„DreamTeam“ - a platform for synchronous collaborative applications

J. Roth[1] and C. Unger[1],

[1] Praktische Informatik II, FB Informatik, FernUniversität Gesamthochschule Hagen
{Joerg.Roth, Claus.Unger}@fernuni-hagen.de

Summary

This paper presents a platform for developing, testing and executing synchronous collaborative shared applications in a distributed, heterogeneous environment. Even though several environments exist nowadays, specific problems are not treated satisfactorily. Especially in „real“ network environments, problems like unstable network connections and low bandwidths have to be considered.

The DreamTeam platform addresses the special needs of environments with non-optimal characteristics which can be found for example in distance education scenarios. DreamTeam comprises a development environment, a simulation environment and a runtime environment; it is based upon the concept of a fully decentralised architecture and encourages rapid prototyping.

In addition to several demonstration applications, two serious applications have been developed so far: a collaborative Web browser and a diagram design tool.

1 Introduction

Co-operative (CSCW) applications play a major role e.g. in distance education, for group discussions, jointly working on electronic courses, jointly visiting Web pages, remote laboratories, video conferencing, joint program development, co-operative publishing, etc. It is a difficult and time-consuming task to develop such a co-operative application.

[Dommel et al. 97] distinguish three types of co-operative applications:

- *Collaboration unaware applications* offer no collaboration services themselves; they are single user applications, running in a shared environment.
- *Collaboration aware applications* are developed for co-operative environments, but their services for collaboration are hard-coded.
- *Collaboration transparent applications* provide their services for collaboration by using high level services of a standard collaboration environment.

Taking the last approach, a developer can concentrate on application details and can use the collaboration oriented services from the standard collaboration environment.

In the following, we describe *DreamTeam*, a Java oriented development and runtime environment for synchronous, collaboration transparent applications. As examples for such applications we further describe DreamView, a co-operative Web browser, and Designer, a collaborative diagram design tool, which both have been developed within the DreamTeam environment.

2 The DreamTeam environment

The DreamTeam environment [Roth & Unger 98] allows the developer to develop co-operative applications like single user applications, without struggling with network details, synchronisation algorithms, etc. The environment consists of three parts: a development environment, a runtime environment and a simulation environment. In the following, the basic DreamTeam concept as well as the three components are described in more detail.

2.1 The concept

Architecture: Co-operative applications can either be implemented in a centralised or in a de-centralised way. In a centralised architecture, all messages are routed via a single group server, thus group events and data accesses can easily be synchronised. On the other hand, group servers often are performance bottlenecks, a shut down of a group server shuts down the session; all messages, even those between clients residing in the same LAN, have to be sent through the group server.

In a de-centralised architecture, availability and bandwidth problems can be avoided, but synchronisation and serialisation have to be handled by higher level protocols. On the other hand, a de-centralised architecture allows different kinds of communication channels, e.g. via ISDN connections.

The actual version of DreamTeam supports de-centralised architectures without the need for any central server.

Session management: In a de-centralised architecture, session management has to be handled in a de-centralised way too. The originator of a session defines a ses-

sion profile and generates the session. As soon as she has started the session, other users can join and leave the session. When the originator himself leaves the session, the session can continue with the remaining members but no new members can join in.

Relaxed WYSIWIS: A common paradigm for shared workspaces is WYSIWIS (What You See Is What I See) [Stefik et al. 87]: the strict WYSIWIS policy enforces identical workspace displays on every participating system, whereas a relaxed WYSIWIS policy allows private areas and slight layout differences. Since DreamTeam is designed for heterogeneous environments, it supports relaxed WYSIWIS.

Application modes: For an end-user it may be meaningful to use a shared application in a private environment as well, without having to learn a new user interface. With minor additional effort, DreamTeam applications can be developed for both, shared as well as private environments. The application can ‘ask’ the environment in which mode it is currently running and can enable or disable specific functions.

