A Collaborative Software Infrastructure based on the High Level Architecture and XML

NICHOLAS MONTGOMERIE-NEILSON

KTH Numerical Analysis
and Computer Science

Master's Degree Project
Stockholm, Sweden 2005

TRITA-NA-E05046

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

A Collaborative Software Infrastructure based on the High Level Architecture and XML

NICHOLAS MONTGOMERIE-NEILSON

TRITA-NA-E05046

Master's Thesis in Computer Science (20 credits)
at the School of Engineering and Business Management,
Royal Institute of Technology year 2005
Supervisor at Nada was Henrik Eriksson
Examiner was Stefan Arnborg

A collaborative software framework based on the High Level Architecture and XML

Abstract

A study is made of using the High Level Architecture (HLA) as foundation for distributed applications in the domain of Computer-Supported Collaborative Work. A plug-in, peer-to-peer infrastructure for such applications is proposed, aimed at facilitating development and management of collaborative software. Users of the framework collaborate in groups and sessions, described by a replicated state XML information model. A prototype infrastructure is developed, along with three prototype collaborative applications. Results of performance testing show that a transport system built on HLA compares reasonably well with a socket-based transport system. On the whole, results demonstrate feasibility of the infrastructure and of the objective of extending the HLA to non-simulation applications. Future work to adapt full-scale applications to the collaborative infrastructure is invited.

Keywords: HLA, CSCW, computer supported collaborative work, XML, distributed systems

En mjukvaruinfrastruktur för datorbaserad samverkan, byggd på High Level Architecture och XML

Sammanfattning

Möjligheten att använda High Level Architecture som bas för distribuerade applikationer för datorbaserad samverkan studeras. En infrastruktur för sådana applikationer utvecklas, grundad på plug-in- och peer-to-peer-principer och med målen att förenkla utveckling och administration av program för datorbaserad samverkan. Samarbete i infrastrukturen sker i grupper som arbetar i sessioner. Grupper och sessioner beskrivs av en informationsmodell uttryckt i XML som underhålls i distribuerade, replikerade kopior. En prototyp för infrastrukturen utvecklas och dessutom tre prototyper för samverkansprogram. Testresultat visar att kapaciteten för det HLA-drivna kommunikationssystemet är jämförbar med ett direkt socketdrivet system. Generellt sett visar studien att den föreslagna infrastrukturen är möjlig. Den demonstrerar också ett sätt på vilket HLA kan användas för andra syften än simulering. Framtida arbete för att anpassa en fullskalig applikation till samarbetsinfrastrukturen uppmuntras.

Nyckelord: HLA, CSCW, datorbaserad samverkan, XML, distribuerade system

Table of Contents

List of figures

List of tables

Acknowledgements

A large *thank you* goes out to Henrik Eriksson for being a great supervisor – your comments contain more information per word than any I have ever seen, to Jenny Ulriksson and many others at FOI for the excellent support and hospitality you have offered, and to Mattias Liljeström for countless valuable discussions and good times during course of the thesis project.

1. Introduction

As the morning subway train approaches the center of the city, Jane powers up her laptop computer. Via a wireless internet connection, she logs into her collaborative desktop to find it just as she left it on her office workstation late last afternoon. The same applications she was running yesterday appear, each loaded with current working material. Jane notices that Robert has already logged in and done some more sketching on the new library his and Jane's architecture firm are designing. Jane sends Robert a "good morning"-message and compliments him on his new ideas. Jane then switches to the collaborative word processing program where her team is jointly composing a progress report on the library project. As project manager, she writes a short section on Robert's latest initiative and then makes a modification in the goals paragraph which she notices Lisa has added last night. Jane then enters the project calendar application and confirms that an entry has been properly made for the dentist's visit she is about to make. She powers down, looks out through the window and thinks about the fresh taste of dentist's tooth polish.

1.1 Computers and computer communication

The story of computer technology in the 20th century is the story of a revolution. Whether one looks at usage volume, performance, or societal impact, one will find that computer technology compares with or surpasses any other identifiable class of technology.

Tanenbaum & Steen (2002) tell the following tale about computer performance development in the last sixty years. In 1945, a computer cost 100 million dollars and executed one instruction per second. Today, a 1000 dollar computer executes 10 million instructions per second, representing a price/performance improvement of one trillion (10^{12}). This pace of progress outclasses that of any other industry.

In more recent years, there has also been an explosive expansion of networking. Gartner (2004) estimates that there were 550 million internet subscribers in 2004. These people have access to information and communication resources on an unprecedented scale. Internet users receive around 30 billion e-mails *per day* (IDC, 2002). Even taking into account the problems of spam (junk-mail represents just under half of current global e-mail volume), e-mail is a hugely successful application of network technology and has had a substantial impact on professional and personal communication.

E-mail represents the most successful example of so-called *collaborative software*. E-mail has been successful despite the fact that (or perhaps because) it does not address all of the four capabilities that collaborative software is ideally supposed to support: communication, collaboration, coordination, and control (Eseryel, Ganesan, & Edmonds, 2002). Out of these four, e-mail supports only communication.

There are various applications that support all of the four services just mentioned—but none have been successful. For the type of application that 'Jane' uses in the example above—the type of application where users can collaborate directly

to perform common tasks—both supply and demand have been weak. One problem this type of software faces is the difficulty of achieving effective collaboration between people that can use only a narrow channel of communication. Another problem is the complexity inherent in designing applications that need to address networking issues, manage collaborator groups, provide communication channels between collaborators, and synchronize collaborator activities—and doing all this while also managing their core tasks. As a result of these issues, the superior software for most computer-related professional tasks is a single-user, local desktop application.

1.2 Thesis goals

The present Master's thesis project attempts to solve some of the problems that arise when designing collaborative applications. It proposes a framework application that provides a pluggable interface for collaborative tools, and that relieves these collaborative tools of responsibility for managing user groups and of most of the responsibility for communication. In a theoretical treatment and a practical implementation, the thesis proposes a framework application using XML-based group definitions and a communication infrastructure built on an architecture called the HLA (High Level Architecture).

Table 1-1 Conditions that the proposed application needs to satisfy

Key design conditions
▪ Employ a pluggable collaborative tool architecture ▪ Manage groups for collaborations ▪ Define an XML group information model ▪ Employ peer-to-peer architecture ▪ Use HLA for distribution support

The most important specification items for the proposed framework application are listed in Table 1-1. Investigating a "pluggable collaborative tool architecture" is the main goal of the application—an architecture where collaborative applications can easily plug into a framework application to take advantage of its various services. The aim here is not to allow existing single-user applications such as *Emacs* (text editor) or *Paint* (drawing application) to be plugged in directly to a collaborative framework. Rather, the aim is to simplify and quicken the task of adapting such programs for collaborative work, or to simplify and quicken the task of writing collaborative programs from scratch.

To "manage groups for collaborations" is a central condition for the present thesis. A related thesis by Liljeström investigated questions directly linked to the interface to plug-in tools. The work distribution between the present thesis and Liljeström's thesis is schematically illustrated in Figure 1-1.

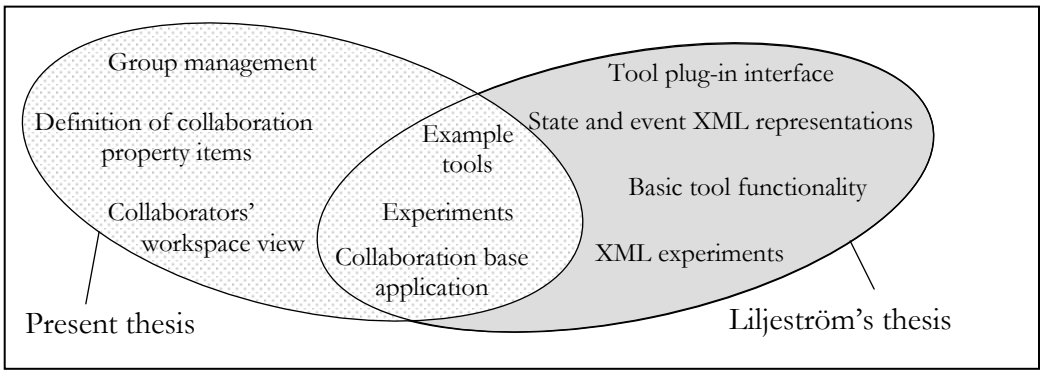


Figure 1-1 Thesis work distribution
Certain work in the present Master's project was done in cooperation with Liljeström's NADA Masters' project. The figure illustrates how some of this work was separated and shared.

The present thesis investigated a series of questions related to the design conditions mentioned previously. The questions are listed in Table 1-2. Every question was answered by a design recommendation. Most of the design recommendations were also implemented in demonstration code.

Table 1-2 Listing of the principal questions that the conditions in Table 1-1 lead to

Key design questions

- What kinds of distribution support will be enlisted by the HLA?
- How will consistency be ensured in the collaboration?
- How is a collaborative group defined?
  - What data should the definition include?
  - How should the definition's life-cycle look?
- How should definitions of collaborative groups be maintained?
  - …with regards to allowing efficient storage?
  - …with regards to enabling searches?
  - …with regards to properly managing modifications?
- How should logins and logouts to/from the collaboration be handled?
  - …with regards to performance?
  - …with regards to enabling effective awareness?
- How should the interface to tools look, concerning general functions?
  - Handled in a parallel FOI/KTH study authored by Liljeström

1.3 Where the thesis was developed

The present thesis was presented at the computer science department of the Royal Institute of Technology, Stockholm (KTH NADA). Research and report writing was performed at the Swedish Defence Research Agency department of Systems Modeling (FOI Systemmodellering). A partial goal of the thesis was to provide foundations for a module within a system for networked defense currently under development. Military concerns have affected the thesis parameters somewhat—the condition of peer-to-peer architecture originates from military robustness needs, for example—but the thesis by no means prohibits civilian applications.

1.4 Report structure

The current chapter introduced the thesis topic and gave a perspective on general computer and communications development. It also presented the thesis goals and laid out the principal questions to be answered. Furthermore, it contrasted the thesis work with a parallel KTH thesis being developed at FOI.

Chapter 2, *Theory*, presents background theory from the two major research fields that the thesis uses: simulation and Computer Supported Collaborative Work (CSCW). It also presents the HLA and describes previous research efforts that have been made to integrate the HLA with collaborative software.

Chapter 3, *Method*, contains a brief account for the methodological considerations in the Master's project.

Chapter 4, *Design*, presents the design solution that the project proposes for the problems laid out in the introductory chapter, including the *key design questions* from Table 1-2.

Chapter 5, *Implementation*, gives a roadmap to the components of the proposed collaborative infrastructure's prototype implementation. It gives brief descriptions of certain visible front-end components and goes into some depth on background data management functions.

Chapter 6, *Experiments*, presents results of performance tests that were performed during the project to gauge the RTI's quantitative qualities as communication substructure.

Chapter 7, *Conclusion & future work,* summarizes the project's findings, discusses items of future work, and concludes the report.

2. Theory

The present thesis draws upon two research fields within computer science, computer-supported collaborative work and modeling & simulation. Part of the thesis goal is to apply results of the former onto practices of the latter. This chapter will give theoretical background information on these two fields.

2.1 Simulation

This section describes simulation and the closely related field *modeling*.

2.1.1 Simulation definition

A simulation is defined as "the imitation of the operation of a real-world process or system over time" (Banks, Carson, Nelson, & Nicol, 2001, p. 3). Although simulations do not necessarily require computers, computing advances in recent decades have made computer simulation a premier problem solver in a wide variety of fields. Simulations are used regularly in designing and testing new technology (cars, communication networks, nuclear systems, semiconductors etc.), in studying complex systems (the environment, street traffic, the human body etc.), in training (flight training, decision-making training etc.), in decision support (in business, in medicine, in the military etc.) in visualization of complex proposals (architecture, equipment design), to name a few applications (Banks et al., 2001; Obaidat & Papadimitriou, 2003).

It is often said that simulation can be helpful when analytical, or closed-form, solutions are impossible or impractical (more situations when simulation is helpful or not helpful are discussed below). If this is unclear, perhaps the following example will be useful.

Table 2-1 Example contrasting analytical problem-solving with simulation

• ***Problem****: How long does it take to drive between Stockholm and Gothenburg if there is no other traffic on the freeway?*

Analytical (or closed-form) solution:	[distance]	/	[speed]	=	[travel time]
	468 km	/	110 km/h	=	4.25 hours

The problem is inappropriate for simulation since an analytical solution is on hand.

• ***Problem****: How long does it take to drive between Stockholm and Gothenburg if the following is true:*

1. *I drive 110 km/h unless I have slow cars right in front of me*
2. *Every minute, I run a 50% risk of catching up to a slow car. If I do, my speed (and the risk of catching up to more slow cars) drops by one tenth*
3. *Every minute of driving behind slow cars, I have a 25% chance of passing. If I pass, I pass all of the cars right in front of me and resume driving at 110 km/h*

Analytical (or closed-form) solution: [complicated]

This problem is appropriate for simulation. While it is difficult to find an exact, analytical solution, it is relatively easy to enter the model into a computer and have the computer perform 100 or 100 000 "test drives" between Stockholm and Gothenburg. The average test drive time will be a good prediction of the real-life driving time.

2.1.2 Modeling

A model can be defined as a representation of a system for the purpose of studying the system (Banks et al., 2001). This includes physical models such as model airplanes and architectural models, as well as mathematical models such as the two car-trip models in Table 2-1. Important special cases of mathematical models are *conceptual models,* which focus on describing objects in a system, their attributes, and their relationships to other objects.

According to the general definitions of modeling and simulation, there is at least one simulation for every model and at least one model in every simulation.[1] For computer-based simulation, however, this is not true—some models cannot be incorporated in computer simulations. Furthermore, for some models that can be computer simulated, simulation is not an appropriate method at all. The case where an analytical solution is readily available has already been mentioned; cases where simulation would be too expensive or too imprecise are relevant as well.

The converse of the above statement is definitely true, however—there must be one model (at least) for every computer simulation. This model must be a mathematical model of the process the simulation aims to represent. Besides being mathematical, the model may either be dynamic or static, deterministic or stochastic, discrete or continuous. See Banks et al. (2001) for explanations of these terms in the context of simulation.

Certain computer-based simulations include not only computers but also other equipment and humans. This inclusion expands the range of models that can be incorporated in computer simulation. If a traffic planner for example wants to study the safety effects of cell-phone conversations in cars, she cannot do so using only computers since it is difficult to find a valid mathematical model for human behavior in this situation. Neither can she perform real-life experiments with this question since it would be dangerous. However, by using a computer, an arcade-game style car-simulator, and a person, this complex system can be successfully simulated. This simulation would include actual equipment (a real cell-phone might be used), physical models (a mock-up hands-free and mock-up steering instruments), mathematical models (the driving simulation program), and a human being.

Generally, involving people and equipment in simulation can offer two benefits: the ability to accept input from entities that would be difficult to model mathematically, and the ability to include real entities or physical model entities which possess specific physical characteristics that are relevant to the simulation.

Since model-building is such an important part of constructing simulations, many include it when describing simulation as an academic field. When this thesis refers to simulation as a field, others may equivalently have referred to *modeling and simulation*, or *M&S*.

[1] Actually, since the simulation definition refers to "imitation over time", so called *static models* cannot be associated with a simulation. *Dynamic models*, however, can always be associated with a simulation.

2.1.3 Distributed simulation and parallel simulation

The terms *distributed simulation* and *parallel simulation* are related but not equivalent. Distributed simulation can be defined as "a single 'run' of a simulation program across multiple processors" (Fujimoto, 2003, p. 124). Parallel simulation refers to running distributed simulations with the objective of maximizing execution speed, often on a tightly coupled computer system such as a supercomputer or a shared memory multiprocessor (Fujimoto, 2003). Parallel simulation can also be executed on clusters of workstations, although this is still relatively uncommon.

Fujimoto (2003) discusses two other motivations for distributed computing that exist alongside the motive of speed. One motivation concerns making simulations distributed in order to integrate simulation entities that cannot be practically put on a single machine because of geographic separation. This applies primarily to distributed interactive simulation (refer to 2.1.4), where people and equipment participate in the simulation and the people and the pieces of equipment may be located far apart. An example of this is a simulation for military training where a commander may be located by a computer, responding to simulated enemies, while a group of air-force pilots are interacting with the simulation from inside their flight simulators at a different location. A second example might be kids playing multiplayer online games simulating adventure or action scenarios.

Another motivation for distributed simulation arises when security concerns prevent simulation entities from being moved to a single machine. This applies in military simulations (and possibly in some business simulations) where the internals of simulation models may contain highly sensitive data. For example, a distributed military simulation may be run by a field commander who wishes to evaluate the effects of calling for air-support or for a cruise-missile strike. The model describing airplanes may be stored in an aircraft carrier computer system whereas the cruise-missile model may be stored at an air-command center. These models will contain sensitive internal data that should not be downloaded to the field-commander's workstation, for risk of falling into enemy hands.

2.1.4 Distributed interactive simulation

There are three basic types of computer-based simulations (referred to as "simulations" in the remainder of this thesis). Most of the simulations discussed thus far are *pure simulations* (Zhao & Georganas, 2001). Pure simulations may be either distributed or non-distributed. Within distributed simulations, one may define the two subcategories *people-in-the-loop simulations* and *equipment-in-the-loop simulations* (Zhao & Georganas, 2001). These have been introduced in previous sections and will be made more rigorous in the present section. The two terms are collectively termed *distributed interactive simulations* (refer to Figure 2-1 for a graphical classification scheme).

Pure simulations are simulations where the computer is solely responsible for determining and maintaining system state and for generating successive simulation events. People interact with the simulation only by supplying initial

parameters (possibly), starting the simulation, and inspecting simulation results. If the Stockholm-Gothenburg driving example above were to be simulated, it would be a pure simulation.

People-in-the-loop simulations allow human users as well as the computer to generate simulation events. Training simulations are nearly always of type people-in-the-loop. Humans may be connected to the simulation via Virtual Reality simulator studios, vehicle simulators (such as the ones used in pilot training), a PC, or a more limited interface device such as a GPS transmitter.

