Synchronization Policies in Collaborative Work Environments

MARIANO MALDONADO

Examiner: Prof. Rassul Ayani
Supervisor: Tekn. Lic. Jenny Ulriksson

Master of Science Thesis
Stockholm, Sweden 2006

Abstract

Most of the Computer Supported Cooperative Work (CSCW[1]) systems of today are simple-purpose when it comes to shared objects, which means that there is only one underlying synchronization policy that is designed for a specific behavior pattern. If the users would interact with the object in an unexpected fashion the synchronization policy could cause unnecessary overhead because it would not be able to handle that behaviour optimally.

This master thesis project attempts to find a way of improving the CSCW systems of today by providing a simulator platform for testing synchronization policies in different scenarios and by presenting guidelines describing how the tested policies perform under different circumstances. The main purpose of the simulator is to analyze and evaluate synchronization policies in different scenarios. It could also, together with the guidelines, serve as a useful tool in the development of a mechanism that would be able to choose the best suited synchronization policy in real-time based on the behavior of the participants and the properties of the system.

The simulator implements three different synchronization policies that can be tested with a variety of different parameters; such as number of participants, reading velocity, interaction need and rollback penalty. The guidelines were formulated by observing how different user behavior and system properties affected the performance of the synchronization policies in a variety of scenarios.

[1] Systems that allow geographically distributed groups of people to share a working environment enabling them to collaborate towards a common goal.

Sammanfattning

De flesta Computer Supported Cooperative Work (CSCW[1]) system idag har endast en underliggande synkroniseringsmekanism för de delade objekten, som oftast bara klarar av att hantera en viss typ av beteende effektivt. Om någon av användarna skulle interagera med systemet på ett oväntat sätt skulle synkroniseringsmekanismen kunna orsaka prestandaförluster eftersom den inte skulle kunna hantera beteendet på ett optimalt sätt.

Detta examensarbete ämnar hitta ett sätt att förbättra dagens CSCW-system genom att tillhandahålla en simuleringsplattform för synkroniseringsmekanismer och genom att presentera riktlinjer som beskriver hur de testade mekanismerna presterar under olika beteenden och omständigheter. Det första ändamålet med simuleringsplattformen är att den ska användas för att analysera och utvärdera synkroniseringsmekanismer i olika scenarier. Den kan också tillsammans med riktlinjerna tjäna som ett användbart verktyg i utvecklingen av en mekanism som i realtid ska välja den bäst lämpade synkroniseringsmekanismen baserat på användarnas beteende och systemets egenskaper.

Simulatorn implementerar tre olika synkroniseringspolicys som kan testas med en mängd olika parametrar, bland dessa finns antalet användare, läshastighet, interaktionsbehov och rollback-straff. Riktlinjerna utformades genom att observera hur de olika beteendena och systemegenskaperna påverkade de olika synkroniseringsmekanismernas prestanda i olika scenarier.

[1] System som tillåter geografiskt spridda grupper att samarbeta mot ett gemensamt mål genom att tillhandahålla en gemensam arbetsmiljö.

Acknowledgements

This master thesis was carried out at The School of Information and Communication (ICT) at The Royal Institute of Technology (KTH) and at the Swedish Defence Research Agency (FOI) in Stockholm, Sweden. The work on the thesis commenced in September 2005 and ended in April 2006.

I would like to thank my supervisor Jenny Ulriksson at (FOI) for her excellent guidance and assistance and my examiner Rassul Ayani at (ICT/KTH) for his insightful and useful comments on the thesis and related papers.

Index

List of figures

List of tables

List of acronyms

ACM	Association for Computing Machinery
CREW	Concurrent Read Exclusive Write
CSCW	Computer Supported Collaborative Work
CVE	Collaborative Virtual Environments
DES	Discrete Event Simulation
DIVE	Distributed Interactive Virtual Environment
GUI	Graphical User Interface
HCI	Human-Computer Interaction
NTP	Network Time Protocol
SDK	Software Development Kit
UDP	User Datagram Protocol

1 Introduction

1.1 Motivation

Computer systems for collaborative work create virtual environments where geographical distances between co-workers become less important by providing a platform that can potentially facilitate and enhance the collaboration. These platforms offer a way for people to collaborate world-wide in an efficient manner [4] by allowing them to share documents and resources, interact simultaneously on objects and communicate in real-time.