Information distribution: To distribute information among session members, DreamTeam uses a special kind of Multicast Remote Procedure Calls, through which a local system can call a procedure at all participating remote systems. Complex data structures can be serialised, transferred as procedure parameters and rebuilt at the target site. All objects defining a DreamTeam session can be serialised; thus even for a late-comer the actual session state can easily be generated.

2.2 The protocol hierarchy

A DreamTeam application developer can build his or her applications upon high-level services. Basic problems such as establishing network connections, broadcasting events, synchronising data etc. are kept away from the developer, thus allowing her to concentrate on application rather than communication aspects.

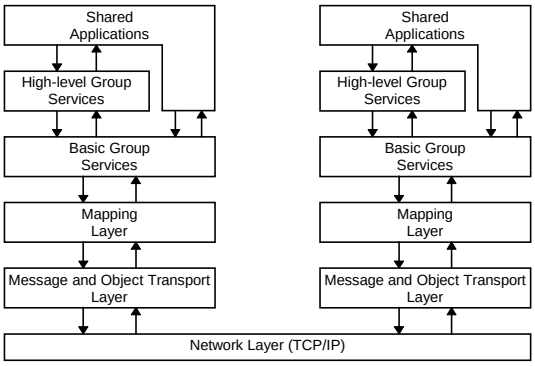

Figure 1: The DreamTeam protocol hierarchy

DreamTeam protocol services are divided into different protocol layers (Figure 1).

The lowest layer represents the access layer for the runtime system to the network. The current implementation is based on TCP/IP, but other network layers can easily be integrated.

The *Message and Object Transport Layer* provides elementary transport services for unicasting and multicasting. On this level, different participants are represented by an internal identification rather than the raw network ID. This is necessary, because network IDs are not always sufficient for user identification (see chapter 2.4.1). To transport arbitrary data, Java’s Object Serialisation [SUNb] is used.

The *Mapping Layer* can be seen as a big multiplexer/demultiplexer. Shared applications are divided into a number of *resources*. Every resource can communicate with its corresponding replicated resource via the mapping layer. Figure 2 shows the procedure.

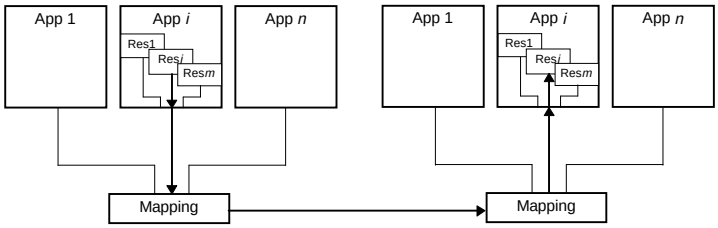

Figure 2: Resource mapping

The mapping mechanism contains a list of all applications of a session and its resources. If data are passed to the mapping layer, it determines the originator. Application and resource identifiers are added to the data package. On the receiver side, this information is stripped from the package and the data are directed to the specific receiver resource.

The *Basic Group Services* layer is the lowest layer which can directly be accessed by shared applications. This layer contains services such as multicast RPC, distributed semaphores and shared objects. Special services give access to session and user profiles, e.g. an application can retrieve a profile list of all users currently participating in the session.

The *High-level Group Services* layer maps basic services to complex services. This category includes distributed mousepointers (see 2.4.2), tracking windows (see 2.4.3) and participant windows.

2.3 Platform components

In the following, the three main components of the DreamTeam platform are described in more detail.

2.3.1 The development environment

The development environment mainly consists of a huge hierarchical class library [Roth 98]. It is entirely written in Java [SUNa], thus runnable on many operating systems. We strictly paid attention to use platform independent language elements only, and tested the library on *Solaris*, *Windows95*, *OS/2* and *Linux*.

A developer, who wants to build a shared application with the help of DreamTeam, has to proceed as follows:

- Build a subclass of a library class called *CSCWApplication* in which a standard behaviour of shared applications is already defined. In the subclass, only functions have to be coded which differ from the standard.
- Write your own classes where necessary. For some Java classes, counterparts exist which define a standard group behaviour. The classes *CSCWFrame* and *CSCWCanvas*, e.g., are the DreamTeam variants of the Java classes *Frame* and *Canvas*.