Equipment-in-the-loop simulations contain at least one piece of actual equipment connected to the simulation. The piece of equipment may respond to input from computer-generated simulation entities, or it may itself generate events. Examples include airplane instrumentation testing, radar station testing, weapons development and weapons testing.

The three simulation types can be found in combination as well.

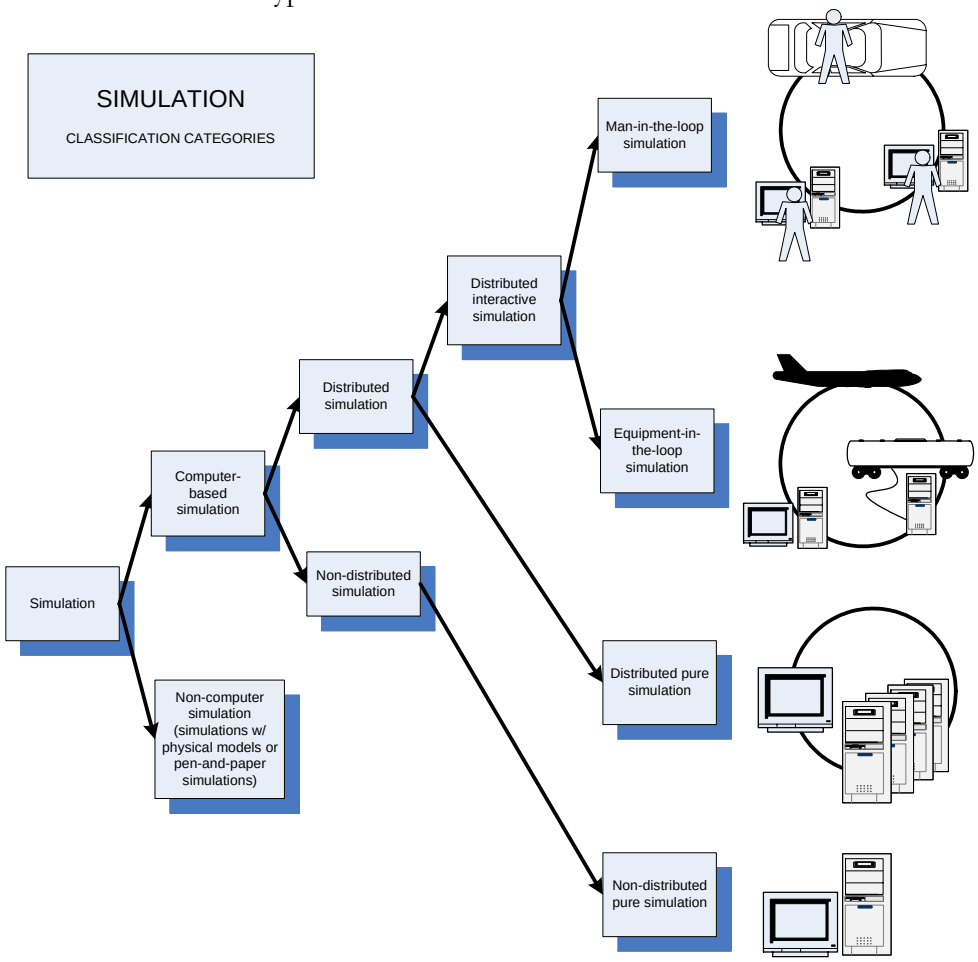


Figure 2-1 Names for different categories of simulations

2.1.5 How simulations are built

Computer simulations can be constructed using general-purpose programming languages, specialized simulation languages, or integrated simulation environments.

General-purpose languages such as Java, C, or C# give the programmer full flexibility regarding how to represent models and how to define event generation and event handling.

Specialized programming languages such as GPSS make simulation programming more efficient by providing programmers with a range of ready-made simulation constructs. They also provide event-scheduling algorithms, statistics tracking and report generation (Banks et al., 2001). Some of these services are offered in libraries to general-purpose languages as well.

Integrated simulation packages such as Arena, AutoMod, Quest, Extend, Taylor ED, and Witness grant the programmer yet more efficiency at the cost of some flexibility (Banks et al., 2001). These packages contain numerous models and even complete simulations, for the user to build from. Often, a graphic user interface allows users to manipulate simulation parameters, combine simulation elements, and inspect 2D and 3D graphical views of simulations. In some cases, packages let users access and modify the actual code that makes up simulations. This may be Java-code, GPSS-code, or code in a proprietary package language. Packages that offer this option constitute no sacrifice of flexibility.

2.1.6 High Level Architecture (HLA)

As discussed above (2.1.5), simulations may be built using many different languages. Even simulations implemented in the same language can be very different with respect to methods of constructing models, states, events etc.

The *High Level Architecture* (HLA) defines a framework for component-based simulation systems (Kuhl, Weatherly, & Dahmann, 1999). The goal of the HLA is to provide interoperability between simulations implemented in different languages and/or using different methods. A "happy side effect" (Kuhl et al., 1999, p. 20) of the quest for interoperability is that the HLA also provides extensive support for distributed simulation. A second happy side effect of interoperability is the possibility for recombination and reuse of simulation components.

Below are a few important points about the HLA are presented, taken from Kuhl et al. (1999). For a complete discussion, refer to this standard work.

- The HLA is a software architecture.
- The HLA prescribes how software for certain parts of a simulation program should be written in order to enable the simulation to interoperate with other simulations.
- The HLA defines the interface of a support system called the *Run-time infrastructure (RTI).* An RTI is assumed to be present when HLA simulations are run.
- The HLA is not the RTI, nor does the HLA provide it, nor define its implementation. The HLA defines RTI interface, and RTI implementations are provided by various independent vendors in various languages.

The HLA introduces various terms for its architecture elements. The following is a summary of the most important terms taken from Reid (2000, p. 2):

- **Federate**: An individual simulator application or executable component. These are the independent simulators, which the HLA integrates into a larger collaborative simulation.
- **Federation**: A simulation composed of two or more (often many more) federates integrated together.
- **Federation execution**: A session in which a federation is running, usually as a distributed system.
- **Federation Object Model**: The common object model that describes the data shared between federates within the federation. […]
- **Simulation Object Model**: The object model that describes the data that an individual federate shares with the federation. It also contains some other interfacing information about the federate. In some ways, the FOM is a subset of the collection of SOMs defined for the various federates in the federation. […]
- **Object**: In a conceptual sense, an HLA object is an entity that the simulation models. HLA objects represent "actors" that play in the simulation. In a literal sense, an object is a container for shared data that are created by a federate during the federation execution and persist for the duration of the federation execution or until deliberately destroyed. The FOM defines all classes of object and any federate that wishes to publish or subscribe to an object must also compatibly define that object in its SOM. HLA objects store their data in attributes.
- **Interaction**: An HLA interaction is essentially a broadcast message that any federate playing within the federation execution can send or receive. […] Interactions carry their data in parameters.
- **Runtime Infrastructure (RTI)**: The software that implements the HLA interface specification and runs the federation execution.
- **Object Model Template (OMT)**: The standard template used for defining the form, type, and structure of the data shared within the federation and for some other interfacing information. All FOMs and SOMs are documented in accordance with the OMT. […]

As mentioned above, the RTI provides various services for federates. These services come in six classes, which are described as follows in Moradi & Ayani (2003, p. 465):

- **Federation management**: Provides functions for creating, modifying, controlling and destroying a federation execution. After creating a federation execution, federates join and resign the federation as they wish as long as it serves the purpose of the simulation.
- **Object management**: Federates create, modify, or delete objects and interactions through Object management services.
- **Declaration management**: Provides federates with the ability to express their intentions or interests in publishing or subscribing to object attributes and/or interactions.
- **Time management**: Provides a flexible and robust means to coordinate events between federates.

- **Ownership management**: Provides federates with the possibility to exchange ownership of object attributes among themselves.
- **Data distribution management**: Provides mechanisms for efficient routing of information among federates.

The HLA was developed by the Defense Modeling and Simulation Office (DMSO) of the United States Department of Defense. It is a successor to a more limited DMSO simulation framework called *Distributed Interactive Simulation* (DIS). Since 2001, all American military simulations are required to be HLA-compliant. Since 2000, the HLA is also a civilian IEEE standard (IEEE 1516) and the DMSO is working actively to achieve a technology transfer of the HLA to civilian actors and to promote HLA use in civilian applications. The HLA has not yet had its breakthrough in civilian applications, however. Some believe this is because the main services of the HLA (interoperability, distribution-support for simulations with secret internals) are primarily geared towards military needs, and are not worth the overhead in civilian applications. Others believe that civilian simulation is bound to grow significantly and that now that a strong civilian standard exists, the HLA will satisfy an emerging demand for reusability and interoperability in simulations in business, university, and entertainment (Kuhl et al., 1999).

2.2 CSCW, computer-supported collaborative work

This section describes the relatively young research field labeled *computer-supported collaborative work*. It gives a definition and summarizes major considerations within the field.

2.2.1 Computer-supported collaborative work definition

The conventional meaning of computer-supported collaborative work is obvious—using a computer to perform tasks in cooperation with others. Since the middle of the 1980s, the term also has a more specific meaning as the name of a multidisciplinary research field focused on this activity. In this role, the term is often abbreviated CSCW. One definition of CSCW is the following:

> "[CSCW is] an endeavor to understand the nature and characteristics of cooperative work with the objective of designing adequate computer-based technologies."
>
> (Bannon & Schmidt, 1991, p. 3)

In this thesis, the acronym CSCW will be used both in the sense being of a research field and in the conventional sense. Hopefully, context will make reference intentions clear.

2.2.2 Modes of collaborative work

Computer-supported collaborative work can occur in four modes (refer to figure 2.2).

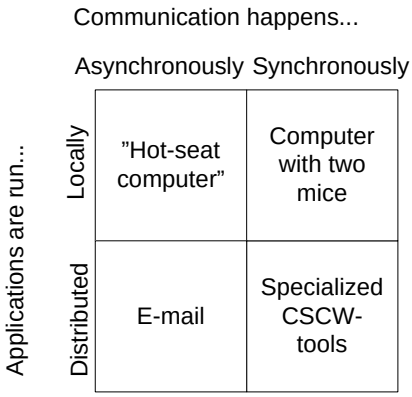


Figure 2-2 CSCW-tools may work in four different modes

Asynchronous, local collaboration would occur if two people were to try to collaborate on a project using a single personal computer, continually swapping seats at the keyboard. Little research is devoted to this mode of CSCW.

Collaboration with *distributed applications* employing *asynchronous communication* is a more common phenomenon. It includes e-mail—which of course is based on users having separate, distributed e-mail clients and where users communicate asynchronously (i.e. after sending a message, users do not have to wait passively for a response but can carry on with other activities). It also includes many internet software development projects, where users collaborate with software such as CVS to typically produce competitive, open source code. This type of CSCW has received some research attention (eg. Yamauchi et al., 2000; Bowen & Maurer, 2002), and certain findings are presented below (see section 2.2.5).

Collaboration with *local applications* based on *synchronous communication* is another area being researched. The type of system called Single Display Groupware, especially, is being studied by various researchers (eg. Tse et al., 2004; Tollinger et al, 2004; Stewart, Benderson, & Druin, 1999). Single Display Groupware is based on several users having individual input devices to a single computer or a single interactive table top. Users can immediately see and react to other users' actions, so communication is synchronous.

The focus of this thesis lies on the fourth mode of CSCW, which concerns *synchronous, distributed collaboration*. This is the bottom right quadrant of Figure 2-2. This mode of software includes general Groupware and CVEs, which will be discussed next. When the present chapter mentions CSCW-software, it focuses primarily on the synchronous, distributed category.

2.2.3 Groupware and CVEs

Computer systems designed to support CSCW are often labeled collaborative systems or *groupware* (Prakash, Shim, & Lee, 1998). Groupware comes in many different forms, including full-fledged 3D-implementations where users can inter-act in virtual reality worlds.

Groupware featuring 3D are commonly known as *Collaborative Virtual Environments* (CVEs) and has attracted substantial attention in years past in parts of the CSCW community (Churchill, Snowdon, & Munro, 2001; Zhao & Georganas, 2001). However, a recent trend has been to move away from 3D in favor of "toolbar and 2D community products" (Churchill et al, 2001, p. 100), particularly in business-oriented groupware.

2.2.4 CSCW in practice

Eseryel et al. (2002) list tools supporting CSCW. The most successful tool by far is electronic mail. Although e-mail might not match most people's definition of groupware, it is an enabler of CSCW and it outclasses other CSCW-tools with regards to usage volume and impact. Other tools are:

- Domino Office Procedure System
- Lotus Notes
- Xerox Docushare
- SevenMountains Integrate

(Eseryel et al., 2002)

These systems fulfil the requirement that "an integrated CSCW-system should address four basic areas of concern: Communication, Collaboration, Coordination, and Control (Ganesan et al., 2001)" (Eseryel et al., 2002, p. 131). Some newer systems that also fulfill the requirement are:

- eGroupware
- XOOPS Dynamic Web CMS
- CVS (limited Communication support)

Examples of systems that address only a few of the areas, but still qualify as CSCW-tools are:

- e-mail
- Microsoft NetMeeting
- ICQ
- MSN Messenger
- LAN/WAN file sharing capabilities

2.2.5 Various results from CSCW research

CSCW uses results and methods from a range of disciplines including the study of distributed systems, communication, human-computer interaction, artificial intelligence and social theory (Severinson-Eklundh, 1998). The remainder of this section will describe some research results that may be relevant to the forthcoming discussion.

The conclusion of Eseryel et al. (2002) that communication, collaboration, coordination, and control are important aspects of CSCW systems has already been mentioned. Furthermore, many authors place additional demands on "communication" by claiming that CSCW-systems need to afford their users *awareness* of each other, and of each other's actions (Prakash et al., 1998; Churchill et al., 2001; Li & Li, 2002).

It is also seen as important that users trust that other users share their view of the collaborative work area (Churchill, 2001). This is succinctly communicated with the term WYSIWIS (what-you-see-is-what-I-see). Under certain circumstances, however, this requirement must be relaxed in order to allow personalized screen layouts, particularly if users collaborate with differently sized devices (Marsic, 2000).

Regarding actual instances of collaboration, Ahmed, Kumar & Tripathi (2003) have observed that the success of collaborations is not merely a technological matter. They find that collaboration users, just as people in many other settings:

> "may compete to maximize their personal gains. A user can only be trusted to maintain integrity of collaboration entities […] if he/she has vested interest to do so"
>
> (p. 1)

In synchronous distributed CSCW (refer to Figure 2-2), consistency issues have been widely studied. Consistency issues deal with the problem of how to maintain shared data up-to-date with all users. Two approaches exist: *centralization (client-server)* or *replication (peer-to-peer)*. The conclusions on centralization versus replication in CSCW mirrors conclusions from general distributed computing; centralization is easier to implement and performs faster for certain operation (eg. searches) whereas replication is more fault-tolerant and faster in other capacities (eg. data manipulation when data is administrated close to the local machine) (Prakash et al., 1998).

Furthermore, CSCW-researchers have identified a number of subtle human capabilities that come naturally in co-located collaboration but that risk being lost when people collaborate remotely via computers unless they are addressed. This includes the ability to rapidly survey what people around you are doing and whether they are available, the ability to engage in unforced social contact, and the ability to politely dismiss people when one is busy (Churchill et al., 2001). Catering to these capabilities, and similar ones, is an important design element in synchronous distributed CSCW software.

Research on online open source software development, a special case of asynchronous distributed CSCW, has identified some success factors in this type of "narrow band" collaboration. Various effects stem from the fact that work precedes coordination. Typically, collaborators read "ToDo-lists", then individually complete tasks they choose for themselves, and finally submit work for integration with the main project. This stands in contrast with conventional work organization, where coordination precedes work. It has been found to encourage innovation, spontaneity and new member recruitment; and discourage procrastination. Narrow band media has also been found capable of affording effective and efficient communication when used wisely (Yamauchi et al., 2000).

Finally, it has been emphasized that CSCW systems need to prevent cognitive overload in users, that they need to be concerned with user security and user integrity, and that they need to be scalable.

2.3 New applications of simulation and CSCW

This section will give a brief account of some previous work that has been done with the objective of introducing CSCW into simulation. It will also look at previous efforts to extend the simulation architecture HLA to other purposes than simulation.

2.3.1 Example of a collaborative simulation application

Ayani & Dharma (2003) implemented a web-based collaborative educational simulation application. The application allowed multiple users to chat, and to view either a CPU-caching simulation or a two-server queuing system simulation. Only the initiator of each session could modify simulation parameters and control simulation flow.

The application did not use a ready-made architecture such as the HLA but constructed its own client-server architecture for messaging and synchronization. Clients were Java applets that displayed simulation state and handled user messaging. The initiator's client was the one that actually computed simulation state and forwarded state information to the server. The server was a Java process that received chat messages (from all users) and simulation messages (only from the initiator), synchronized those messages, and forwarded them to clients. TCP/IP was used as communication protocol.

2.3.2 Example of extending HLA to gaming

Vuong et al. (2004) designed and implemented an ambitious framework for multiplayer online gaming on top of the HLA. The framework features a server-based game lounge which players can log on to from PCs or cell phones. In the lounge, players can chat and launch games. The framework also features a test implementation of a chess game, and an implementation of a membership server which keeps records of all members and all completed game results.

The framework, which its authors label *Scalable Collaborative Environment,* utilizes an HLA RTI for communication between game players, observers and membership servers. The players' client application and the observers' client application are partly implemented as HLA federates, and game sessions are partly implemented as HLA federations. The RTI handles transport, traffic filtering and synchronization.

2.3.3 Example of extending HLA to virtual shopping

Zhao & Georganas (2001) implemented a collaborative virtual shopping mall using the HLA. The shopping mall was represented as a virtual environment with 3D, person-like avatars representing shoppers and with various 3D objects representing shopping items. Shoppers were implemented as HLA federates and shopping sessions were implemented as HLA federations. Communication was handled by an HLA RTI.

The authors go on to make a thorough evaluation of the HLA as an enabler of collaborative virtual environments. Conclusions include the following.