When several users are sharing a document simultaneously there are various ways of synchronizing changes in order to maintain the consistency of the document (i.e. that everyone has a consistent view of it). When choosing a synchronization policy for a shared document, the strictness of the policy could be measured between the need of an accurate consistency and the need for high performance [8].

Since collaboration participants may change their interaction behavior during the collaboration it can be difficult to find a specific synchronization policy that is near optimal at all times. Therefore, in the opinion of the author, an optimization would be a mechanism that could alternate between different synchronization policies in real-time to best suit the behavior and needs of the users. This mechanism could monitor user interaction rates, delays, consistency needs, synchronization needs and other system properties, and with this information make an estimated guess of what synchronization policy would perform near-optimal in a given scenario. Among the parameters to evaluate the overall performance there are delays, throughput and traffic volume as they all have an impact on quality of the virtual environment due to the technical limitations behind it.

Computer-supported collaboration work (CSCW[1]) systems are complex in their nature since they provide platforms, consisting of many underlying mechanisms, where humans can collaborate in an efficient manner. These platforms support functionalities as people-awareness, consistency of data and user-views, replication of data and real-time communication in order for efficient collaboration to take place [4, 7]. Due to the complexity of all the parts working together it is not trivial to develop a mechanism, as described above, which could change between synchronization policies in real-time in a near-optimal fashion. Therefore, in the opinion of the author, a synchronization policy simulator, where the efficiency of a policy could be measured for a specific scenario, would serve as a useful tool in the development and support of such a feature. The simulator could be used to compare the performance of various policies in scenarios likely to occur and with the help of the simulation results guidelines describing the dynamics of the evaluated policies could be compiled.

[1] Systems that allow geographically distributed groups of people to share a working environment enabling them to collaborate towards a common goal.

1.2 Problem statement

In [1] Ayani and Ulriksson claim that most CSCW systems of today are mainly simple-purpose when it comes to the interaction on a shared object. This means that there often only exists one underlying synchronization policy, which has been designed for a specific kind of behavior. If users would interact in a different fashion it could cause unnecessary overhead, since the policy would be sub-optimal for the situation, resulting in a performance loss. For example, the synchronization policy of a simple shared whiteboard application may beneficially be event driven and relaxed since there is a small amount of interaction between users. On the other hand, if we look at a fast paced action game that is played cooperatively then the synchronization policy may be very strict in order to minimize overhead [1].

The aim of this master thesis project is to show the strengths and weaknesses of various synchronization policies in different scenarios in a CSCW system. The purpose is to create general guidelines that illustrate which policies are optimal for certain situations in a CSCW system and why. To evaluate how the policies perform under different circumstances they will be tested on a simulator platform developed by the author.

It is the opinion of the author that with such guidelines it could be possible to create more advanced CSCW systems that could handle shared multiple-purpose objects and interact with several users whose behavior may change over time. An enhancement of CSCW systems could be accomplished by developing a mechanism that monitors the behavior of the users in real-time and changes the synchronization policies accordingly, with the objective of keeping overhead and delays at a minimum.

Implementing a synchronization policy handling layer could make the interaction between users with heterogeneous and variable behavior more efficient, since the system would be able to adapt to the current behavior. It could also create possibilities for new collaboration models and also make simple-purpose systems more tolerant against network delays and unexpected behavior.

1.3 Problem limitations

Some limitations where done on beforehand regarding the properties and the scope of the simulator in order for the work to be performed within the scope of twenty weeks. The first limitation was that the amount of synchronization policies that the simulator should support did not have to be many as long as they where validated and extensively tested. Also the synchronization policies did not have to be interchangeable during runtime, since the main objective of this simulator is to test how a certain policy performs in a given scenario.

1.4 Goals

The main goal of this master thesis project is to develop a general simulation platform for simulation of synchronization policies in collaborative environments that is able to model many different behaviors and environments. The main purpose of this simulator is to act as a tool for testing and evaluating synchronization policies in different scenarios. Another goal of the thesis is to provide a good information base for future implementation of a synchronization policy manager. This is achieved by simulating various scenarios where many parameters are varied in relation to each other. Among these parameters we find: number of participants, reading time, synchronization need and rollback penalties. The purpose of these simulations is to find specific scenarios where a certain synchronization policy is more efficient than others.