Using the DreamTeam class library leads to compact programs. A distributed, window-oriented *Hello-World* program, for example, contains only two classes with a total of 150 lines of Java code, most of them defining the dialogue behaviour.

2.3.2 The runtime environment

In order to start sessions and shared applications, a runtime environment has to be established. In addition to a front-end the following tasks are started:

The *Log Manager* collects relevant system messages and stores them into a file. This task is import for debugging purposes. System messages are classified into *information messages*, *notification messages*, *minor exceptions*, *fatal exceptions* and *errors*. Depending on the message level, either the message is put into a message list box, or an alert window opens. For important events, audio signals are provided to attract the user’s attention.

The *Connection Manager* is active during a running session and handles the communication between shared applications. It uses a fast communication channel with minimal delay and is run as a high priority task. Connection channels are realised via permanent sockets, which are opened when a session starts or when a user joins a session. They stay open until the session finishes or the user leaves a session.

The *Session Manager* handles session profiles, starts and stops sessions and enables joining and leaving sessions. The Session Manager uses a special kind of communication channel which is opened and closed on request for single transactions.

The *Transfer Manager* provides data transport for slow data. Transfer channels are used in the background with the lowest priority. They are, for example, used for file transfer and, like session channels, are opened and closed on request.

The *Archive Manager* can be viewed as a small database for long-term data such as old user profiles or profiles of terminated sessions.

To interact with the system, a front-end is included in the runtime environment. The following figure shows a typical working environment.

Figure 3: The DreamTeam working environment

The upper left main window allows to configure host and user profiles, retrieve user profiles of participants of previous sessions (lower left window); define and edit session profiles; start and stop sessions (icons representing open sessions are shown in the upper right window); start private applications (icons are shown in the lower right window) and view users currently connected to a session.

2.3.3 The simulation environment

For testing software, it is necessary to run it under real conditions. In the case of distributed synchronous applications, it is not always possible to make an adequate network available. If network effects of slow networks connections should be examined (e.g. in case of modems), extensive testing is difficult. DreamTeam offers a simulation environment called *DNS* (*DreamTeam Network Simulator*). DNS allows to start several runtime environments on a single computer. Shared applications can be tested without modifications or recompilation (e.g. no other program library is required). DNS is able to simulate networks with reduced bandwidths and network delays. Every communication channel can be configured separately. To detect performance problems, DNS provides overview functions for network loads.

There are manifold reasons to use a simulator rather than testing in a real network environment:

- In many cases, only one computer is available for developing distributed applications.
- The software developer possibly doesn't have network access or a network access is expensive (e.g. think of students who develop applications at home).
- Mostly, the number of available telephone links and modems is reduced.
- In case of real modem or ISDN connections, special server software has to be set up and configured for testing.
- It is not easy to switch between different scenarios in real environments. A change of link characteristics always requires a reconfiguration of the connection software.
- In addition to simulation facilities, a simulator can offer analyse functions.

Testing shared applications inside the simulation environment ensures stability for a real usage and avoids performance problems which otherwise would be detected very late in the development cycle.

2.4 Special problems and their solutions

In the following, typical problems in a de-centralised, heterogeneous, synchronous environment are described and it is shown how they can be solved on the DreamTeam platform.

2.4.1 Group rendezvous

A de-centralised group rendezvous must work without a „well-known server“. If no central server keeps the group state, it is difficult for a new participant to reach his group. The problem becomes even more complex, if Internet addresses change between dial-ins. To solve this problem, we divide all group members into two categories, the first category containing all participants with fixed network addresses (e.g. the teaching staff), the second one containing the members with variable addresses (e.g. students). Members of the second category are represented by sets of potential addresses. The address or the set of addresses is stored in the user profile. When a participant wants to join a group, his system first has to build a list of all other systems currently online. Hereto the rendezvous component queries all known systems - beginning with the fixed addresses. A query is successful, if another system is already online (i.e. the network access has already been established and DreamTeam has been started). This other system has already built an own address list which it now transfers to the newcomer. The newcomer can now ask the other systems to update their local address lists. This procedure ensures that, after a certain delay, all systems know the state and address of all other systems in their group. Even in the case of variable addresses, a broadcast is