- The HLA can provide a shared sense of space for collaboration participants though its federation concept.
- The HLA can support awareness though its Object Management service.
- The HLA shows "acceptable performance" in supporting a shared sense of time. Data Distribution Management services are helpful in reducing response time. However, question marks linger for HLA performance over the internet.
- The HLA supports dynamic entries and exits to the collaboration through its Federation Management service. However, there is one limitation: any special characteristics in an arriving collaborator must have be predefined in the FOM in order to be used.

(Zhao & Georganas, 2001, p. 19-20)

3. Method

The present investigation employed various methodologies. This chapter is a brief account of some of these.

The duration of this Master's project was twenty weeks. It involved a literature review, informal interviews (predominantly with personnel at the Swedish Defence Research Agency and the Royal Institute of Technology), spiral-model software design, demo implementation, and performance testing. Performance testing compared the speed of Run-Time Infrastructure (RTI) communication with the speed of regular socket communication, and was performed in a mini-lab setting at the Defence Research Agency.

3.1 Progress of work

The thesis work consisted of four major tasks: literature review, application design, application implementation, and performance/feasibility experimentation. The four tasks were intermingled over the course of the project. Method considerations for three of them will be described more closely in the following subsections.

3.1.1 Application design

The design effort was aimed at providing a complete design solution for the collaborative framework application that is proposed in this thesis. Particular focus was given to design of the management of users in group constellations. Design was largely an iterative process, although the better part of the design effort was completed before implementation started. For presentation of the design solutions, the applications Microsoft Visio and Eclipse 3.0 with the hyperModel plug-in were used.

3.1.2 Demo implementation

The present thesis includes a demo implementation of the system it proposes. The primary purpose of the demo implementation is to demonstrate feasibility and to provide a code-foundation for performance testing. As usability and GUI-development were not primary concerns, only limited user tests were performed.

The demo application was implemented in Java Standard Edition. Some incompatible code caused the project to be released in two versions: one for JRE 1.4.2 and one for 1.5.0. The demo application is included in the CD-ROM distributed with original prints of this Master's report. Look for the CD-ROM in the pages directly following the reference list.

3.1.3 Experimentation

Approximately two weeks were spent on performance testing. This task was completed in collaboration with a related Master's project by Liljeström (refer to Figure 1-1). The objective was to compare the speed of HLA RTI communication with direct, low-level socket communication. The intention was not to find the true speed of an HLA RTI, but rather to find how the RTI performs relative to a standard socket method of communication under a variety of conditions.

The experiment was set up around point-to-point communication on pairs of PC:s on either a dedicated, single switch LAN or on a shared LAN with interfering traffic. Two pairs of PC:s were used, one pair of medium performance machines and one pair of high performance machines.

In several test sessions, a custom-made Java test program at machine A sent messages of various sizes, in various quantities, to another test program at machine B. In some tests, machine B returned the message upon reception while in other tests, machine B was set up to simply time reception intervals. The HRTimer package (Roubtsov, 2002) was used for precision timing.

Experiments were carried out at facilities of the FOI Swedish Defence Research Agency, and results are presented in Chapter 6.

4. Design

This chapter will present a design solution that addresses the questions and conditions presented in chapter one (re-listed in Table 4-1). Chapter 5 will describe how a portion of the design solution was implemented.

Table 4-1 Re-listing of the principal design parameters from chapter 1

Key design conditions

1. Employ a pluggable collaborative tool architecture
2. Manage groups for collaborations
3. Define an XML group information model
4. Employ peer-to-peer architecture
5. Use the High Level Architecture for distribution support

Key design questions

1. What kinds of distribution support will be enlisted by the HLA?
2. How will consistency be ensured in the collaboration?
3. How is a collaborative group defined?
4. How should definitions of collaborative groups be maintained?
5. How should logins and logouts to/from the collaboration be handled?
6. How should the interface to tools look, concerning general functions?
    - Handled in a parallel FOI/KTH study authored by Liljeström

In section 4.2, explicit answers will be given to key design questions 1 through 5. Before that, Section 4.1 discusses design at a more general level. This discussion will present how the key design conditions were satisfied and it will reveal the overall rationale behind the selected design solution.

4.1 Preliminary design findings

This section discusses key design conditions and design rationale by delving into how certain central issues were resolved and by presenting two use cases that were developed early in the Master's project.

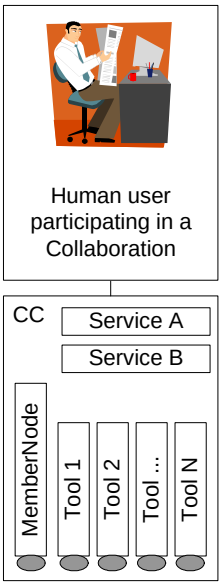


Figure 4-1 CC
The centerpiece of the proposed software is a peer-to-peer module called Collaborative Core (CC). CC offers various collaboration services and supports the plugging in of collaborative tools.

4.1.1 General features of design solution

Design conditions require a peer-to-peer system. The peer module in the selected design solution was given the name *Collaborative Core* (CC, Figure 4-1). The CC collects the various services offered to collaborations, and it is the unit where one

plugs in specialized collaboration applications such as collaborative versions of Emacs, Paint or Word. Such collaborative applications will be referred to as Tools in the remainder of this design presentation.

The principal service offered by the CC is transparent communication with other CCs, in order for users to collaborate. A user who is collaborating with a determined group of other users is said to participate in a *Collaboration*.

The CC provides communication through an HLA Run-Time Infrastructure—which was a condition of the design specification. The vices and virtues of using an HLA RTI will be examined in chapter 6. Two models were considered for organizing the RTI communication. One was to use a single HLA federation for every Collaboration that a user participates in (Figure 4-2). This implies that a user's CC contains one single federate. In HLA simulations, as mentioned in 2.1.6, federates are computer processes that may represent one or more simulation objects, or an entire simulation. The federate has access to various types of support for communicating with other federates. In the CC, a federate is not a simulation, but it makes heavy use of said communication support.

The option of having a single, general-purpose federate was ultimately discarded. Handling traffic from an unknown number of federates while at the same time managing group functions would lead to a complex design, without significant associated gains in performance.

The model that was in fact selected is presented in Figure 4-3. This model is based on coupling every Tool with an instance of a small generic Tool-federate. It also employs a specialized federate for group functions (detailed below). Group management occurs in the *CCFederation,* which is formed by the set of online *MemberNode* federates.

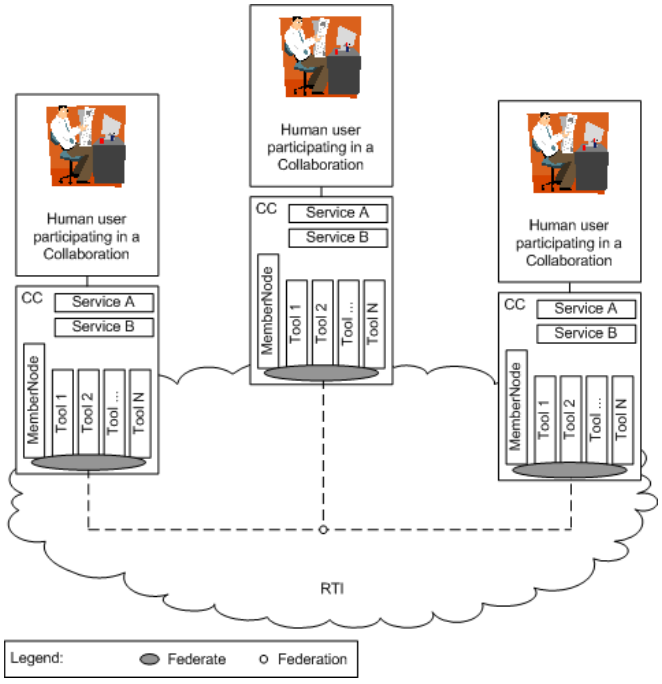


Figure 4-2 Communication based on a single general-purpose federate at each client

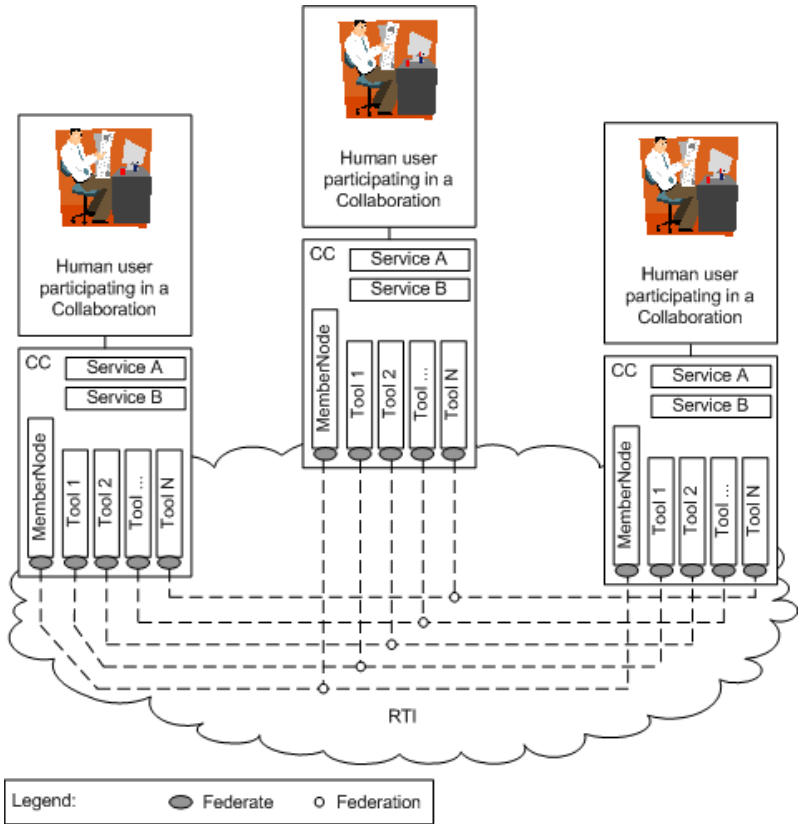


Figure 4-3 Communication based on several federate instances at each client
Each client has a specialized MemberNode federate (for group-management communication) and several instances of a generic Tool-federate (for Tool communication).

4.1.2 Multiple collaborations

Before going into how the key design questions were answered, an account will be given of how the proposed design handles participation in several Collaborations. This refers to the situation where a person may be in a collaborative session with her project colleagues while she is simultaneously in a different session with a group of union representatives. The selected design solution is to spawn one CC for every Collaboration dynamically (Figure 4-4). A structure called *CollabFrame* charged with managing the several CCs is introduced in the layer closest to the human user.

Multiple Collaborations may exist in one RTI, or may exist in separate RTIs. As mentioned above, an RTI is somewhat similar to a distributed operating system and it may be set up on several computers on a Local Area Network, or on several computers connected via the internet. The RTI offers synchronization and data transmission services to its connected computers. All users in a collaboration session must be connected to the same RTI to be able to communicate (refer to Figure 4-4 and Figure 4-5).

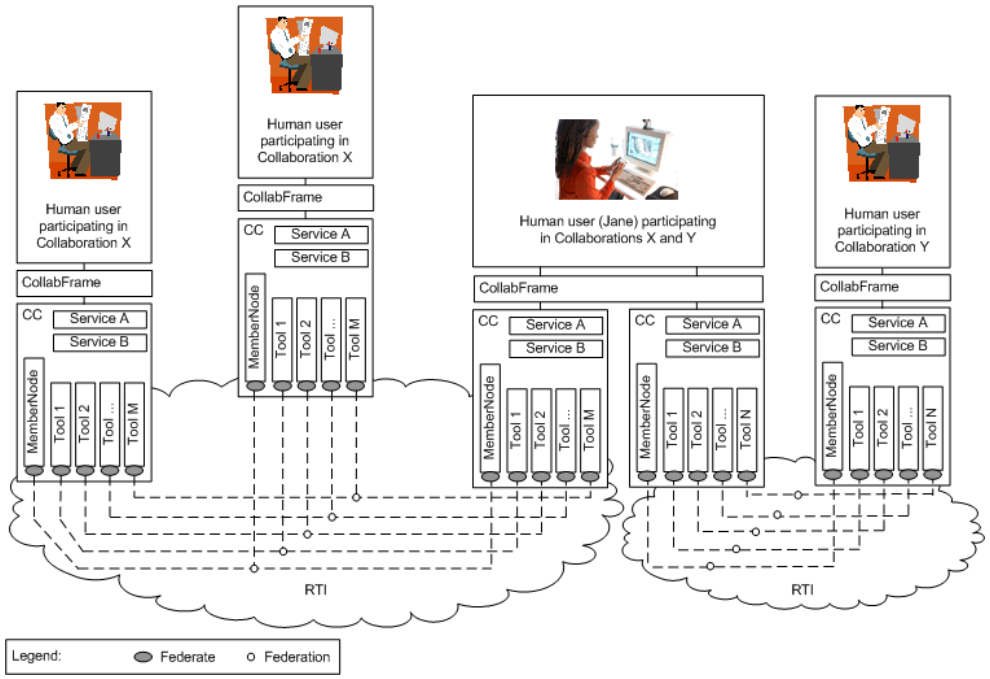


Figure 4-4 Multiple Collaborations, multiple RTIs

When a user such as Jane participates in several Collaborations at once, the CollabFrame transparently runs one Collaborative Core for every simulation. The several Collaborations may use separate Run-Time Infrastructures, or they may share an RTI (as in Figure 4-5).

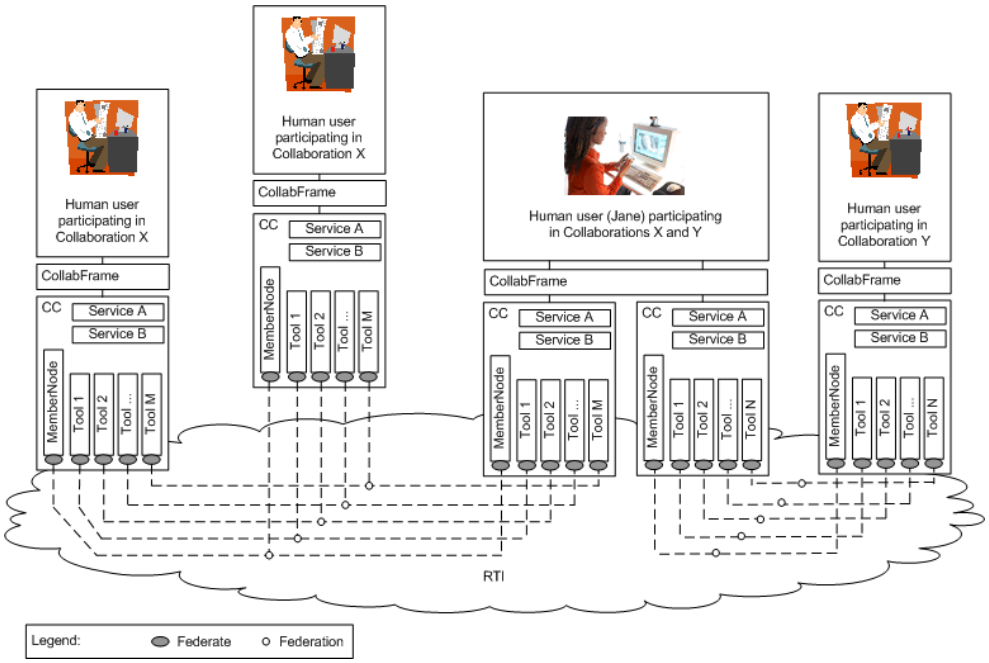


Figure 4-5 Multiple Collaborations, single RTI

In this situation, two Collaborations run on the same RTI. To users, and to (nearly) all parts of the various CCs, this situation is equivalent to the one in Figure 4-4.

4.1.3 Format of Tool support

Designing a Tool plug-in framework has thus far in the discussion been an unequivocal condition of the study. An exact mechanism for actual Tool plug-in is discussed at length in Liljeström (2005), and straightforward results are

produced. Certain design elements of the Tool interface were subject to discussion between the present author and Liljeström, however. Specifically, a choice had to be made between a framework supporting several modes of Tool operation, and a more general framework that defers some complexity to Tools. The question essentially concerned whether the framework should be intelligent regarding both Tool *status* and Tool *events*.

It was seen as clear that the "matter" of a Collaboration, in analogy with a distributed simulation, is made up of (1) the application states in all clients and (2) of the events that are exchanged between clients. The collaborative framework must hence have the capacity to relay events between collaborators, and the capacity to transmit the status of one collaborator to another, for example when a new user arrives, and support for saving client states. This implies a certain level of intelligence regarding states. Does the framework need this level of intelligence regarding events as well? That is: does it need to be able to interpret and store a history of events? The details of this discussion will be omitted, but among other things, event-intelligence could require as many as twelve modes of operation for collaborative Tools (refer to Appendix A). This would relieve Tools of some event-related responsibility but would also place stricter design demands on Tool developers. Ultimately, it was determined that the costs of assuming substantial event-related responsibility were greater than the benefits. Hence the CC design simply accepts state reports (through a Tool method such as *<Object>.getState()*), cares for state synchronization and event forwarding, and defers the brunt of intelligence regarding events to Tools.

4.1.4 Other design considerations

In order to build a solid design foundation, hypothetical use cases were developed in consultation with the thesis's FOI sponsor. Conclusions of the use cases include the following:

- The system would benefit from built-in mini-applications such as a text messenger and a contact list
- Users should get cues about Tool activity in the Collaboration
- The system must carefully consider user authorization
- The system needs to offer persistent state storage both when users are online (when peer-to-peer methods can be used) and when users are offline

Finally, the question of how to enable peer-to-peer based searches for Collaboration sessions was addressed. The solution for this rests on using an agreed-upon name for a Federation dedicated to searches. Whenever a user starts the framework, she transparently joins a Federation Execution named "__SysFederation". If no such Federation Execution exists (meaning that no other users are online), one is initiated. Thereafter, all Collaborations the user joins are posted in the *SysFederation* as HLA objects. This enables a newly arrived user to make a search of online Collaborations by inspecting the *SysFederation*'s HLA objects and avoids the need for a central searching service.