Regarding the implementation of the simulator there are various sub-goals. The first is to validate the output and behavior of the simulator, which is crucial for being able to trust the results. Therefore there will be extensive testing in order to validate the correctness of the simulator. The second is to provide good documentation that could be used to quickly understand what the simulator does and how to modify and develop new features. The third is to create a simple and intuitive user interface.

1.5 Thesis outline

Chapter 1 gives an introduction to the subject; chapter 1.1 motivates why this thesis is important and what it can be used for, further it explains how a simulator could be used for developing a new mechanism that potentially could increase the efficiency of existing CSCW systems. In chapter 1.2 the problem that the thesis tries to solve is defined more extensively, followed by a limitation of the problem. Chapter 1.4 defines the author's goals regarding the thesis and the simulator.

Chapter 2 covers the technical basics that a reader needs to understand in order to fully comprehend the simulator implementation, the experiments and the conclusions. The technical background covers distributed systems, synchronization, computer supported simulation, CSCW systems and consistency along with some consistency related concepts.

Chapter 3 presents the simulator platform beginning with an explanation of the implemented synchronization policies and the general abstractions behind the simulator. Then an overview of the implementation is presented, followed by a guide of how to use the simulator.

In chapter 4 various experiments are performed with the aim of finding patterns between performances of the synchronization polices and the behavior of participants. Results are presented in graphs and are followed by an analysis of the results.

In chapter 5 the conclusions are presented. They consist of a summary of all the analyses in chapter 4 creating general guidelines which show during which circumstances the different policies perform well. Future work is also proposed in this chapter.

2 Background

This chapter covers the technical background needed to follow the ideas and concepts of the thesis. The first part covers the basics of distributed systems in general, followed by a description of synchronization and some related concepts. The following part explains the fundamentals of computer supported simulation and discrete event simulation. The next sub-chapter gives an introduction to CSCW systems and some interesting technical challenges that are interesting to address. The following sub-chapters discuss consistency management in non-CSCW systems and CSCW systems, relaxation of consistency and a comparison of consistency management in CSCW systems and other systems.

2.1 Distributed systems

Many of the services that today's computer systems provide are very complex and demanding and often the traditional centralized systems are incapable of delivering a satisfying solution. In [9] we find that distributed systems have the potential to scale the performance, be fault tolerant through redundancy and to handle geographical distances with the help of replication.

One of the widely accepted definitions of a distributed system is Andrew S. Tanenbaum's:

"A distributed system is a collection of independent computers that appears to its users as a single coherent system."

The main characteristics of a distributed system are the following [9]:

- Transparency; the system should appear as a centralized system to the users without distinction between local and remote resources.
- Fault tolerance; the system should not have a single point of failure; this is normally achieved with redundancy and fault recovering mechanisms.
- Mobility; users should be able to log into the system from different places and see the same environment.
- Scalability; it should be easy to expand the system to manage increasing workloads and it should also be able handle arbitrary geographical distances between users.

It is also stated in [9] that when it comes to designing distributed systems there are many important aspects to consider. Among these are topology (interconnection of nodes), communication (how do the nodes talk to each other), synchronization (usage of shared resources) and resource management. All these aspects make the development of distributed systems more complex and difficult than ordinary centralized systems. But still, the advantages often outweigh the difficulties.

2.2 Synchronization

Synchronization is needed in order to make sure that generated events (e.g. messages) are replicated across a system (i.e. distributed to all nodes) within real-time constraints and is also used to control and support distributed units providing a common time frame. A related concept in regard to synchronization is the causal ordering among events, which makes sure that events maintain there causal relationship (i.e. that they are received in the same order as they where sent) when replicated. Another related concept is concurrency which defines a system's ability to handle simultaneous events[14].

When it comes to CSCW systems the responsiveness[1] and concurrency of a system are in an inverse relation with the consistency level (how consistent the views of different users are). To maintain responsiveness and concurrency as high as possible the consistency level can be varied by providing just enough synchronization and ordering to meet current consistency requirements (i.e. perform consistency management) [14]. By providing sufficient synchronization and ordering the events are processed in the same order, states are updated equally at all nodes and results are computed equally at all nodes maintaining the distributed system in a consistent state.

2.2.1 Event ordering

When applying synchronization to distributed systems it gets more complicated than with centralized systems because of the absence of global physical clocks, global shared memory and the possibility of partial faults [10].