avoided if at least one participant with a fixed address is online. In the worst case (all group members have variable addresses), a broadcast is necessary. Nevertheless, the network load for this broadcast is acceptable, because only addresses stored in the user profile have to be tested. If participants are connected to an MBone-enabled network, multicast IP is used rather than broadcasting messages via unicast datagrams. The rendezvous component tests the network capability for multicast automatically, thus manual set-up is not necessary.

Before a group member can join his group for the very first time, he has to have an initial host list. In order to get this list, he has to be invited explicitly by another group member (e.g. by a teacher). This member transmits the own list of known hosts to the newcomer by file. This can be done either by email, ftp or by sending a floppy via mail. The rendezvous component reads the invitation file and automatically builds the corresponding internal structures, thus a user has neither to deal with raw network addresses nor confusing configuration files.

2.4.2 Distributed mousepointers

Distributing mousepointers between users is a basic group-specific service. Mousepointers are an ideal way to point to items in shared documents. Unfortunately, it is not possible to map mouse co-ordinates directly between the connected users. According to the relaxed WYSIWYS paradigm, shared documents represent the same content, but have slight differences in layout (different scales, line breaks, scrollbar settings etc.). Thus, a direct mapping is impossible. DreamTeam provides two different schemes to map mouse co-ordinates across sessions:

- *relative mapping*: for this mapping, it is assumed that a document layout on different platforms only differs in scale, e.g. fonts have different sizes, but line breaks are identical. In this case, mouse co-ordinates can simply be transferred and multiplied by a constant factor.
- *arbitrary mapping*: if in addition to different scales, the relative positions and line breaks differ, an arbitrary mapping is necessary. The next figure shows the mapping method.

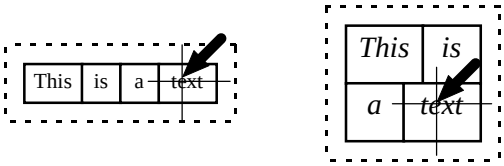

Figure 4: Mapping of mouse co-ordinates

Each document item (e.g. word, image) is represented by an item identification (e.g. sequence number inside the document) and a bounding box. If a mousepointer on one platform points to an item, the item identification and the relative

position inside the bounding box is broadcasted to other participants. To show the pointer on other platforms, the local bounding box of the corresponding item is retrieved and the local relative position is calculated. This method ensures that when someone points to a specific item, the distributed mousepointers on other systems point to this item as well.

2.4.3 Document tracking

A problem in synchronous collaborative work is to give an orientation about the current work of other participants. Although modifications are broadcasted to every participant, it is not always clear, which part of a document is currently edited by other users. This problem becomes more complex in large documents. [Gutwin et. al. 96] suggest dialogue elements such as radar views and multi-user scrollbars to obtain orientation. DreamTeam provides an element, which is a combination of a multi-user scrollbar and a radar view, called the *Document Tracker*. A Document Tracker is a small window, which presents the document window of another user on a 1:3 scale, including the corresponding scrollbar. The document tracker keeps track of all scrollbar events, thus the tracking window is updated immediately. A tracking window can be opened for every other user currently participating in the session. It is possible to synchronise the own document window with another user’s document window. This service is useful for electronic lectures in which the teacher has control over a document and students follow the explanations.

3 Applications based on DreamTeam

The Web browser *DreamView* is the first substantial co-operative application developed in the DreamTeam environment. Beside providing a useful tool for distributed education, DreamView was developed for testing and validating the DreamTeam design concept.

DreamView allows a group of users to co-operatively browse the World-Wide Web. The reasons for implementing a co-operative Web browser are manifold. In addition to browsing remote documents, Web browsers can be used to browse local HTML documents; manuals and course materials can be published as HTML trees and distributed on CD-ROMs.