That concludes the general look at the design solution. Next, the key design questions will be answered.

4.2 Answers to key design questions

This section will explicitly answer the five design questions first outlined in Table 1-2 and repeated in Table 4-1.

4.2.1 What kinds of distribution support will be enlisted by the HLA?

The HLA will be utilized for three major distributed functions for the proposed collaborative application infrastructure: *transport management*, *time management* and *data filtering*.

Transport management refers to the fact that the application framework relies on the HLA entirely for data transport between collaborator computers. The HLA, or specifically, the HLA RTI, is responsible for message passing and to some extent responsible for data maintenance. The RTI, in its turn, relies on an underlying network (LAN or WAN) for actual transport. In many transport-related respects, the RTI performs equivalently to how a socket-based communication system would perform. A performance comparison between these two types of systems is presented in chapter 6. In aspects such as fault tolerance, consistency guaranteeing, and time management, however, the HLA offers support whereas pure socket-systems do not.

The proposed mechanism for transport management with regards to group management and Collaboration control—the core tasks for the present design—relies on HLA interactions. An interaction class with four data fields (HLA parameters) is defined for this purpose. The fields are *sender*, *messagetype*, *receiver* and *message*.

Time management service refers to making sure Collaboration events are time-stamped and providing support for event-ordering and reliable event delivery. Time management is often critical in simulation and in certain collaborative tasks, but not quite as critical in Collaboration control and group management. The facilities of HLA time management are therefore not leveraged in the present investigation. For an example of time management implementation in applications other than simulation executions, refer to Liljeström (2005). For a discussion of consistency (related to time-management) mechanisms in Collaboration control and group management, refer to section 4.2.2.

Data filtering refers to discriminating communication data with regards to recipients. The benefit, of course, is that network traffic is minimized. The RTI offers a class of services called Data Distribution Management (DDM) that fit well with this objective. Originally designed to filter simulation events from simulation objects that are not affected by them, DDM allows a designer to define the event space in terms of various user-defined dimensions. The designer may then specify that certain events are only relevant in certain ranges of certain dimensions. A dimension may correspond to a physical dimension such as "latitude", "longitude," or "altitude", or a logical dimension such as "communication priority" or "team affiliation". Dimension-bound events in a DDM-enabled federation execution will only be propagated to objects within relevant dimension ranges.

The actual filtering logic resides within the RTI implementation, and its mechanics are not defined in the HLA specification. When a federate at a certain machine calls on its RTI-stub (*RTI Ambassador*) to send an interaction or an object attribute update to a certain region, the RTI then works out which the receiving federates will be. Certain RTI implementations solve this distributed-computing task using a centralized approach. This class of RTIs implements the brunt of the RTIs intelligence in code residing at a single, central machine and deploys RTI stubs at remaining simulation machines, mostly concerned with making relays to/from the cental node. With centralized RTIs, it is clear that the Data Distribution Management does not perform traffic filtering to its full potential, as every message must first travel to the central node. Fully de-centralized RTIs that perform traffic filtering at the actual sources of traffic exist however, e.g. Object Web RTI (Fox, Furmanski, & Ozdemir, 1998) and Magnetar RTI (Vuong et al., 2004). There are also hybrid RTIs, e.g. pRTI, which centralize some functions but not others.

A straightforward DDM-approach is taken to achieve traffic filtering in the present Collaboration control design solution. User IDs are taken to be coordinates in a DDM dimension called *active-listeners*. If non-numeric user IDs are used (such as by external mandates), a hashed ID number is used. System events, messages, and other collaboration events that are directed to a particular set of recipients are then associated with the region composed of the various relevant locations in the *active-listeners* dimension. If the execution uses a distributed RTI-implementation, traffic filtering will then be transparently carried out at each message source, and network traffic will be minimized.

4.2.2 How is consistency ensured in the collaboration?

Ensuring consistency ranks among the most important tasks in distributed computing, if not in all kinds of computing. Parts of the HLA are explicitly geared towards ensuring consistency in distributed simulation, and its facilities can be utilized for consistency for the present CSCW purposes as well. Two design options are available:

I. The Collaboration's work is represented as HLA objects. Communication in the Collaboration is represented as either interactions or attribute updates. Consistency support provided through time managed attribute updates and interaction transmissions.
II. The Collaboration's work is represented as objects internal to Tools. Tools must be prepared to on demand reveal their state in an agreed-upon representation such as XML. Communication in the Collaboration is represented as interactions. Consistency support is provided through time managed interaction transmissions.

Option I leverages the built-in HLA capabilities more than option II does, but in doing so, also ties the CC more tightly to the HLA specification. It also ties the Tool developer closer to CC technology. It was determined early on in the Master's project that such tight couplings were undesirable. Option II was hence selected for being more component-oriented and more standards-oriented. Standards-orientation lies in the fact that option II merely requires

Tools to be able to reveal their state as XML at certain times, whereas option I requires Tools to constantly store their state as HLA objects. Figure 4-6 and Figure 4-7 expand on the idea of reducing requirements on Tool designers.

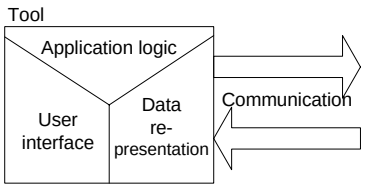


Figure 4-6 Simple model for distributed applications
The figure shows a simple four-category breakdown of the principal logical elements that make up a distributed application.

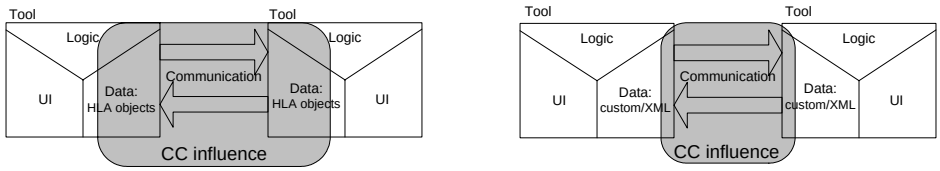


Figure 4-7 A Tool designer's perspective on option I and option II
If the CC requires Tool data to be represented as HLA objects as in option I, Tool developers' design freedom is undercut without an associated significant decrease in workload. The right part of the figure attempts to illustrate option II influencing (i.e. imposing requirements on, or assuming responsibility for) communication but leaving the Tool designer greater freedom with regards to application logic and data representation.

The discussion of this section so far has addressed the question "how is consistency ensured in the collaboration" in the sense of consistency with regards to internal collaboration data. Consistency is a key issue with regards to Collaboration metadata as well, even though Collaboration metadata undergoes less intense modification. The metadata discussion has been deferred since what the metadata should be has not yet been spelled out. While metadata contents will not be revealed until the next section, some conclusions can be drawn on consistency-preserving schemes using an assumed metadata entity.

Using the following assumptions (which section 4.2.3 will confirm as true), one can draw conclusions on what would be a viable consistency protection mechanism:

- Metadata can be completely visible to individual collaborator
- Metadata modifications occur as result of actions of one peer at a time
- Read-write conflicts do not cause fatal errors; write-write conflicts do
- Metadata is modified intermittently

If "intermittently" is taken to mean "typically with intervals of seconds or tenths of seconds" rather than, say, "every five milliseconds", the above enumeration leads to the conclusion that a writer's lock on metadata will provide sufficient guarantees for consistency. Such a lock can be implemented in HLA (Liljeström, 2005).

4.2.3 How is a collaborative group defined?

The definition of collaborative groups is a central design issue. The definition consists of the data that a computer would need to have in order to completely describe a group of people collaborating around some goal. This might be termed *Collaboration metadata* or *Collaboration Description*. The remainder of the discussion will use the latter term.

The question was addressed in the following two senses:

- What data should the Collaboration Description include?
- What should the Collaboration Description's life-cycle look like?

The Description's life-cycle was investigated first, as it has certain implications for what data to include in the definition. An appropriate life-cycle should provide support for dynamic creation of groups of people who want to collaborate, support for groups that are non-transitory and support for live entry and exit to collaborations without noticeable adverse effects to other users.

The definitions in Table 4-2 were found useful when designing the Collaboration Description (some of the terms have been used earlier).

Table 4-2 List of terms used in Collaboration design

Collaboration	A group of people engaged in a collaborative activity with each other using some form of supporting software, and their associated work.
Collaboration Description	An object or document containing all the data that define a Collaboration.
Collaboration session	A session of Collaboration activity. In a Collaboration session, one or several participants are online, interconnected and engaged in collaborative activity through their computers.
Member	A person/user that participates in a Collaboration.
join/resign Collaboration	Permanently enter or leave the Collaboration .
login/logout from Collaboration	Temporarily enter or leave the Collaboration (such as by going online or offline).
Tool	An application that is used jointly by Members.
Applicant	A person who wishes to become Member.
Application Document	The document that Applicants must submit when they wish to become Members.
Collaboration repository	Central persistent storage facility for Collaboration Descriptions and Tool work data. May be implemented as a folder in a distributed file system.

Using the first few terms of Table 4-2, a powerful scheme for the Collaboration Description life cycle can be constructed. Figure 4-8 shows such a scheme.

How to store the Collaboration Description throughout the life-cycle is another important design question (one which also will be explored in the implementation chapter). With this question, two of the design parameters actually find themselves in a conflict that needs to be resolved. First, it was stated that the system should support non-transitory Collaborations. This implies that Collaboration data, including the Collaboration Description, are preserved even when no Members are directly linked to one another. This, in

turn, implies that a central storage exists, accessible to all Members whenever they return to online status and wish to resume a Collaboration. The wish for a central storage is in conflict with the condition that the design be developed according to peer-to-peer principles.

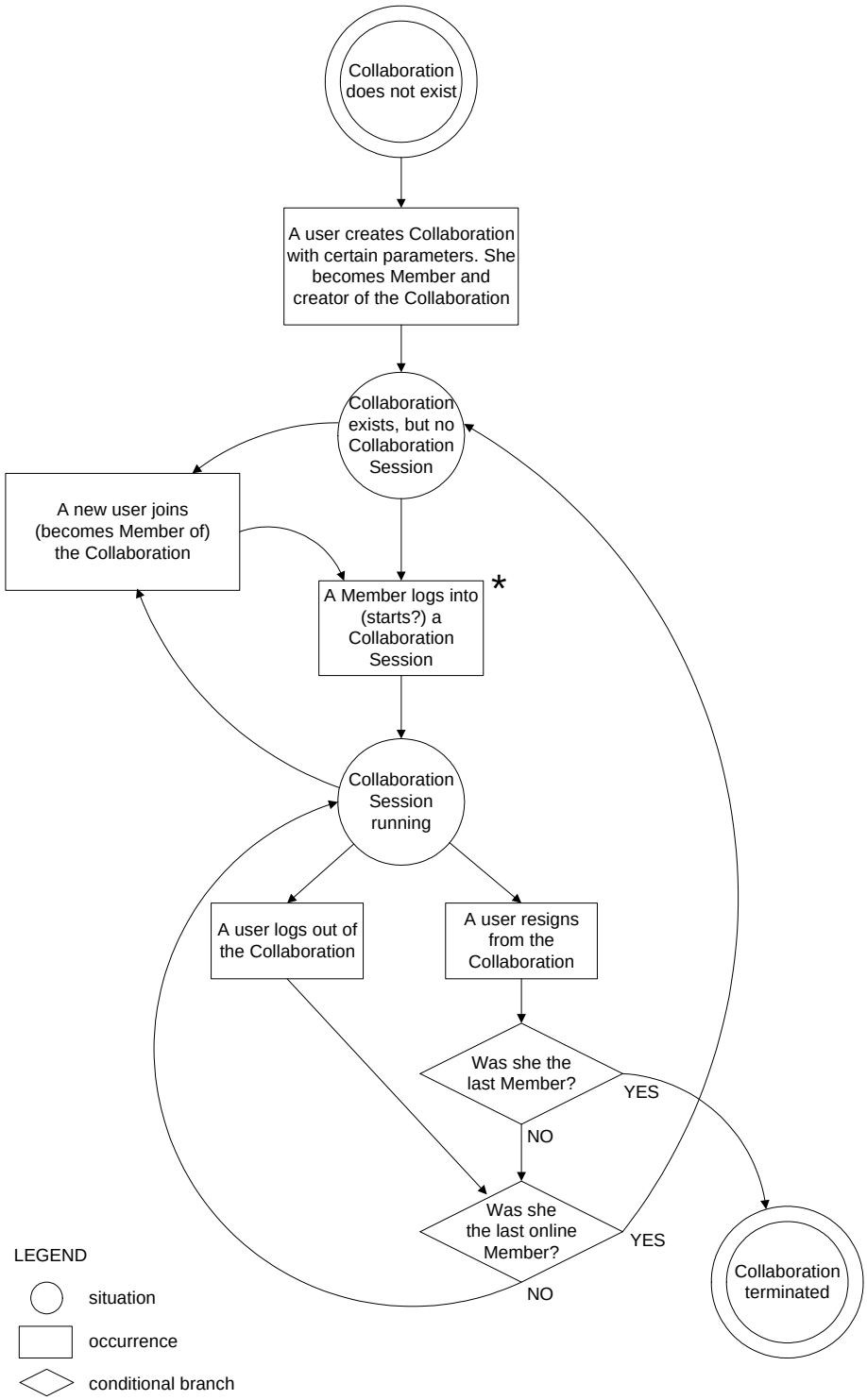


Figure 4-8 Life cycle of a Collaboration
The figure shows the various stages in a Collaboration's life cycle.
*At the starred box, the member logging in will start a new Collaboration session if she is the only Member online. If not, she will enter an existing Collaboration session.

Figure 4-9 details the storage/maintenance implications for the Collaboration Description for every stage in the Collaboration life-cycle. The figure mentions a "persistent storage" that will become necessary whenever no Members are online. The workaround to the apparent conflict between the peer-to-peer condition and the persistence preference is to use a distributed file system for persistent storage. The implementation of a suitable file system is investigated in Baymani and Strifeldt (2005).

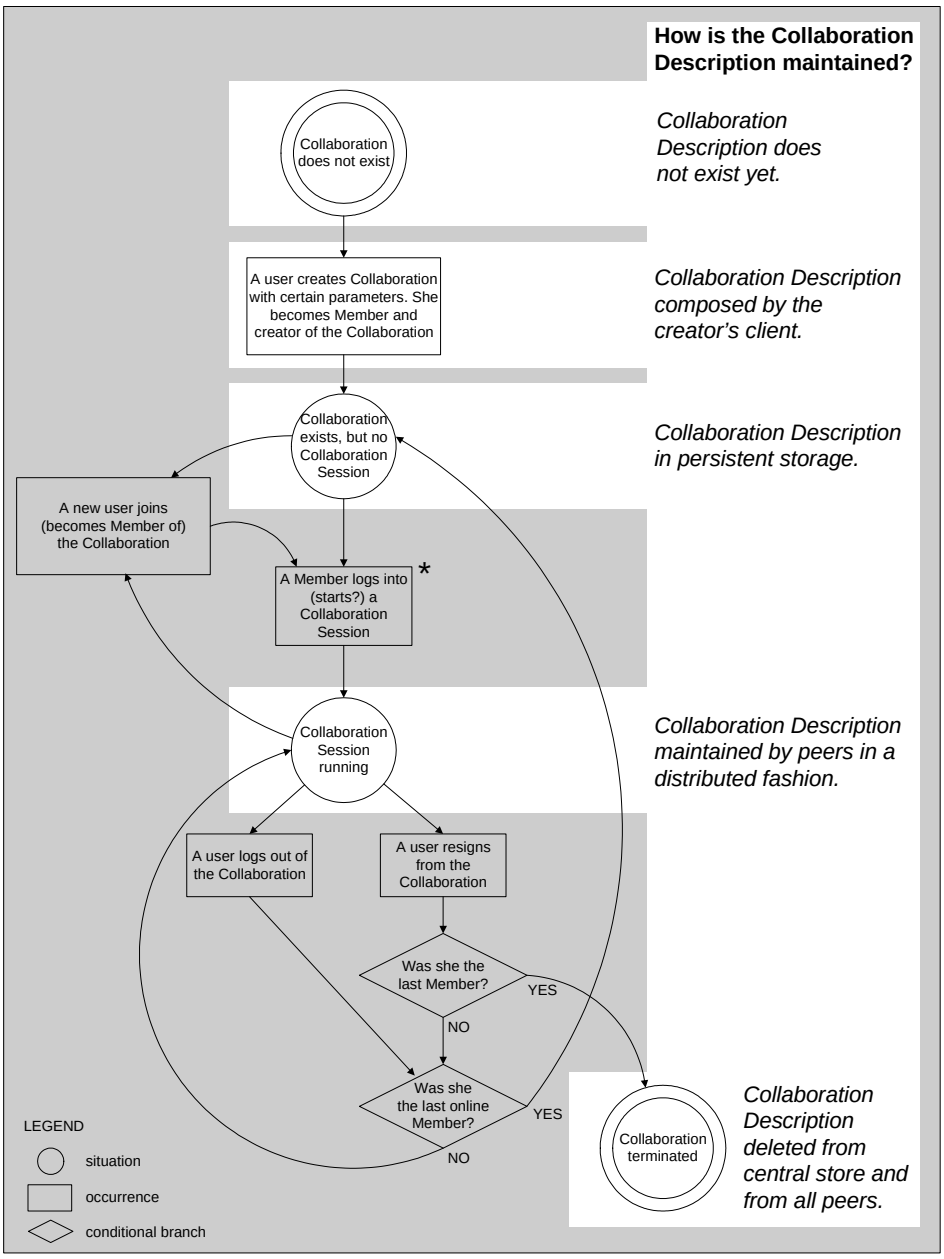


Figure 4-9 Life cycle of a Collaboration and its associated Collaboration Description

The figure details how management of the Collaboration Description occurs throughout the life of a Collaboration. The wish for non-transitory Collaborations and the condition of peer-to-peer design are fundamentally at odds here. The proposed resolution is to use a distributed file system for persistent storage. Such a file system for e.g. a collaborative platform is investigated in Baymani & Strifeldt (2005).