To synchronize physical clocks there are various synchronization algorithms. They either synchronize all clocks against a single one (accuracy) or they synchronize all clocks amongst themselves (agreement) [10]. Generally, in distributed systems, it is enough to agree on a common time even though it is not accurate with the real time. Also the timestamps themselves are not that important, but rather the ordering of concurrent events.

Another way of ordering events is by introducing logical clocks which do not have anything to do with real time but instead they keep track of the "happens-before" relationship between two events. Iif event A is sending a message and event B is receiving the same message the relationship "A→B" is defined, since B cannot happen before A. If neither "A→B" nor "B→A" is true then the two events A and B are considered to be concurrent "A‖B" as described in [10].

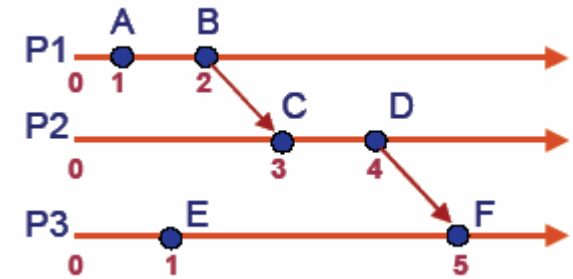

Figure 2.1

Figure 2.1 illustrates three processes executing events, among these events there is sending and receiving of messages which represented with an arrow between the processes' timelines. Here we can see the following relationships: "A→B", "B→C", "C→D", "D→F" and "A‖E".

[1] Responsiveness is the delay between the time an action is performed and the time that a user perceives that it has been performed [17].

One problem with logical clocks is that they do not capture causality (i.e. that events arrive in the same order as they were sent), meaning that if event A and B have the relationship "happens-before" then A must have happened before in real time as well. But if we know that event A happened before event B in real time we cannot know if they have the relationship "A→B". This can be solved using vector clocks which create timestamps with information about how many events have been processed at the other processes also. With this additional information and some rules regarding how to update timestamps it is possible to determine causality.

2.2.2 Mutual exclusion

Distributed systems normally have shared resources which only can be used by one participant at the time. To make sure that this rule is not violated a mechanism called mutual exclusion is introduced. This mechanism has two main properties, the first is safety which guarantees that at most one process may execute in the critical section at the time. The second property guarantees liveness, meaning that all the requests to enter or exit the critical section eventually succeed.

There are various ways to implement mutual exclusion in a distributed system; the easiest way is to emulate the centralized solution with a server that manages the lock to the critical section. The requests to enter and exit the critical section are done by sending a message to the process that has been elected as coordinator. The advantages with this implementation are that it is easy to guarantee mutual exclusion and fairness; the disadvantages are the possible performance bottleneck and the single point of failure.

Another way of implementing mutual exclusion is by creating a logical ring topology regardless of their physical connectivity. To enter the critical section a participant must have obtained a lock which is forwarded around in the logical ring in a given direction. There must also be a maximum time for holding the lock to guarantee liveness.

In many cases there is certain process that has a special role, for example the coordinator that manages the lock to critical section in the centralized solution mentioned above. Since this kind of solution creates a single point failure in the system a technique has been developed to elect a new coordinator in case the current one fails to respond. In order for the participants to agree on a new coordinator the different nodes have to be distinguishable (i.e. have identification numbers) and they need to have a time-out mechanism to notice that the coordinator is not responding. A simple algorithm to elect a coordinator is called "the bully algorithm", which simply elects the responding participant with the highest identification number to the new coordinator.

2.3 Computer-based simulation

Modeling and simulation can be done with several tools; among them we find pen and paper, a physical prototype or with the aid of computers. In many ways it is practical to simulate problems with computers since they can handle computation intensive tasks and there are many tools for modeling real problems. When using a computer to simulate the behavior of a system, the problem is abstracted into a model which can be run on a computer and where the results can be observed. Computer based simulation is normally done when creating a prototype is too expensive, unfeasible or impossible.

Simulation is generally used to a find the solution to a problem, for example how the fuel consumption of car varies for different kinds of driving behavior. Obviously, it would be possible to take a real car and measure the fuel consumption under different situations, but when creating a model of the problem it is potentially possible to simulate various car types, and many different scenarios. Thus, simulation offers more flexibility than testing with a prototype and has the potential to give a more detailed answer to the question at hand [15].

