channels, which was also suggested as protocol for the implementation of the Telepointers. The problem is that we do not know whether the Grid service notifications between the CSM and the CommunicationsHandler will be able to keep up.

Latency

One of the requirements was to create a thin client, which resulted in the CSM. Since this is another layer between two clients, latency will be higher than if the Client side had communicated directly with the ISM. It may be possible that for the Instant Messaging this will not prove to be a problem, however when streaming media, such as video and audio, are concerned this will have to be properly tested. It the latency proves to be too much, the thin-client may prove to be a liability.

8.3 Future Work

At this point our design and implementation have been focused on the Instant Messenger and the Userlist. As we explained in Chapter 4, the Collaborative System has more than just these two features. These requirements need further fleshing out so that they can be properly included in the VL-e. As the field of CSCW is evolving rapidly many new research papers are appearing on subjects relevant for our system. These developments should be monitored closely as they can greatly help in identifying requirements for specific components, as well as in their design.
Where our current system is concerned, there is room for a lot of improvement. In the previous Chapter we showed items which are yet to be included in our system. Furthermore, as we are dealing with work in progress new question keep coming up: How will we deal with conversations over a channel when that channel's ISM is destroyed by the administrator? Should the system be able to function without the GM, in a stand-alone version? Wouldn't it be better to let the CSM provide the ISM with the Client handle, instead of letting the client do that? Should the status be shown of users who are "online in other"?
Nevertheless, now that we have proved the architecture has potential, we should make a stable, usable version. However, the developments of the Globus Toolkit will have to be closely monitored. Their switch from OGSI to WSRF will very likely make our current implementation incompatible with the new toolkit, though they claim the effort required to change on OGSI-based system to WSRF will be small. It is important to monitor the changes closely and change to WSRF as soon as it is apparent that this will be their definite course and a new Globus Toolkit becomes available.

Bibliography

[1] Afsarmanesh, H., Belleman, R.G., Belloum, A. S. Z., Benabdelkader, A., van den Brand, J. F. J., Eijkel, G. B., Frenkel, A., Garita, C., Groep, D. L., Heeren, R. M. A., Hendrikse, Z. W., Hertzberger, L. O., Kaandorp, J.A., Kaletas, E. C., Korkhov, V., de Laat, C.T.A.M., Sloot, P.M.A., Vasunin, D., Visser, A., Yakali, H. H. (2002) **VLAM-G: A Grid-Based Virtual Laboratory**

[2] Allen, C. *http://www.alacrityventures.com/DoG.html* (Definitions of Groupware)

[3] Ariane Training (2001) **UML Applied Object Oriented Analysis and Design Using the UML**

[4] Baker, K., Greenberg, S. and Gutwin, C. (2001) **Heuristic Evaluation of Groupware Based on the Mechanics of Collaboration.** In M.R. Little and L. Nigay (Eds)

[5] Belloum, A.S.Z. (2001) **The Virtual Lab-AM Collaborative System**

[6] Belloum, A.S.Z. **VLAM-G user's Guide Proposal**

[7] Burridge, R. (1999) **Java Shared Data Toolkit User Guide**

[8] Bradner, E., Mark, G. (2002) **Why Distance Matters: Effects on Cooperation, Persuasion and Deception**

[9] Cadiz, J.J., Venolia, G., Jancke, G., Gupta, A. (2002) **Designing and Deploying an Information Awareness Interface**

[10] Bruegge, B., Dutoit, A.H. (200) **Object-Oriented Software Engineering Conquering Complex and Changing Systems**

[11] Dyck, J., Gutwin, C., Subramanian, S., and Fedak, C. (2004) **High-Performance Telepointers**

[12] The Globus Aliance *http://www.globus.org*

[13] Greenberg, S., Marwood, D. (1994) **Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface**

[14] Greenberg, S. Gutwin, C., Roseman, M. (1996) **Semantic Telepointers for Groupware**

[15] Gutwin, C., and Greenberg, S. (2001) **A Descriptive Framework of Workspace Awareness for Real-Time Groupware**

[16] Gutwin, C., Penner, R. (2002) **Improving Interpretation of Remote Gestures with Telepointer Traces**

[17] Gutwin, C. and Greenberg, S. (2000). **The Mechanics of Collaboration: Developing Low Cost Usability Evaluation Methods for Shared Workspaces**.

[18] Grudin, J. (1988) **Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces**

[19] Handel, M., Herbsleb, J.D. (2002) **What is Chat Doing in the Workplace?**

[20] Johnson-Lenz, P., Johnson-Lenz, T. (1998) **Groupware: Coining and Defining It**

[21] Johnson, P. (1996) **State as an Organizing Principle for CSCW Architectures**

[22] Nardi, B.A., Whittaker, S., Bradner, E. (2000) **Interaction and Outeraction: Instant Messaging in Action**

[23] Prince, S., Cheok, A.D., Farbiz, F., Williamson, T., Johnson, N., Billinghurst, M., Kato, H. (2002) **3D Live: Real Time Captured Content For Mixed Reality**

[24] Taylor, J. *http://www.e-science.clrc.ac.uk/* (e-Science definition)

[25] Usability First *http://www.usabilityfirst.com/groupware/index.txl* (Groupware)

[26] WTCT NV (2003) **Virtual Lab e-Science: Towards a new Science paradigm**

[27] Stevens, M. *http://www.developer.com/design/article.php/1010451* (Service-Oriented Architecture Introduction)

Appendix A: Use Cases

Userlist

Use case name	Initiate Conversation
Participating actor	Invoked by User
Entry condition	User A clicks on User B’s name in the userlist
Main flow	
	1. The CS determines that a new message window has to be created addressed to that user
Alternative flow	User B is Busy / Away
	1. User A receives a warning from the CS concerning User B’s state
	2. Either <Main flow> or <Alternative flow: Window Exists>
	Window Exists
	1. The CS determines that the User already has an existing message window towards the user. He uses the old one.
Exit condition	A message window to User B is opened by the CS