* Star explanation in Figure 4-8.

With the life-cycle of the Collaboration Description established, the question of what data the Description should include can now be addressed. It is clear that the description should contain information about the following, at least:

- Members
- Tools
- Current session (or last session, if none is currently running)

The information about Members is to include identification data, settings data and role data, and the information about Tools is to include version data.

Aside from these basic features, additional sets of data were placed within the Collaboration Descriptor. These include Member window-data to allow desktop layout sharing, signup handling data, and Member role data on a Tool-by-Tool basis. The resulting Collaboration Description is shown in Figure 4-10.

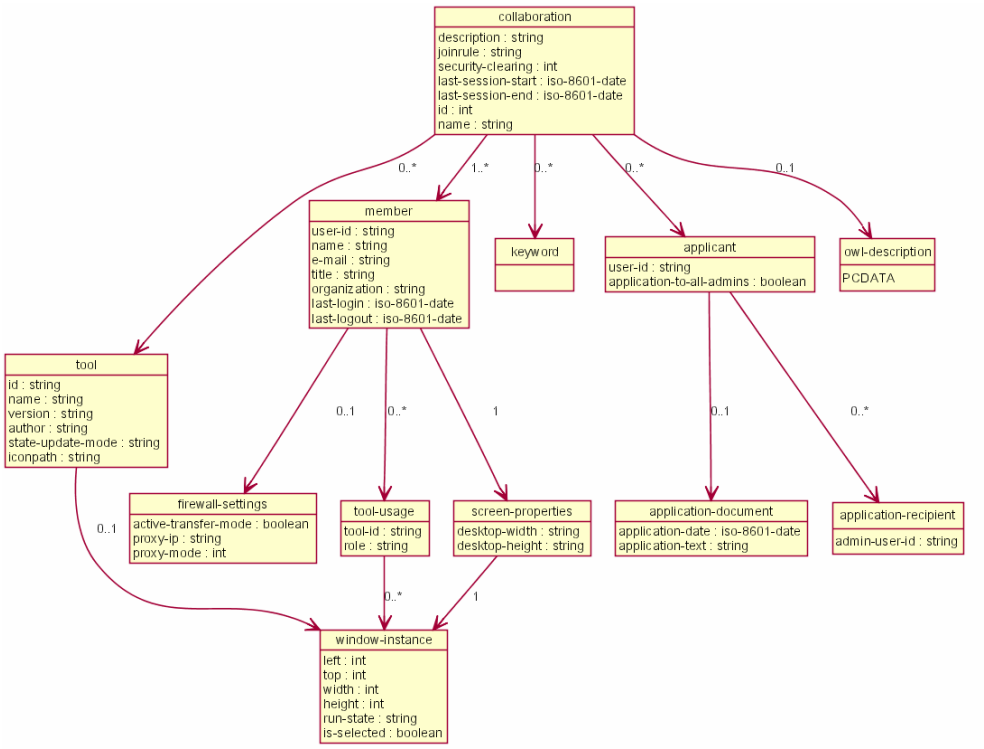


Figure 4-10 Collaboration Description information model

The above is a UML model for the information in a Collaboration Description. Chapter 5 describes how this model was implemented using XML.

In the above rendering, the Collaboration Description reflects all relevant information about a Collaboration, except for internal Tools settings and Tool working material. Handling Tool working material is outside the scope of this design, but the design might very well be extended to include Tool settings. This could favorably be accomplished by adding sub-classes to the UML class *tool-usage*. Other such extensions are also possible.

4.2.4 How should definitions of collaborative groups be maintained?

The maintenance of the Collaboration description is outlined in Figure 4-9. Recapping, when a Member is offline she has a set of Collaboration Descriptions stored locally. These are used to retrieve connection information and to connect to (or to start) Collaboration sessions. Once in a session, the Member receives the most recent version of the relevant Collaboration Description from one of the online Members. After that, any actions that the first Member or any of the other Members take to modify the Collaboration Description are propagated to other Members's local versions of the Description. An HLA-implemented circular writer's lock prevents write-write conflicts.

4.2.5 How should logins and logouts to/from the collaboration be handled?

The following describes non-authentication related procedures for login and logout to Collaborations in the present design. As authentication and security in collaborative systems is studied extensively elsewhere at FOI, these issues were excluded from the present Master's project.

In the first step, the Member's MemberNode federate connects to the Collaboration session's group management federation, goes through an authorization process, and then receives access to the distributed Collaboration Description. Upon receiving this access, the Member client will set the attribute *collaboration::member::last-login* to the present time. If the user was a Member before, the Collaboration Description will already contain a profile on the user and data on the Tools she used during her last session. If any of these items have changed during the time offline, the user's MemberNode will effect these changes on the Collaboration Description directly after login.

Next, a number of Tool-instances are launched by the framework. Recorded in the Collaboration Description, these may be the Tools the Member was using her last session, or they may be the default set of Tools for her or for the Collaboration. For each Tool, a ToolFederate is launched and an up-to-date version of the Tool's work state is retrieved from the Collaboration session (i.e. from an online Member). If no Members are online, up-to-date state is retrieved from a Tool state repository such as a distributed file system.

Logouts from the Collaboration are described in Figure 4-9. If the Member is last to log out of a Collaboration, the logout process is somewhat complicated. An up-to-date version of the Collaboration Description must be saved in the Collaboration repository along with the work state of each individual Tools that is running. If there is a service breakdown before this operation is finished (such as if the user powers down without saving), Collaboration work is lost. To mitigate the effects of this, the CC should feature periodic auto-saves of the Collaboration Description, executed by peers selected in a distributed fashion.

If the Member logs out while there are still other Members online, she does not need to save anything to repository, but she does need to insert the following into the Collaboration Description:

- The present time (to *collaboration::member::last-logout*)
- What Tools she is presently using (*collaboration::member::tool-usage*)
- Windowing information on the Tools she is running and on the client module as a whole (*collaboration::tool::window-instance* and *collaboration::member::screen-properties::window-instance*)

If the Member suffers a service breakdown or if she fails to seize the writer's lock multiple times, outdated information will be reflected in the Collaboration Description. To mitigate the effects of this, peers should feature automatic periodic writing of this data to the Collaboration Description.

5. Implementation

This chapter describes the demo application that was implemented based on the design in chapter 4. The demo application and its corresponding Java source code package were called *ccprototype*. Figure 5-1 is a screenshot of ccprototype in action, showing the framework's desktop application interface and a number of Tools. This chapter will describe ccprototype, the *ccprototype.ccgroupnode* sub-package that manages membership traffic, the several demonstration Tools and the list of requirements the implementation places on its host system.

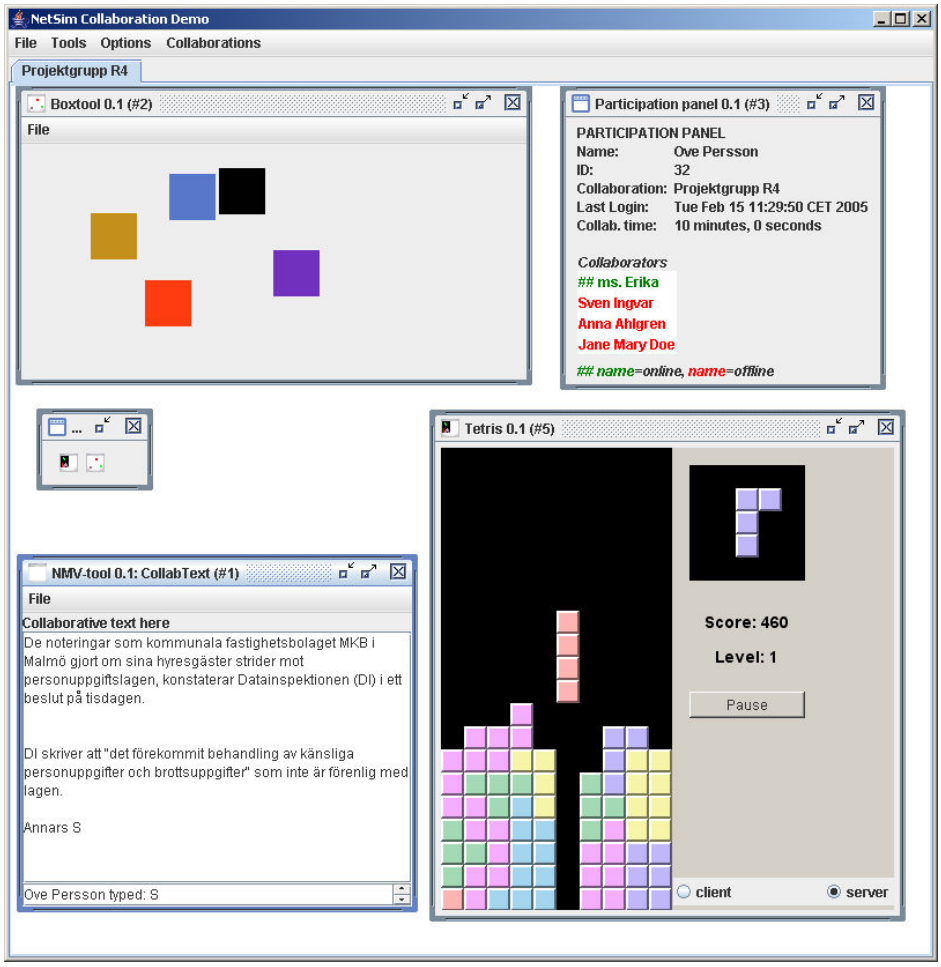


Figure 5-1 Screenshot of the demo implementation
Screenshot featuring a Collaboration with two Members (*Ove Persson* and *ms Erika*), using the Tools Boxtool, Participation Panel, Tetris, CollabText and Activity Panel.

5.1 Requirements

Ccprototype places a number of requirements on its hardware and software environment. This section lists these requirements, briefly describes them, and briefly comments how they were fulfilled or worked around in the ccprototype implementation.

Table 5-1 List of ccprototype requirements

Requirement	*Description*	*Implementation comment*
1. Client computers on an IP network	PCs using any platform that is supported by the JRE and by pRTI (see below)	Windows clients and an Ethernet LAN were used during development. Tests on Linux clients were successful.
2. JRE 1.4 or 1.5	Java Runtime Environment.	Minimal implementation difference. Code for both versions is provided.
3. pRTI 1516, v 2.3	RTI software supplied by Pitch AB.	An FOI license was used during development.
4. Common network location	Users in a Collaboration need read- and write access to a common network location.	Workaround: local directories on Collaborators' hard drives simulated the network location.
5. User ID system	Users of ccprototype must be associated with a unique numeric ID.	Not regarded, when logging in the user chooses any ID she wishes. Upon collision, the user is notified and login fails.
6. User authentication system	Authentication for access control and encryption policies.	Not implemented, which was prescribed by thesis parameters.
7. Tool ID system	Tools must be associated with a unique numeric ID (may coincide with a user ID).	Not regarded, Tool IDs were manually allocated to the five demonstration Tools.
8. Tool delivery	A mechanism for delivering Tools to first-time users.	Not regarded, all Tools were pre-delivered to all user machines.

Item 1 is the only hardware related requirement in the above list. It arises because the weight of the front end of ccprototype makes it unfit for devices less powerful than PCs. The design and the remainder of the implementation (including most of ccgroupnode) are however fully prepared for use on thin devices such as PDAs.

Items 2 and 3 represent straightforward software requirements.

Item 4 represents a requirement that could be satisfied by a distributed network. In the present implementation, it was simply worked around.

Items 5-8 represent requirements that the ccprototype places on its environment. In continuing efforts at FOI, several of these will be implemented, with the aim of integrating the ccprototype in an evolving system called NetSim.

5.2 ccprototype

The ccprototype has two roles.

I. ccprototype is parent to code for the demo implementation's front end.
II. ccprototype is the root package for the ccgroupnode package (explained below) and for demonstration Tool packages. It is also the root for code packages *cctool* and *net*, produced in a parallel Master's project by Liljeström. Liljeström also contributed to the Tool packages.

This section describes item I while following sections will describe the several sub-packages referenced in II.

The first component of the front end is a `JFrame`-based desktop application (`NetSimCollabDemo`, refer to Figure 5-2). It is charged with enabling user input concerning searches for Collaborations, as well as joins, exits, and switches between Collaborations, and RTI settings management. It implements management of multiple Collaborations (discussed in 4.1.2) through multiple `JTabbedPanes`.[2]

Figure 5-2 `NetSimCollabDemo`'s `JFrame` title bar and `JMenu`
Detail from Figure 5-1.

Each `JTabbedPane` contains an instance of the `JDesktopPane` descendant `CollaborationPane` (Figure 5-3). `CollaborationPane` is charged with managing the internal windows that house Tools used in the particular Collaboration. The `CollaborationPane` also maintains a link to the `MemberNode`, which is a central back end component discussed in sections 4.1 and 5.3.

Figure 5-3 Tab for the `CollaborationPane` of Collaboration "Projektgrupp R4"
Detail from Figure 5-1.

5.3 ccprototype.ccgroupnode

This section describes the ccgroupnode package, charged with managing the Collaboration information model. First, the section will specify exactly what the ccgroupnode does and what it does not do. It will then describe three implementation highlights: the Collaboration Description, the communication system and the framework front end components. Section 5.4 contains details on how the demonstration Tools were implemented.

5.3.1 What it does

The ccgroupnode implements the design described in chapter 4, principally managing the Collaboration information model and offering membership-related services. From a user perspective, it specifically carries out the following:

- Implements services for searching, creating, joining and resigning Collaborations (see 5.3.5).
- Implements services for reading and writing information about Tools and Members in specific Collaborations, and other Collaboration properties (5.3.3).
- Offers specific features in the collaboration framework front end (instant messaging, desktop layout sharing, presence awareness, activity awareness) (5.3.5).
- Opens its communication infrastructure to developers who wish to implement custom content, traffic-filtered communication between specific users (5.3.4).

[2] In this chapter, components implemented as classes are named with `fixed-width` font.

5.3.2 What it does not do

The following are some examples of what the sub-package ccgroupnode does not do, with explanations:

- Offer an interface to Tools or manage communication among Tools or save the work Collaborators perform with Tools. For an investigation into these questions, refer to Liljeström (2005).
- Offer a complete GUI for Collaborations. Some of this functionality is in ccgroupnode, but components with strong demo-flavor or components that are likely to be replaced or modified in production have been placed outside the sub-package. Most of these were discussed in 5.2.
- Offer applications that might be labeled "collaborative Word" or "collaborative Emacs". As has been mentioned before, this type of functionality is labeled a *Tool* in the present framework. Implementation of Tools in ccprototype are discussed in 5.4.

5.3.3 Collaboration Description implementation

The Collaboration Description is a central entity in the present design. As explained in 4.2.3, it is the vehicle of all relevant Collaboration information except for material produced within Tools. The Collaboration Description takes two implementation forms, each relevant in different parts of its life cycle. When transmitted via the network or when put in persistent storage, the Collaboration Description is implemented as XML. For an example, refer to Table 5-2. When loaded into client computer memory, the Collaboration Description is implemented as a DOM document wrapped in a ccgroupnode class called `CollaborationDescription`. The `CollaborationDescription` offers many convenience methods that access Description data, and offers parsing services to and from XML.

Table 5-2 Example of an XML Collaboration Description

```
<?xml version="1.0" encoding="UTF-8"?>
<collaboration description="A first prototype cd" id="5" name="Development project group">

    <member last-login="2005-02-08T09:45:28.15Z" last-logout="2005-01-26T16:07:15.91Z" name="Kalle Ank" user-id="32"
                                        desktop-width="1280" desktop-height="1024" client-run-state="window">
        <tool-usage tool-id="1" role="admin"/>
        <tool-usage tool-id="2" role="admin"/>
        <tool-usage tool-id="3" role="admin">
            <window-instance width="279" height="320" left="839" top="136" tool-run-state="windowed" is-selected="0"/>
        </tool-usage>
        <tool-usage tool-id="4" role="admin">
            <window-instance width="101" height="71" left="450" top="30" tool-run-state="windowed" is-selected="0"/>
        </tool-usage>
        <screen-properties desktop-width="1280" desktop-height="1024">
            <window-instance width="869" height="836" left="372" top="119" tool-run-state="windowed" is-selected="1"/>
        </screen-properties>
    </member>

    <member last-login="2005-02-01T12:38:29.70Z" last-logout="2005-01-26T15:44:36.56Z" name="Abraham" user-id="33"
                                        desktop-width="1280" desktop-height="1024" client-run-state="window">
        <tool-usage tool-id="1" role="admin"/>
        <tool-usage tool-id="3" role="admin">
            <window-instance width="293" height="320" left="113" top="49" tool-run-state="windowed" is-selected="1"/>
        </tool-usage>
        <tool-usage tool-id="4" role="admin">
            <window-instance width="101" height="71" left="450" top="30" tool-run-state="windowed" is-selected="0"/>
        </tool-usage>
        <screen-properties desktop-width="1280" desktop-height="1024">
            <window-instance width="785" height="679" left="278" top="215" tool-run-state="windowed" is-selected="1"/>
        </screen-properties>
    </member>

    <tool id="1" name="nmvtool" iconpath="C:\ccfolders\network\icons\NMVToolIcon.gif"/>
    <tool id="2" name="boxtool" iconpath="C:\ccfolders\network\icons\BoxToolIcon.gif"/>
    <tool id="3" name="participation-panel"/>
    <tool id="4" name="activity-panel"/>
    <tool id="5" name="tetTool" iconpath="C:\ccfolders\network\icons\TetrisIcon.gif"/>
    <tool id="6" name="tetTool" iconpath="C:\ccfolders\network\icons\TetrisIcon.gif"/>
</collaboration>
```

Collaboration Description handling is implemented in a somewhat naïve fashion. Going back to the life cycle from 4.2.4, the Collaboration Description can exist in three states:

1. Collaboration Description being composed by creator's client.
2. Collaboration Description maintained by peers in a distributed fashion.
3. Collaboration Description in persistent storage.