Beyond common browser functions DreamView offers the ability to put annotations into Web pages. Annotations can be viewed by all session members and can point to page items (e.g. images or words). Tracking windows (see chapter 2.4.3) give orientation about current page positions of other users.

To co-operatively design diagrams such as entity relationship diagrams, we developed a diagram editor called *Designer* (Figure 6). Designer allows to manipulate diagram items as objects rather than as bitmap images, thus diagram structures

can easily be edited and redesigned. Overview functions and distributed mouse-pointers are included.

Figure 5: The browser environment

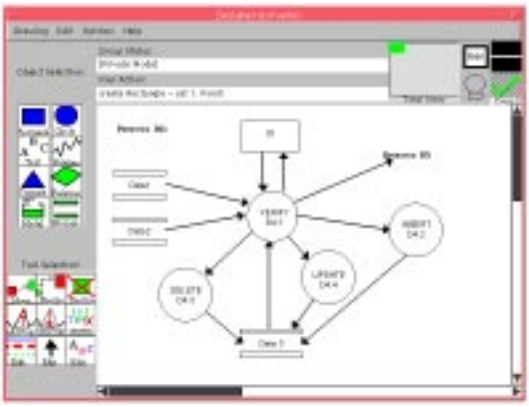

Figure 6: The designer environment

4 Related work

This section relates DreamTeam to other synchronous co-operative platforms.

The *Groupkit* system [Roseman et al. 96] is a package for implementing shared applications under TCL-TK. A library offers services for session management, communication and shared dialogue management. It is mostly based upon a de-centralised architecture, only for the group rendezvous a central server is needed. A co-operative Web server called *Groupweb* is built on the Groupkit platform. In addition to telepointers and co-operative browsing, it allows the annotation of Web pages. In contrast to DreamView, a page can only be annotated as a whole. Annotations cannot be anchored to page items.

Habanero [NCSA] has been completely implemented in Java. Its architecture is centralised, i.e. requires a Habanero server application in order to enable group activities. A co-operative Web browser is available, called *WWW shared session.* Actually, this browser is the Mosaic browser, which is controlled via a data channel. Thus, group specific services (e.g. telepointers) are not available.

The *Rendezvous* system [Hill et al. 93] provides a distribution mechanism based upon X-windows events. This approach is completely different to our approach, since its emphasises on distributing window contents and user events via the X-protocol. This approach can hardly be applied to non-X-windows environments.

Share-Kit [Edlich 93] is a Unix-based platform which provides multicast RPC for C programs. Neither session management nor group specific widgets are included in this platform.

Dolphin [Streiz et al. 94] is a co-operative hypermedia system. It emphasises co-operative editing of hypermedia documents. It is written in Smalltalk and provides one hardcoded application, a shared hypermedia editor. The underlying platform is *COAST* [Schuckmann et al. 96], which is designed to offer general services for synchronous, document-based groupware.

Compared to other platforms, DreamTeam offers a straight forward implementation of shared application and encourages rapid prototyping. The simulation environment helps to find out weak points and ensures stability before delivering applications to end-users.

Especially the de-centralised architecture of DreamTeam is a significant difference to other systems. The de-centralised structure avoids bottleneck problems with central servers and minimises the administration costs for the teaching staff.

5 Conclusion and future work

DreamTeam is a platform for synchronous co-operative applications. It consists of a runtime environment, a development environment and a simulation environment.

The platform is designed to minimise the development cycle for co-operative applications and to encourage rapid prototyping.

The current implementation of DreamTeam uses socket streams for communication during sessions. Socket streams are reliable (no packet losses, packets arrive in order), but are considerable slow. In the future, we plan to integrate real-time services such as audio and video transfer into the platform. For this, it is necessary to use faster communication channels like datagrams or IP multicasting. To provide a reliable communication interface to shared applications, reliable multicast protocols will be added to the *Message and Object Transfer Layer* (see Figure 1). New services for using unreliable but fast real-time channels will be added to the *Basic Group Service Layer*, whereas upper layers will not have to be changed.