Use case name	Invite to Join Session Manager
Participating actor	Invoked by User
Entry condition	User A has selected User B from his Userlist and invoked the Invite to Session function
Main flow	
	1. CS provides User A with a list of his active studies
	2. User A selects a study from the list
	3. CS provides User A with a confirmation request
	4. User A confirms
	5. CS sends the invitation to User B
Exit condition	User B receives the invitation
Exception flow	Cancel Invitation
	1. CS provides User A with a list of his active studies
	2. User A selects a study from the list
	3. CS provides User A with a confirmation request
	4. User A cancels

Use case name	Accept Invitation to Join Session Manager
Participating actor	Invoked by CS
Entry condition	An invitation has been sent to User A

Main flow	1. CS presents the invitation to User A 2. User A accepts the invitation
Exit condition	User A has the information to try to join the Session Manager.
Exception flow	Invitation Declined 1. CS presents the invitation to User A 2. User A declines the invitation

Note: The CS will merely provide User A with the means to join the Collaborative Session here. See the "Join Collaborative Session" use case for more details on joining the Collaborative Session.

Use case name	Add to Personal Contacts
Participating actor	Invoked by User
Description	Allows the User to add another user to his personal contacts. The flows have not yet been created for this use case, as it requires some discussion with the rest of the group on where and how to find the users.

Use case name	Remove from Personal Contacts
Participating actor	Invoked by User
Entry condition	The User selects a user and presses the "del" button
Main flow	1. The CS request confirmation 2. The User confirms
Exit condition	The selected user is removed from the userlist
Exception flow	Cancel Delete Operation 1. The CS requests confirmation 2. The User cancels Illegal Delete Operation 1. The CS detects that the User has selected a user from his studies and not from his Personal Contacts. The User is made aware that he cannot delete the selected user.

Use case name	Change Status
Participating actor	Invoked by User
Entry condition	The User changes his current status to something different (e.g. from "online" to "away")
Main flow	1. The CS broadcasts the changed status

Exit condition	The CS updates the userlists of the other users to reflect the change

Instant Messenger

Use case name	Send Message
Participating actor	Invoked by User
Entry condition	User A presses send button in a message window
Main Flow	1. The CS delivers the message to whom the message window is addressed
Alternative Flow	First Message 1. The CS delivers the message to whom the message window is addressed 2. The CS finds that the User(s) do not yet have an open message window belonging to this conversation. He opens one.
Exit condition	User B's message window belonging to this conversation displays the message.

Use case name	Add to Conversation via Drag and Drop
Participating actor	Invoked by User
Entry condition	User A drags and drop User C from his userlist onto the message window
Main flow	
Exit condition	The CS adds User C to the conversation

Use case name	Add to Conversation via Menu
Participating actor	Invoked by User
Entry condition	User A uses an option from the message window menu to add User C to the message window
Main flow	
Exit condition	The CS adds User C to the conversation
Remark	This requires discussion with the rest of the team, as it may require searching the database.

Use case name	Leave Conversation
Participating actor	Invoked by User
Entry condition	User A clicks on the kill button of the Message Window
Main flow	
Exit condition	The CS destroys the window and terminates the communications

System

Use case name	Join Collaborative Session
Participating actor	Invoked by User
Entry condition	User just joined the Session Manager
Main flow	
	1. User tries to join the CS
	2. CS authenticates the User
	3. Authentication succeeds and the User is allowed to join
	4. The CS adds the User to this study's communication channels
	5. CS determines that this is the first Session the User has joined
	6. All other Users in the system are made aware that the User has joined
Alternative flow	Logged into a Second Study
	1. User tries to join the CS
	2. CS authenticates the User
	3. Authentication succeeds and the User is allowed to join
	4. The CS adds the User to this study's communication channels
	5. CS determines that the User is already logged into another Session
	6. All User who are logged into this SM are made aware that the User has joined
Exit condition	The CS provides the User with the accesslist belonging to the study
Exception flow	
	1. User tries to join the CS
	2. CS authenticates the user
	3. Authentication fails and the user cannot join

Use case name	Leave Collaborative Session
Participating actor	Invoked by User
Entry condition	User leaves the Collaborative Session
Main flow	
	1. CS determines that the User is not logged into another SM
	2. CS removes the User from the communication channels
	3. All users in the system are made aware that the User has left
Alternative flow	Logged into a Second Study
	1. CS determines that the User is still logged into a SM
	2. CS removes the User from this study's communication channels
	3. The users inside this study are made aware that the User has left the study
Exit condition	The User's userlist is updated; the study's accesslist is removed

Use case name	Create Collaborative Session
Participating actor	Invoked by Session Manager Communicates with Accesslist Provider
Entry condition	SM has just been created and now needs to become part of the CS
Main flow	1. CS creates the environment for the SM, the Collaborative Session (e.g. communication channels) 2. CS retrieves the accesslist from the AccessList Provider for authentication purposes
Exit condition	The SM is part of the CS

Use case name	Destroy Collaborative Session
Participating actor	Invoked by Session Manager
Entry condition	SM is being terminated and needs to leave the CS
Main flow	1. CS waits until communications have terminated
Exit condition	The Collaborative Session environment is destroyed

Use case name	Update Accesslist
Participating actor	Accesslist Provider
Entry condition	A change has occurred in the accesslist
Main flow	1. CS is notified of the change
Exit condition	CS updates userlists to reflect the new accesslist

Many of the previously defined use cases have an additional Exception flow namely:

Exception flow	User is offline 1. The CS displays a message that the User is offline