In state 1, the Collaboration Description data is being composed by the user in a dialog like the one in Figure 5-4. When she presses *Generate*, a `CollaborationDescription` instance is created and ccgroupnode attempts to launch a session of the Collaboration. If no Collaboration with identical ID is already in session on the RTI (no prior collision checks are made), the launch is successful and an XML copy of the Description is saved on the local disk. The `CollaborationDescription` instance is kept in memory and state 2 is entered.

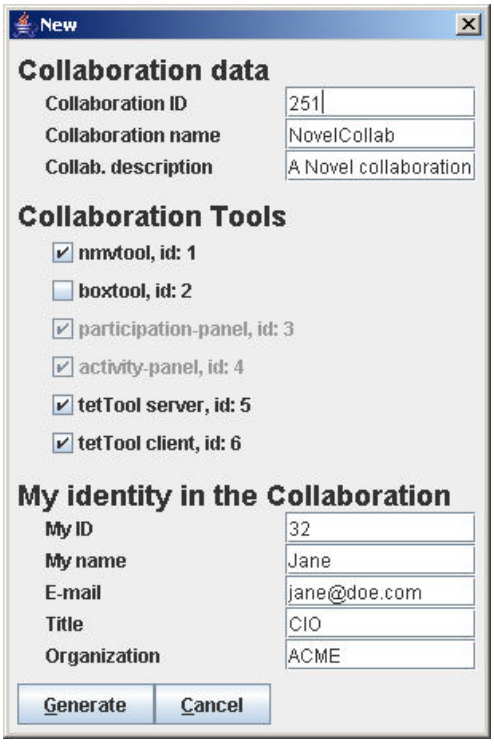


Figure 5-4 Dialog for creating a new Collaboration with a set of properties

The Collaboration Description is in state 2 whenever at least one user is in a Collaboration session. The user(s) then have a `CollaborationDescription` in memory and an XML Collaboration Description on disk to guard against crashes. No XML file exists on the network in state 2 (or state 1). Any user action that prompts a change in the Collaboration Description (name change, Tool addition etc) causes the user's ccgroupnode to transmit the updated Collaboration Description as XML to all other users. Other users then adopt this data as their Description, replacing the former Description in memory and on disk. When a user logs out of the session, her exit state (logout time and active Tool window data) is entered into her `CollaborationDescription` and sent as XML to other users, in the same manner as above. If the user was the

last online Member, she saves the Description to persistent storage (see state 3) and deletes the XML file on her local disk.

Consistency protection for the Collaboration Description was not implemented in the prototype. If this were to be added, a number of options are available. They would all be are relevant for state 2:

- A writer's lock, to ensure that users do not concurrently write to the Collaboration Description
- A detect-and-recover mechanism, based on previously saved Descriptions
- A transaction-based system for changes to the Description, possibly implemented in combination with one of the two previous mechanisms.

State 3 refers to having the Collaboration Description in persistent storage at a location accessible to all Collaboration members. In the project's implementation, persistent storage was implemented as storage in an XML file in a network directory. To simplify further, an implementation where this folder is only simulated to be on a network was used. A future full scale implementation should consider using a location on a distributed network as persistent storage.

5.3.4 Non-Collaboration Description communication

Communication between components in the present implementation is outlined in Figure 5-5 and Figure 5-6. Higher-level objects send data by using references and method calls to lower level objects. Low-level objects send data upwards by using events and listeners. MemberNode is an important hub in the communication system, serving as interface to both the RTI and the Collaboration Descriptor, and as traffic router for messages traveling upwards from the RTI.

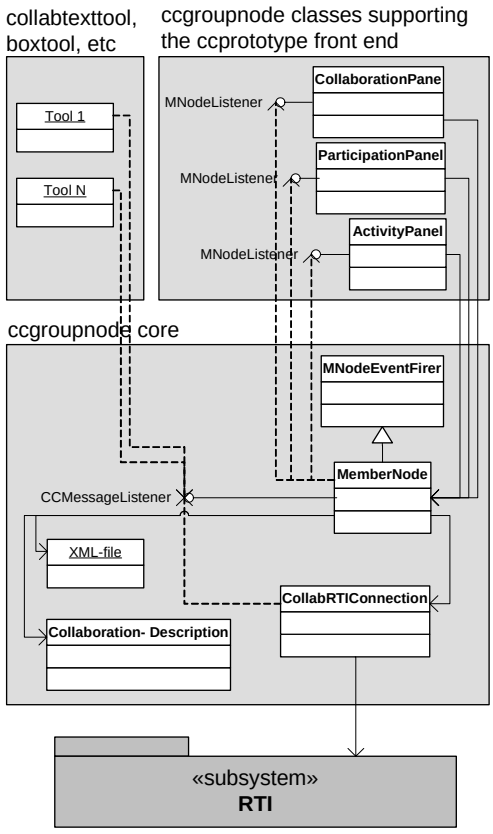


Figure 5-5 The communication infrastructure in ccgroupnode

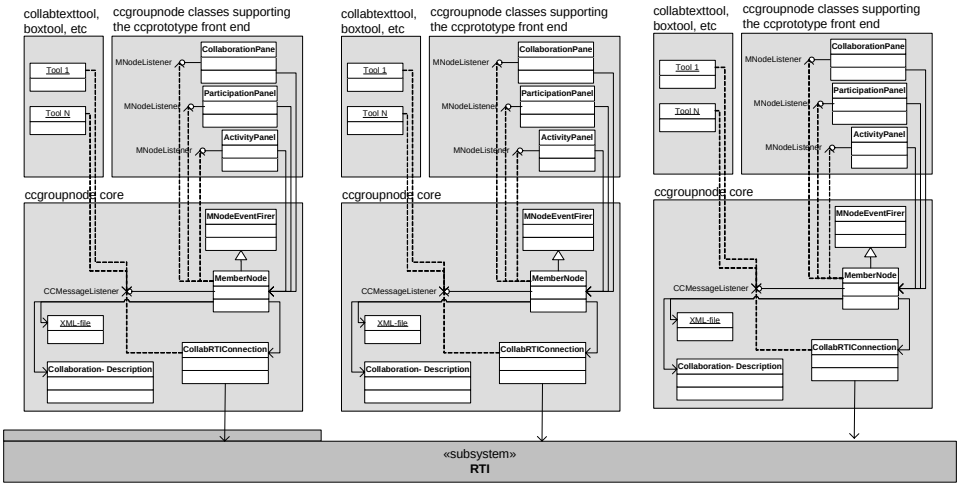
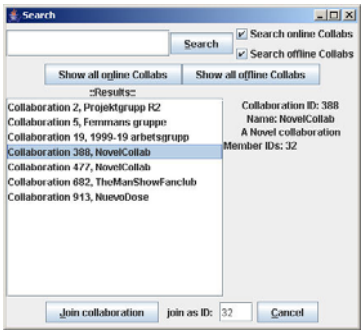


Figure 5-6 ccgroupnodes of three collaborators communicating via an RTI

5.3.5 Framework front end components

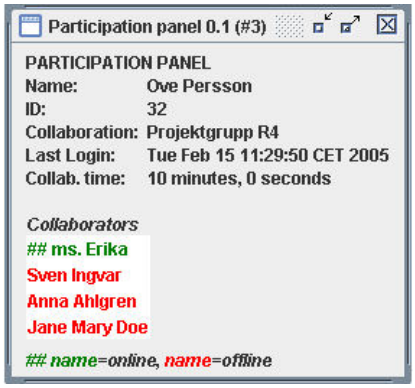
While most of ccgroupnode is concerned with background information management, it contains a number of components that are directly visible to the user. These are the *Collaboration Search panel*, the *Activity panel*, and the *Participation panel*.



The Collaboration Search panel (Figure 5-7) enables searching for and joining offline and online Collaborations. Offline Collaborations are reached via the Collaboration Descriptions in persistent storage. Online Collaborations are reached via their Collaboration Descriptions posted in the SysFederation (refer to 4.1.4).

Figure 5-7 Collaboration Search panel

The Participation panel (Figure 5-8) shows the user properties of the panel's Collaboration and a list of online and offline Collaboration members. Via this list, the panel allows the user interact with collaborators via instant messages. It also offers the option of requesting the desktop layout of a remote user. If the remote user accepts, the requester's application window layout becomes identical to the remote user. This means that the size and position of the application window becomes identical (scale-adjusted if users have non-identical screen resolution), as well as the size, position, and selection of internal windows for Tools (including the Participation panel itself).



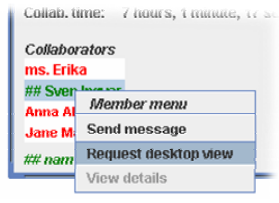


Figure 5-8 Participation panel (above) and a right-click remote desktop request (left)

The Activity panel is a `JPanel` component (within a `JInternalFrame`) that gives a graphical cue whenever there is activity by any collaborator in any Tool. Upon such activity, an icon for the Tool in question blinks for three seconds.

Figure 5-9 Activity panel with *Tetris* and *Boxtool* icons flashing

5.4 Demonstration Tools

In order to give a hint of future infrastructure applications, three demonstration Tools were developed. Work on these Tools was shared between the present author and Liljeström.

CollabTextTool (Figure 5-10) was the first Tool implemented. It is a simple collaborative text editor, where every letter a user types ends up on all collaborators' screens. It does not implement undo or erase funcionality. It uses `MemberNode` services to get the full name of the most recent writer to add it to the typing history (the bottom field).

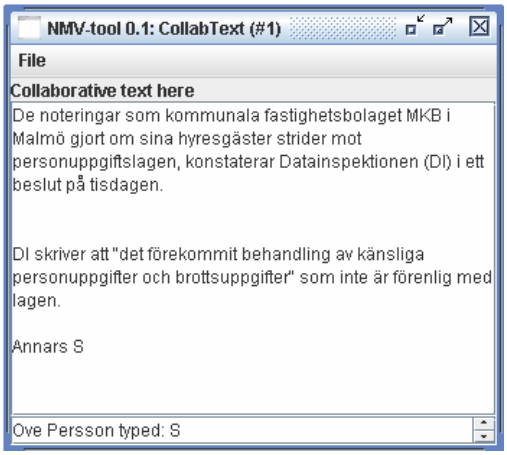


Figure 5-10 CollabTextTool

BoxTool (Figure 5-11) is a graphical Tool that lets users collaboratively add and move boxes. While a box is being moved by one user, another user cannot grab it. BoxTool is intended to hint at more advanced painting applications or modeling applications.

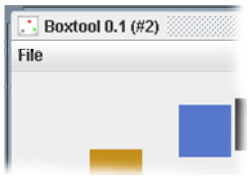


Figure 5-11 Detail from Figure 5-1: BoxTool

The Tetris tool (Figure 5-12) is a collaborative version of the game Tetris. The initiating user starts a server-instance of this game and starts playing it the way she would play a normal single-player Tetris. Collaborators can then join as clients, and get to see what happens in the game and get to send move commands to pieces. The Tetris tool is an open source Tetris implementation by Cederberg (2003), adapted for the ccprototype infrastructure by Liljeström.

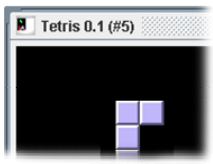


Figure 5-12 Detail from Figure 5-1: Tetris Tool

6. Experiments

This chapter describes HLA performance tests that were carried out during the Master's project. Results show that HLA communication is somewhat slower than raw socket communication and notably less regular. Disturbance patterns in HLA communication that invite further investigation are also shown. Experiments were done in collaboration with Liljeström's Master's project.

6.1 Experiment setup and method

Tests were carried out on pairs of computers, where each computer was equipped with a software test module that could send and receive batches of test messages and accurately time transmission cycles. Clocks were not precision-synchronized, but this presented no problem as experiment time was recorded by only one clock at a time: either the sender's clock (to time *send message-receive reply* cycles) or the receiver's clock (to time *receive reply-receive reply* cycles). Refer to Figure 6-1 for an overview of the experiment setup and to Table 6-1 for a list of variables that were investigated during the course of the experiment.

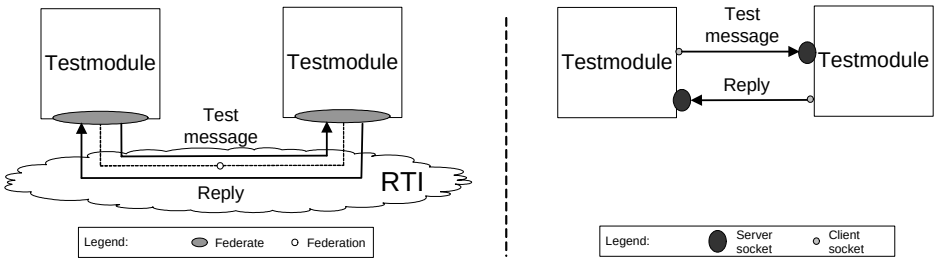


Figure 6-1 HLA communication vs. socket-based communication
The figure illustrates the dual machine 'bounce-back' experiment setup. In the other setup that was used, 'single-direction,' no reply message was sent.

Table 6-1 Experiment variables and testing intervals

Transmission times for messages in a batch *(dependent variable)*	-
Transmission method to use	Socket or HLA
Transmission cycle to time	Bounce-back or single-direction.
Size of each message in a batch	10 bytes - 16 000 000 bytes
Number of messages per batch	100 or 1000 messages.
Cache control	Either the same message was used in an entire batch, or several messages were used alternately, to counteract caching effects.
Network topology	Two or three computers on a reserved 100 MBit LAN or two or three computers on a shared 32 computer 100 MBit LAN with interfering traffic.
Test-machine performance	Pairs or triples were made out of the following: Three 1 GHz Pentium III, PCs and one 1.5 GHz Pentium IV PC. All had Windows XP, 256 MB RAM and 100 Mbit network adapters.
Pause between send intervals.	0 or 50 milliseconds.
HLA-specific variables	Transfer type: *HLA-Reliable* or *HLA Best Effort* RTI server setup: On receiving PC or on third PC

Transmission time was the central variable under study. A typical test run lasted a few minutes and after that time, the duration of every individual transmission was available in a text file on the initiating machine. The text file also contained aggregate data on the test. Individual tests were called *point-tests* because they tested one point in the nine-dimension space of dependent variables. Multiple point-tests could be performed in an automatic sequence. However, it was not feasible to perform tests with parameters modified according to every possible combination, even with the finite test ranges outlined in Table 6-1. Therefore, the experiments aimed to identify variables that had no significant effects on the dependent variables, then test these variables extensively in limited configurations. If repeated tests failed to reject the null hypothesis for a variable, the variable was then eliminated from further study, thereby reducing the dependent variable space by one dimension.

Table 6-2 shows an example of a point-test result file. After collection, text file data were inserted into spreadsheets for analysis. Certain data were exported to text-files anew for analysis in a second custom-made program equipped with capabilities not present in the spreadsheet program. Final results were always compiled in spreadsheets however, and they are presented in the following section.

Table 6-2 Example of a "point-test" result file
Experiment data for a 1000-message, single-direction HLA performance test run are shown. The column "interval" shows the millisecond length of the seven first intervals between message transmissions. The column "Recv interval" shows the length of the corresponding first seven intervals between transmission receptions timed at the receiving machine.

Start	Messages	Bytes/msg	Interval (msec)	Medium
Mon Jan 10 18:18:46	1000	650000	0	HLA
size	Sandtid, snitt	Mottagtid, snitt	Mottagtid, tot	Sandtid, tot
650000	137.6633918583393	137.77852056358194	137640.74204301834	137663.3918583393
size	Sandtid, stdav	Mottagtid, stdav		
650000	15.340667080921751	39.4946067546835		
NON-BOUNCE-BACK RESULTS				
Interval	Recv interval	Cumulative send time	Cumulative reception time	
129.3376669883728	81.51147699356079	129.3376669883728	81.51147699356079	
177.2269937992096	209.63362646102905	306.5646607875824	291.14510345458984	
127.31813669204712	130.01903867721558	433.8827974796295	421.1641421318054	
129.89695620536804	135.0498456954956	563.7797536849976	556.213987827301	
135.08420753479004	127.529616355896	698.8639612197876	683.743604183197	
127.44049859046936	87.88463354110718	826.304459810257	771.6282377243042	
165.4713099002838	204.4988956451416	991.7757697105408	976.1271333694458	
126.83818745613098	130.59341382980347	1118.6139571666718	1106.7205471992493	
130.62749600410461	127.22622585296631	1249.2414531707764	1233.9467730522156	
127.16644144058228	143.71184062957764	1376.4078946113586	1377.6586136817932	
143.68250727653503	88.30759191513062	1520.0904018878937	1465.9662055969238	
166.00098609924316	205.9367880821228	1686.0913879871368	1671.9029936790466	
128.04895615577698	127.92324161529541	1814.1403441429138	1799.826235294342	
•••				

6.2 Experiment results

Out of the nine independent variables that were studied, seven could be eliminated after extensive testing. Modifications of these seven variables were not associated with significant effects on the set of experiment transmission times. Eliminated variables are not to be seen as uninteresting, but neither are they likely to be responsible for the phenomena in the dependent variable accounted for below.

For each of the seven eliminated variables, a fixed experiment level was chosen for the tests in the remaining two-dimension independent variable space. The experiment levels were chosen in ranges that had been previously tested (and found not to effect the dependent variable) and they are listed in Table 6-3.

Table 6-3 Fixed levels for eliminated independent variables

Transmission times for messages in a batch *(dependent variable)*	-
Transmission method to use	Socket or HLA.
Transmission cycle to time	*Eliminated.* Set to single direction.
Size of each message in a batch	10 bytes - 16 000 000 bytes
Number of messages per batch	*Eliminated.* Set to 1000.
Cache control	*Eliminated.* No cache control was used.
Network topology	*Eliminated.* Two PCs on a 100 MBit reserved LAN were used.
Test-machine performance	*Eliminated.* 1 GHz Pentium III computers were used.
Pause between send intervals.	*Not regarded.* Set to 0 milliseconds.
HLA-specific variables	*Eliminated.* Transfer set to *Reliable,* RTI server run on receiving PC

Some comments are in order. Transmission cycle time turned out to be not very different whether the bounce-back or the single-direction setup was used (refer to Figure 6-1). Neither was it different when, in the single-direction setup, the remote reception values or the local sending values were used. The value used for "time" below is "intervals between sequentially sent messages, in milliseconds".