To further minimise the development effort for new shared applications, we plan to define a language extension. A new set of keywords allows to write more compact programs. A precompiler translates this keywords into a set of library calls. Precompiling results can then simply be translated by the original Java compiler, thus the Java utilities can still be used.

Addressing another direction, next versions of DreamTeam will make component concepts available for shared applications. We carefully examined Java’s *Beans* concept [Hamilton 97], but in our opinion Beans cannot simply be made shareable in synchronous sessions. Beans are small isolated programs which offer a number of methods. The Bean developer determines which implementation details are made visible to the outside world. For our aims, it is necessary for a component to fulfil a number of additional requirements in order to run in our environment. Therefore we will define our own component interface standard, which better conforms to groupware concepts.

6 References

[Dommel et al. 97] Dommel, H.-P.; Garcia-Luna-Aceves, J.J. (1997): Floor control for multimedia conferencing and collaboration, Multimedia Systems, Vol. 5, 1997, 23-38

[Edlich 93] Edlich, S. (1993): Software Cupertino with the Share-Kit: Influences of Semantic Levels on the Working Efficiency, Vienna Conference on Human Computer Interaction VCHCI '93, Vienna, Austria, Sept. 20-22, 1993, 225-234

[Gutwin et. al. 96] Gutwin, C.; Roseman, M.; Greenberg, S. (1996): A Usability Study of Awareness Widgets in a Shared Workspace Groupware System, Proc. of the ACM’96 Conference on Computer Supported Co-operative Work (CSCW ‘96), ACM Press, Nov. 1996, 258-267

[Hamilton 97] Hamilton, G. (ed) (1997): Java Beans, Sun Microsystems, 1997

[Hill et al. 93] Hill, R.D.; Brinck, T.; Patterson, J.F.; Rohall, S.L.; Wilner, W.T. (1993): Rendezvous Language, Comm. of the ACM, Vol. 36, No. 1, Jan. 1993, 62-67

[NCSA] NCSA Habanero Homepage, http://www.ncsa.uiuc.edu/SDG/Software/Habanero/HabaneroHome.html

[Roseman et al. 96] Roseman, M.; Greenberg, S. (1996): Building Real-Time Groupware with GroupKit, A Groupware Toolkit, ACM Transactions on Computer-Human Interaction, Vol. 3, No. 1, Mar. 1996, 66-106

[Roth & Unger 98] Roth, J.; Unger, C. (1998): „DreamTeam“ - a Synchronous CSCW Environment for Distance Education, to appear in the ED-MEDIA/ED-TELECOM’98, Freiburg, Germany, Jun. 98

[Roth 98] Roth, J. (1998): How to write shared applications with „DreamTeam“, Internal Technical Reference, Fernuniversität Hagen, Jan. 1998

[Schuckmann et al. 96] Schuckmann, C.; Kirchner, L.; Schümmer, J.; Haake, J.M. (1996): Designing Object-Oriented Synchronous Groupware With COAST, Proc. of the ACM Conference on Computer Supported Cooperative Work (CSCW ‘96), ACM Press, Nov. 1996, 30-38

[Stefik et al. 87] Stefik, M.; Bobrow, D.G.; Foster, G.; Lanning, S.; Tatar, D. (1987): WYSIWIS revised: early experiences with multiuser interfaces, ACM Transactions on Office Information Systems, Vol. 5, No. 2, Apr 1987, 147-167

[Streiz et al. 94] Streitz, N.A.; Geißler, J.; Haake, J.M.; Hol, J. (1994): DOLPHIN: Integrated Meeting Support across LiveBoards, Local and Remote Desktop Environments, Proc. of the ACM Conference on Computer Supported Co-operative Work (CSCW '94), Chapel Hill, North Carolina, Oct. 22-26, 1994, 345-358

[SUNa] JavaSoft Home Page, http://java.sun.com

[SUNb] Java Object Serialisation Specification, Sun Microsystems, 1997