The number of messages had no effect on the dependent variable—provided the number was not small. A slight warm-up effect was recorded, with the first message in each batch typically being slower than the overall average. This could alternatively have been controlled by sending uncounted warm-up messages.

The "pause" variable could not be eliminated. When attempts were made to pause between transmission cycles, the JVM clock that determined the pause length turned out to be far less accurate than the experiment's HRTimer facility (featuring tenth-of-a-millisecond measurement precision; Roubtsov, 2002). A proper way to study the effects of sending-pauses would be to time the pauses as well as the sending periods with an accurate clock and to coalesce these data into an accurate measure.

The presentation uses SI-prefixes; 1 kB = 1000 bytes, 1 MB = 1 000 000 bytes.

6.2.1 Initial speed measurements, 400 B - 4 MB

The objective of the experiments was to arrive at a measurement for the speed of the HLA relative to a direct socket method. Results should hence not be seen as absolute or exact measures of HLA speed. Experiments regarded both sockets and HLA as raw communication vessels and did not consider factors other than speed. Figure 6-2 shows an initial wide-net investigation covering five orders of message-size magnitude, ranging from 400 bytes to 4 000 000 bytes.

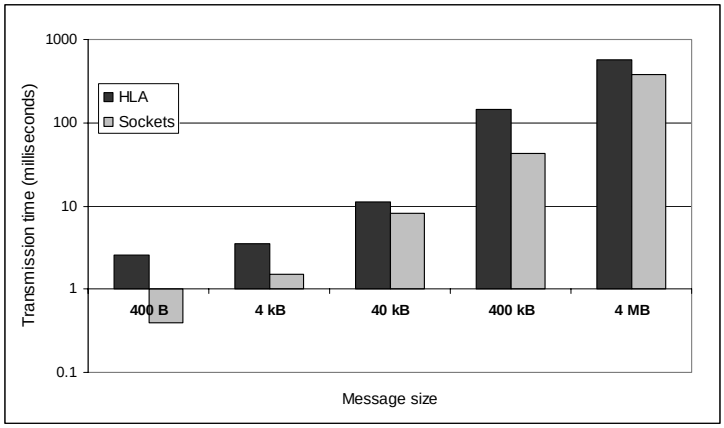


Figure 6-2 HLA vs Sockets, average sending time (logarithmic), 400 B - 4 MB
10 000 test transmissions in total.

The average transmission speed in the test in Figure 6-2 was 2.9 MB/s for HLA and 5.7 MB/s for sockets. Large messages were sent with significantly higher per-byte speed. The greatest difference between the two methods occurred with 400 byte and 400 kilobyte messages, where sockets were 5.5 and 2.4 times faster than HLA, respectively.

HLA transmissions were fairly variable around the HLA regression line inserted in Figure 6-3. The 5000 HLA transmissions were on average 36.0 milliseconds off the HLA line while the 5000 socket transmissions were only off the socket line by an average 1.0 millisecond. The greatest variance within an individual batch occurred with the 1000 messages in the HLA's 400 kB batch (refer to Figure 6-4).

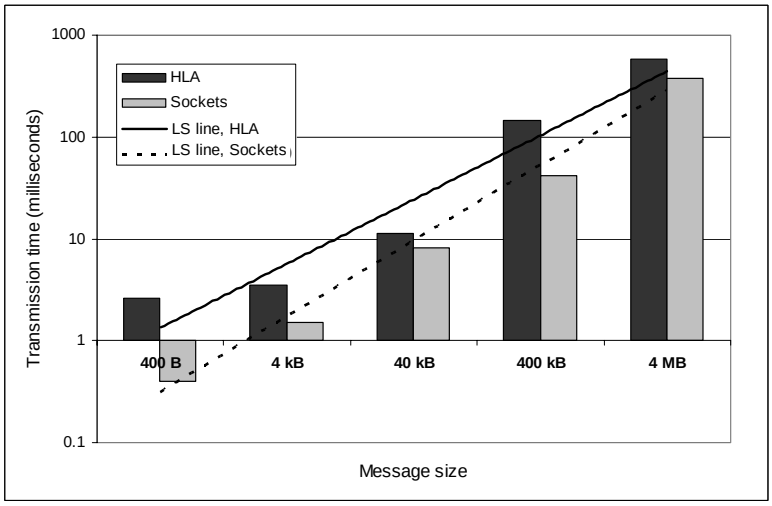


Figure 6-3 HLA vs Sockets, 400 B - 4 MB, with least-squares regression lines

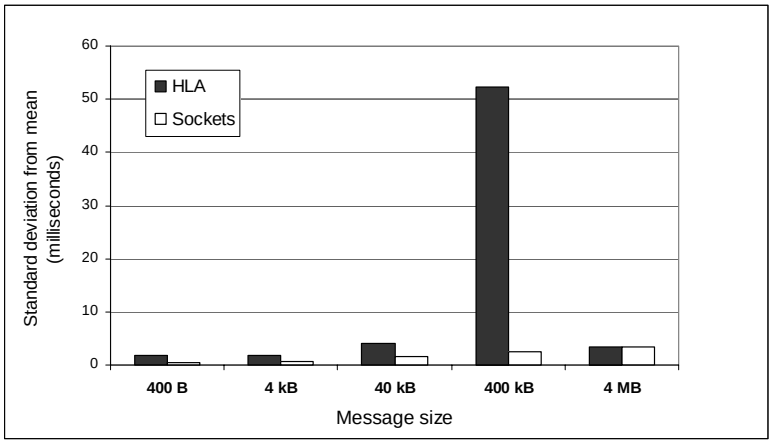


Figure 6-4 HLA vs sockets, standard deviation from mean

6.2.2 Finer grain speed measurements

The apparent HLA anomaly at the 400 kB size was investigated further in higher resolution tests around this level. As before, HLA tests were paired with socket transmission tests, and message sizes were varied. Figure 6-5 shows an investigation of socket transmission times for 20 batches of 1000 messages containing messages ranging from 50 kB to 1 MB.

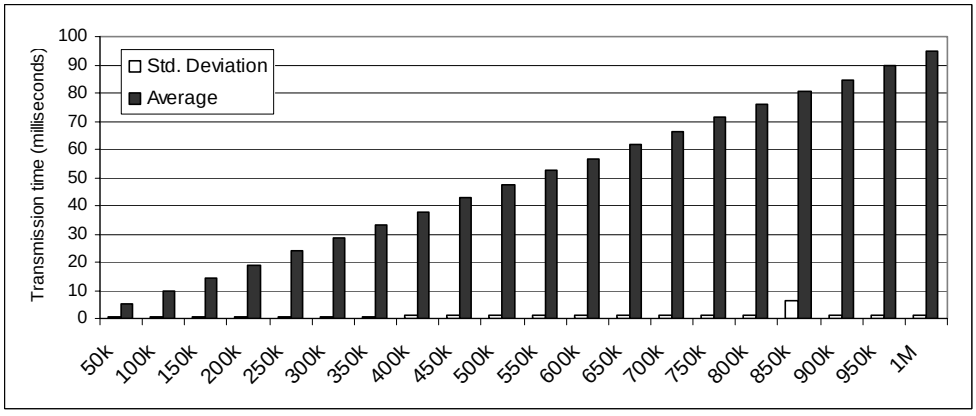


Figure 6-5 Average *socket* send time for batch-wise sent messages of varying sizes.

Figure 6-6 shows a same-sized sequence of point-tests as Figure 6-5 with HLA as transmission method. The regular linear pattern in Figure 6-5 stands in contrast with the slightly more irregular linear relationship in Figure 6-6. The average standard deviation within the twenty socket batches was 1 millisecond while the corresponding figure for the HLA batches in Figure 6-6 below was 28 milliseconds.

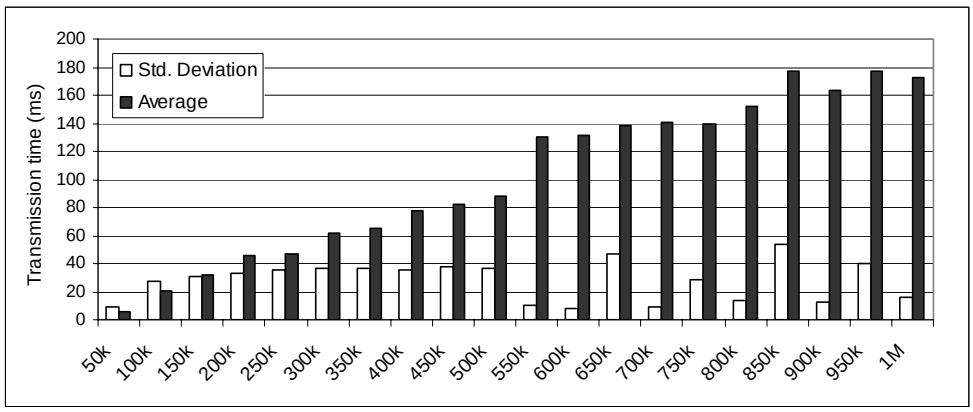


Figure 6-6 Average *HLA* send time for batch-wise sent messages of varying size

A similar HLA irregularity can be seen in the finer-grain, smaller size tests in Figure 6-7.

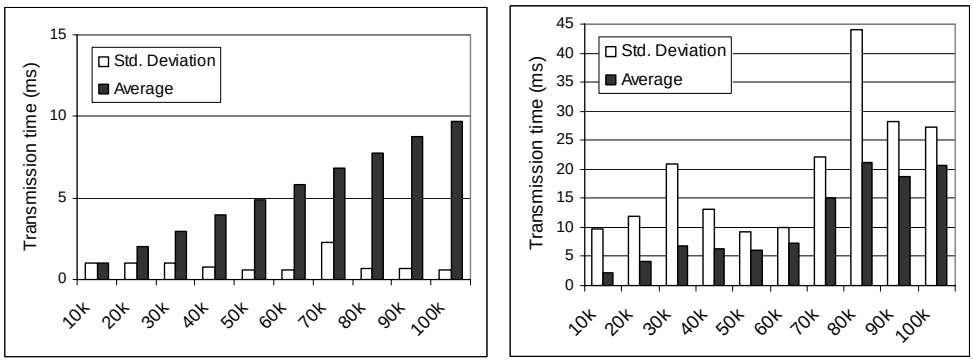


Figure 6-7 Transmissions of messages sized 10-100 kB, HLA and socket
Ten socket test runs are shown left and ten HLA test runs right.

6.2.3 Further investigation into HLA irregularity

To search for patterns in the irregularity of HLA transmissions, several histogram analyses were performed.

Figure 6-8 shows a histogram for a single point-test. Based on timing data for a batch of a thousand 50kB socket transmissions, it reveals that the typical transmission took between 5 and 6 milliseconds, with 599 transmissions falling within this range.

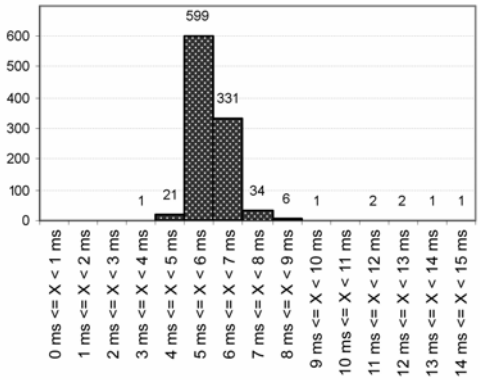


Figure 6-8 Histogram of durations of a thousand 50kB socket transmissions
Durations are approximately normally distributed.

Figure 6-9 shows a histogram breakdown of an entire sequence of twenty socket tests (the sequence in Figure 6-5 actually), with the bottom left one being the 50kB test from Figure 6-8. The figure reveals an apparent linear dependency between size and transmission time. Furthermore, mean transmission times for individual batches exhibit approximate normal distribution.

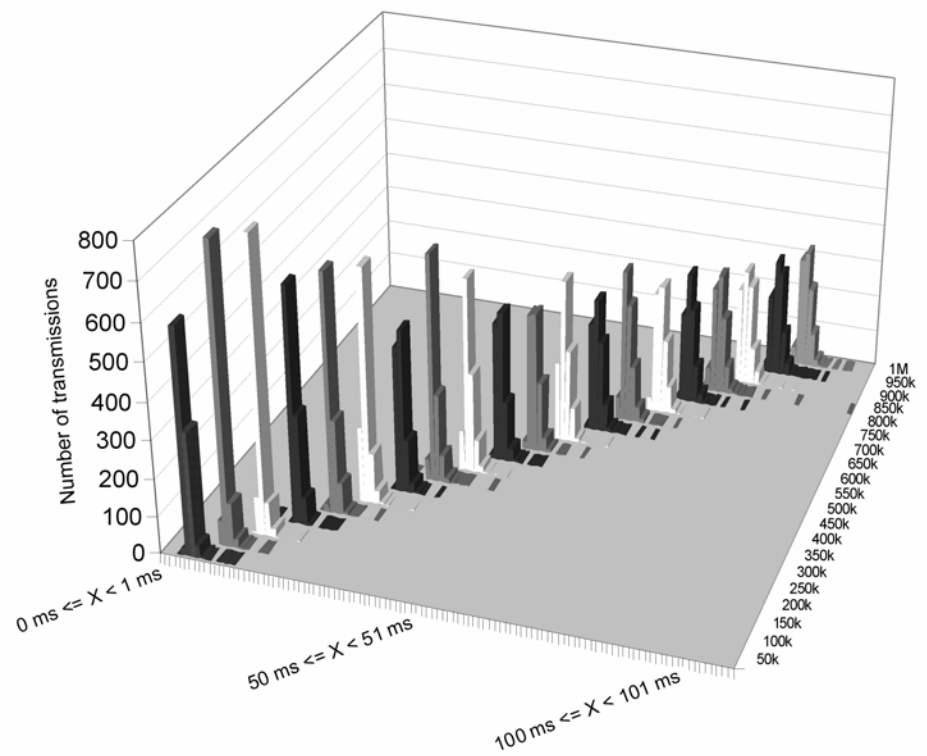


Figure 6-9 Histograms for 20 socket batches, each of 1000 transmissions
Transmission times appear to be normally distributed and linearly size-dependent.

The HLA-diagram in Figure 6-10 is constructed in the same way as Figure 6-9, but it reveals a fundamentally different pattern. Most HLA transmission times exhibit bi-modal distribution, with values clustering around two means instead of one. Some batches containing larger messages (650 kB and above) even exhibit tri-modal distributions.

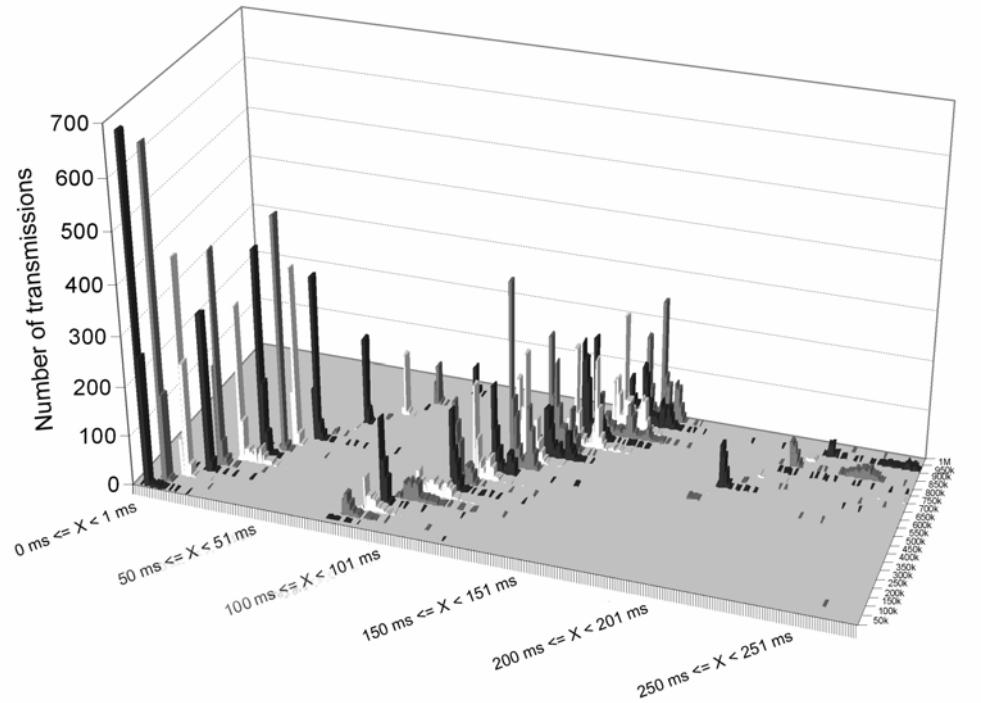


Figure 6-10 Histogram over 20 HLA batches, each containing 1000 messages
The contrast to Figure 6-9 is striking. Distributions are now bi- or tri-modal—e.g. a typical 200k HLA transmission took *either* 22 *or* 92 milliseconds.

For HLA batches with messages ranging from 50kB to 550kB, the spacing betweens apparent peaks in the bi-modal distribution was studied. Table 6-4 shows the results of this least-variance investigation, and it indicates a stable between-peak period of 75 milliseconds. Batches of messages sized over 550 kB exhibited less clear-cut distributions. Following the hypothesis that larger distributions are tri-modal (supported by 650kB, 850kB and 950kB series), the third peaks can be seen after a second approximate 75 millisecond period.

Table 6-4 Dual subgroup means in each batch (milliseconds)
Each pair of subgroups was formed so that the sum of their respective variances was smaller than the corresponding sum for any other configuration of subgroups.

	50kB	100kB	150kB	200kB	250kB	300kB	350kB	400kB	450kB	500kB	550kB
Mean 1	5.0 (σ1.2)	10 (σ0.8)	16 (σ0.8)	22 (σ1.0)	24 (σ1.0)	33 (σ3.5)	37 (σ0.8)	42 (σ0.9)	45 (σ0.4)	52 (σ1.4)	56 (σ0.1)
Mean 2	85 (σ12)	85 (σ15)	90 (σ3.6)	92 (σ2.3)	103 (σ4.9)	108 (σ4.7)	114 (σ2.7)	114 (σ2.8)	120 (σ3.4)	125 (σ4.0)	131 (σ8.8)

Average mean spacing: 75.0 ms Standard deviation (σ) of mean spacing: 2.7 ms

A similar period was observed in other batches as well. Figure 6-11 shows a series of 10 kB to 100 kB message batches. Although the second peaks are less common, the peak period is similar. Ignoring series with weak bi-modality (30k and 60k) or with many outliers (80k), this sequence's average mean spacing is 79 milliseconds with the Table 6-4 method.

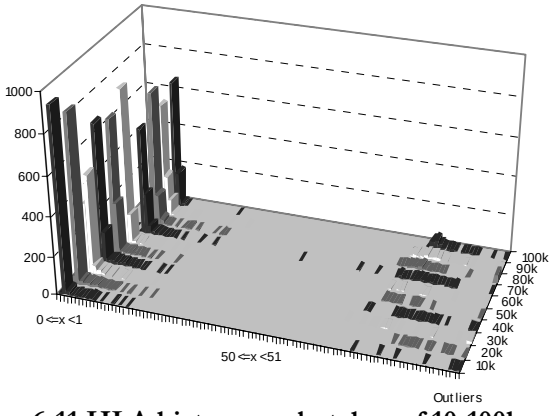


Figure 6-11 HLA histogram, batches of 10-100k

6.3 Experiment discussion

A curious periodicity was observed in transmissions via the HLA RTI. The pattern emerges with messages in the tens or hundreds of kilobytes. Whereas direct socket transmissions performed predictably, with an apparent linear relationship between message size and transmission time, HLA transmissions performed erratically and, on average, slower.

Closer investigations eliminated cache mechanisms and RTI host overhead as explanatory variables. Several other variables were eliminated from the independent variable space for having no or uninteresting effects on speed. These include the number of messages per test session, transmission cycle timing variants, network topology, and test-machine performance.

The erratic aggregate data is founded on a periodicity effect best shown in Figure 6-10 and Figure 6-11. Messages in the 100kB range tend to get delivered in 10 milliseconds (10 MB/s), but a fraction of messages get delivered slower (typically in 85 milliseconds, 1,2 MB/s). With larger messages, the fraction of low-speed transmissions is larger, with a notable turning point at the 500kB level. Several tests show that when messages grow to sizes of 500kB to 550kB,

the slower transmission time becomes dominant. As messages grow larger still, a third typical transmission time appears. The spacing between typical transmission times was observed to be constant at approximately 75 milliseconds.

Discussions with the RTI supplier, Pitch AB, suggest that memory management of the Java Virtual Machine (JVM) may be sub-optimal within the context of pRTI's large object handling. Repetitions of the study should consider using different JVMs used, as well as using several different RTI implementations.

While some simulation applications do not operate with event messages in the ranges of hundreds of kilobytes, many do. Furthermore, if the RTI simulation infrastructure is to be extended to areas such as Computer-Supported Collaborative Work, there will definitely be large message passing and any performance erraticism with this needs to be investigated.

7. Conclusion & future work

This chapter presents the Master's project's findings along with a brief discussion of future work.

7.1 Project findings

The project's conclusion is made up of a number of smaller findings in several categories. This is due to the fact that the project's central research issue was feasibility.

The project's central finding is that it is indeed feasible to construct a software platform for collaborative applications using an HLA communication substructure and an XML information model for group management. This finding can be structured as follows:

1. A collaborative platform based on the HLA and XML can feasibly be developed.
   - **A.** A design centered on the Collaboration concept is theoretically appealing.
     - **i.** A Collaboration can be effectively described in an XML data structure.
     - **ii.** Collaboration communication can occur in an HLA Federation.
       - **a.** Interactions are appropriate transmission vehicles.
       - **b.** HLA Objects are appropriate for posting properties.
       - **c.** RTI Data Distribution Management can be used for traffic filtering
     - **iii.** Peer-to-peer architecture can be used (nearly) throughout the design.
       - **a.** Collaboration sessions can be driven entirely by P2P communication.
       - **b.** Global Collaboration search can be accomplished in P2P by using a global Federation with a reserved name (such as "__SysFederation").
       - **c.** Persistent storage of Collaboration properties and work when collaborators are offline must be handled with non-P2P methods.
   - **B.** An implementation based on pRTI, Java, and XML files has demonstrated appealing features in practice.
     - **i.** Information model maintenance using replicated-state XML files and "write-all" updates is possible. More sophisticated methods are available.
     - **ii.** DOM conversions between XML and a Java representation class (`CollaborationDescription`) were effective when managing and transmitting the collaboration information model.
     - **iii.** Having a number of Collaboration-supporting group services built into the infrastructure (e.g. to provide users awareness of other users' presence and activity) is beneficial.
2. pRTI performance was irregular
   - **A.** The infrastructure is not suited for intense collaboration around large objects.
   - **B.** The infrastructure is not well suited for heavy transfers such as video feeds. It can coordinate such transfers, however, and make sure they occur properly in a dedicated channel outside of the infrastructure.

During the course of the project, one major auxiliary issue arose: item 2 in the structure above. This was an issue regarding the pRTI's performance when handling large pieces of data. The conclusion was that RTI communication capabilities show potential when compared to a direct socket communication method, but that performance is poorer and less regular. A joint investigation into this issue with the RTI supplier yielded the preliminary conclusion that

irregularities could be due to sub-optimal memory management of large objects in the RTI implementation's Java code.

7.2 Future work

The findings tree above is associated with considerable future work, most notably in implementation. Other items of future work can also be foreseen.

7.2.1 Continuing work on the ccprototype implementation

The mechanism for maintaining the Collaboration Description in a distributed fashion while doing work on it (item *1.B.i.*) can be made more robust by implementing a writer's lock to prevent write-write-conflicts. The mechanism can also be made more efficient by implementing a different distribution method such as cascaded sending or by adopting a transaction-like mode of operation instead of updating entire descriptions.

The persistent storage for Collaboration Descriptions could also be refined. Presently seen as a network location, this storage could alternately be a server which indexes Descriptions and offers fast-search features and safe-modification features.

Certain items of work on built-in framework components have also been left for the future:

- Implementing checks and recovery for corrupted Collaboration Descriptions
- Implementing Tool-initiation ability in the Activity panel
- Implementing a history list of active collaborators in the Activity panel
- Implementing a file-transfer feature in the Participation panel

Lastly, two HLA-related work recommendations are the following:

- Implementing saves of DDM region handles in memory once they have been created (in `CollabRTIConnection`), to avoid repeated creation.
- Expanding the DDM ID range (the range of dimension *active-listeners* in the MemberNode FDD file) to be as large as the future range of externally provided numeric user IDs.

7.2.2 Other future work

Regarding item 2 in the section 7.1 findings tree, the suppliers of pRTI suggest RTI performance be investigated using an alternate Java Virtual Machine. Future investigations might want to look into performance with JVMs such as BEA JRockit or IBM's Java Development Kit.

A different line of future work lies in the tasks of making the project's infrastructure (or a similar infrastructure) more adaptable to collaborative applications, and to actually adapt a full-scale application to it. The latter part entails taking an application, preferably an open source application such as *OpenOffice Draw*, and enabling collaborative work within it using the infrastructure's services.

References

This reference list adheres to APA Style as described by the *APA Publications Manual* (2001) with one modification: in two cases where an URL is available but APA Style does not require it to be shown, the URL is given in brackets at the end of the reference.

Ahmed, T., Kumar, R., and Tripathi, A. (2002). *Secure Management of Distributed Collaboration Systems.* (Tech. Rep.). Minneapolis, MN, USA: University of Minnesota, Dept. of Computer Science.

APA. (2001). *Publication Manual of the American Psychological Association* (5th ed.) Washington DC: APA Press.

Ayani, R., & Dharma, R. B. (2003, January). *Web-based Collaborative Simulation* In Tan Soon Huat, G. (Chair), *Cluster-Based Distributed Simulation Using HLA.* Seminar conducted at the National University of Singapore.

Banks, J., Carson, J. S. II, Nelson, B. L., & Nicol, D. M. (2001). *Descrete-Event System Simulation* (3rd edition). New Jersey: Prentice Hall.

Bannon, L., & Schmidt, K. (1991). Four characteristics in search of a context. In J. Bowers & S. Benford (Eds.), *Studies in Computer Supported Collaborative Work: Theory, Practice and Design,* (pp. 3-16). Amsterdam: North-Holland.

Baymani, S., & Strifeldt, E. (2005). *Carbonara—a semantically searchable distributed repository.* Master's thesis, KTH Royal Institute of Technology.

Bowen, S., & Maurer, F. (2002). Using Peer-to-Peer Technology to Support Global Software – some initial thoughts. In Damian, D., Maurer, F., & Sridhar, N. (Eds.), *Proceedings of the ICSE Int. Workshop on Global Software Development* (pp. 2-5), Orlando, FL: Sydney University of Technology.

Cederberg, P. (2003). Online Tetris Game. Retrieved from http://www. percederberg.net/home/java/tetris/tetris.html.

Churchill, E. F., Snowdon, D. N., & Munro, A. J. (Eds.). (2001). *Collaborative Virtual Environemnts – Digital Places and Spaces for Interaction.* London: Springer-Verlag.

Eseryel, D., Ganesan, R., and Edmonds, G. S. (2002). Review of Computer-Supported Collaborative Work Systems. *Educational Technology & Society,* 5(2), 130–136.

Fox, G.C., Furmanski, W., & Ozdemir, H. T. (1998). *Object Web (Java/CORBA) Based RTI to Support Metacomputing M&S.* Retrieved January 17, 2005, from http://www.dtc.army.mil/ hpcw/ 1998/furm4/furm.html

Fujimoto, R. M. (2003). Distributed Simulation Systems. In Chick, S., Sánchez, P. J., Ferrin, D. & Morris, D. J. (Eds.) *Proceedings of the 2003 Winter Simulation Conference* (pp. 124-134). New Jersey, NJ: IEEE Press.

Gartner Research (2004, February 20). *Worldwide Internet Access, 2003 Update (Executive Summary)*. Retrieved December 2, 2004, from http://0-biblioteca.itesm.mx.millenium.itesm.mx/gartner/research/119700/119790/119790.html

Roubtsov, V. (2003, January 10). *JavaWorld - My kingdom for a good timer*. Retrieved December 18, 2004, from http://www.javaworld.com/javaworld/javaqa/2003-01/01-qa-0110-timing.html

IDC (2002, September 27). *We've all got mail: IDC predicts 60 billion e-mails a day by 2006*. Retrieved December 2, 2004, from http://www.computerworld.com/softwaretopics/software/groupware/story/0,10801,74682,00.html

Kuhl, F., Weatherly, R., & Dahmann, J. (1999). *Creating Computer Simulation Systems – An introduction to the High Level Architecture*. Upper Saddle River, NJ: Prentice Hall.

Li, D., & Li, R. (2002, November). Bridging the gap between single-user and multi-user editors challenges, solutions, and open issues. In Sun, C. (Chair), *Session III of the Fourth International Workshop on Collaborative Editing Systems*, New Orleans, LA, USA.

Liljeström, M. (2005). *Generic XML-based Interface for Computer Supported Collaborative Work in an HLA Environment*. Master's thesis, KTH Royal Institute of Technology.

Marsic, I. (2000). Real-Time Collaboration in Heterogenous Computing Environments. In *Proceedings of the International Conference On Information Technology: Coding And Computing* (pp. 146-151). Las Vegas, NV, USA: Computer Society Press.

Moradi, F., & Ayani, R. (2003). Parallel and distributed simulation. In M. S. Obaidat & G. I. Papadimitriou (Eds.), *Applied System Simulation – Methodologies and Applications* (pp. 457-486). Dordrecht: Kluwer Academic Publishers.

Obaidat, M. S., & Papadimitriou, G. I. (2003). Introduction to applied system simulation. In M. S. Obaidat & G. I. Papadimitriou (Eds.), *Applied System Simulation – Methodologies and Applications* (pp. 1-8). Dordrecht: Kluwer Academic Publishers.

Prakash, A., Shim, H. S., & Lee, J. H. (1999). Data management issues and trade-offs in CSCW systems. *IEEE Transactions on Knowledge and Data Engineering,* 11(1), 213-227.

Reid, M.R. (2000, November). *An Evaluation of the High Level Architecture (HLA) as a Framework for NASA Modeling and Simulation*. Paper presented at the

25th NASA Software Engineering Workshop, Goddard Space Flight Center, Greenbelt, MD, USA. [http://sel.gsfc.nasa.gov/website/sew/2000/topics/MReid_SEW25_Paper.PDF]

Severinson-Eklundh, K. (1998). *Computer-supported cooperative work*. Retrieved January 10, 2005, from Royal Institute of Technology, NADA Web site: http://www.nada.kth.se/~kse/ cscw/F1.pdf

Stewart, J., Benderson, B. B., and Druin, A. (1999). Single Display Groupware: A Model for Co-present Collaboration. In *Proceedings of the CHI 99 Conference on Human Factors in Computing Systems* (pp. 286–293). New York: ACM Press.

Tanenbaum, A. S., & Steen, M. van, (2002). *Distributed Systems – Principles and Paradigms*. Amsterdam: Prentice Hall.

Tse, E., Histon, J., Scott, S. D., & Greenberg, S. (2004). Avoiding interference: how people use spatial separation and partitioning in SDG workspaces. In *Proceedings of the ACM Conference on Computer-Supported Collaborative Work* (pp. 252-261). Chicago, IL, USA: ACM Press.

Tollinger, I., McCurdy, M, Vera, A. H., & Tollinger, P. (2004). Collaborative knowledge management supporting Mars mission scientists. In *Proceedings of the ACM Conference on Computer-Supported Collaborative Work* (pp. 29-38). Chicago, IL, USA: ACM Press.

Vuong, S., Scratchley, C., Le C., Cai, X. J., Leong, I., Li L., Zeng, J., and Sigharian, S. *Towards a Scalable Collaborative Environment (SCE) for Internet Distributed Application: A P2P Chess Game System as an Example*. Unpublished manuscript, University of British Columbia [http://www.magnetargames.com/Technology/DAIS-Vuong Chess-230603R.doc (retrieved October 12, 2004)]

Yamauchi, Y., Yokozawa, M., Shinohara, T. & Ishida, T. (2000). Collaboration with lean media: How open-source software succeeds. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (pp. 329-338). Philadelphia, Pennsylvania, USA: ACM Press.

Zhao, H., & Georganas, N.D. (2001). Collaborative Virtual Environments: Managing the Shared Spaces. *Networking and Information Systems Journal*, 3(2), 1-23.

Appendix A: Tool communication modes

An important CC task is to enable late arrival to Collaborations. Presently, ensuring consistency in late-arriving Collaborators is a complex task for collaborative applications. The CC is intended to facilitate this task for application designers, and one model to do so would be to support the following 12 update modes.

Information necessary for updating newly-arrived collaborator

	History			
State	No history	Full history (all events)	Partial history (last N events)	Selected history (certain relevant events)
No state	X	X	X	X
Full state	X	X	X	X
Selected final state (certain relevant parts of state information)	X	X	X	X

In other words, tools plugged into the CC must declare what kind of information is necessary to update a new collaborator using that same tool. Collaboration sessions are defined by state-information and by the events that have occurred. These two variables take three and four modes, respectively, making 12 combinations. A few examples follow.

1. Two collaborators start a chat session and each writes five messages. A third collaborator joins. **No state or history information necessary for update.** (Person number three may start chatting immediately, without knowledge of prior messages. In fact, for integrity purposes, he is not *supposed* to have knowledge of prior messages as they may not have been intended for him.)

2. Two collaborators start a collaborative drawing session. They each draw five shapes and then collaborator A modifies two while collaborator B deletes one. Collaborator C joins. **Collaborator C needs to be updated with the final state of the drawing.** Alternately, a different drawing application may want to update him with **final state and full history**, or **final state and a partial history consisting of e.g. the last three events**.

3. Two collaborators and an observer start a multi-game, multi-player board game application. They select the game chess and the players each make five moves. An additional observer joins. **The observer needs to be updated with the full state of the collaboration (game selection, color selection etc) and the full history of events**.

4. Two collaborators start a video conference and talk for five minutes. The video channel is handled outside of the CC for performance reasons. Applications use the CC exchange information necessary to set up the external channel however – i.e. to exchange IP addresses and agree on encoding settings. A third collaborator joins. **The third collaborator needs to be updated with the state of the collaboration (i.e. the conference settings)**.

5. Three collaborators start using a collaborative article-authoring and typesetting tool. This tool has extensive text editing and graphics-handling capabilities and it allows users to maintain parts of their work private for performance reasons and integrity reasons. The three collaborators work for an hour, during which they each privately produce a two-page article. During the hour, collaborator A invites collaborator B to review one of his pages, while collaborator C offers his entire article for general review. Collaborator C also produces a title-logo which he makes publicly accessible to the collaboration. Collaborator D joins. **Collaborator D needs to be updated with a selected final state**. In other words – he needs to be updated with those selected parts of the final state which are supposed to be available to him: the article and graphics of collaborator C.

ccgroupnode CD-ROM

The CD contains the ccgroupnode package, auxiliary code libraries, and Javadoc documentation. Refer to the folder `readme` on the CD for further content information and for installation instructions.

List of acronyms

APA	American Psychological Association
CC	Collaborative Core
CSCW	Computer-supported collaborative work
CVE	Collaborative Virtual Environment
CVS	Concurrent Versions System
DDM	Data distribution management
DIS	Distributed interactive simulation
DMSO	Defense Modeling and Simulation Office
FDD	Federation description document
FOI	Swedish Defense Research Agency
FOM	Federation object model
HLA	High level architecture
KTH	Royal Institute of Technology, Stockholm
NADA	Department for numerical analysis and computer science
OMT	Object model template
pRTI	Pitch AB's RTI
RTI	Run-time infrastructure
SOM	Simulation object model