Richard M. Fujimoto's definition of a computer simulation [11]:

"A computer simulation is a computation that models the behaviour of some real or imagined system over time"

2.3.1 Discrete event simulation

A discrete event simulation (DES) system can be viewed as a collection of simulated objects and a sequence of event computations. An object consists of a number of variables that together form a system state which is changed through state transitions (i.e. events) that occur at discrete points in simulation time. The only way for a system state to change is through the processing of an event which may modify the state variables or schedule new events into the simulated future. Each event has a time stamp indicating when it occurs in the simulated system. During simulation the events are stored in a local event list and are processed in time stamped order [15].

2.4 CSCW systems

There are many definitions of the CSCW field; here is one of the most recent and explanatory ones by Ramage [17]:

"A combination of technology, people and organizations that facilitates the communication and coordination necessary for a group to effectively work together in the pursuit of a shared goal, and to achieve gain for all its members."

Figure 2.2 depicts the onion layer model, proposed by Ramage, that is used to illustrate the aspects of CSCW systems and the multiple research fields that are involved in the creation of these systems:

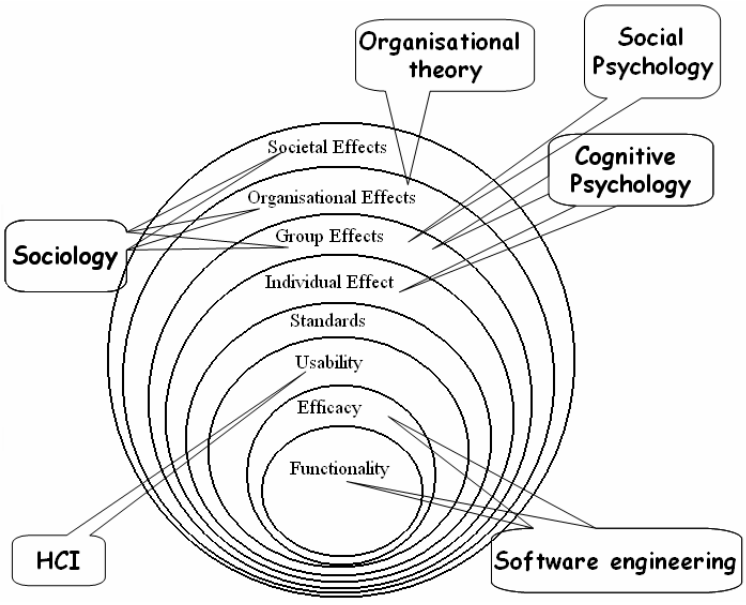

Figure 2.2

Figure 2.2 illustrates the onion model which shows the different layers of CSCW systems.

The innermost layer is functionality and concerns if the system works at all, in the second layer (efficacy) it is considered if the system works well enough in terms of performance. Both these layers are in the field of software engineering. The next layer is usability which considers if it is possible to work with the system in the perspective of a human user. This analysis falls into the field of human-computer interaction (HCI). In this field things like the human information processing model, short and long term memory and models of human reasoning are considered so that the system can be designed for efficient interaction. The following layer is about following the existing standards regarding document representation, resource access and communication protocols in order for the system to be compatible with other computer systems. The subsequent layer deals with individual effects i.e. how the usage of the system affects a single user, which is studied in the field of cognitive psychology. The following layer, group effects, studies the system's impact on the users' work as a group and is studied in fields of cognitive psychology, social psychology and sociology. The next layer treats the organizational effects of the system, meaning how the system affects those who employ or collaborate with the users of the system. The outmost layer analyzes the societal aspects of the system which concerns how it affects the world outside the work related issues [17].

The general dimensions of a CSCW system are the geographical locations (face-to-face or different places) and the time at which events occur (synchronous or asynchronous). For example, presentation support software as PowerPoint is synchronous and face-to-face as the audience receives the information instantaneously (although it is one-way-information) and they normally are at the same location as the presenter. On the other hand, in a mailing system the people communicating are normally at different locations and the communications is asynchronous as the messages are not necessarily received at the same moment as they are sent [17].

2.4.1 CSCW history

The concept of Computer Supported Cooperative Work (CSCW) was originally formulated at a workshop organized by Paul Cashman and Irene Grief in 1984. The workshop gathered people from many disciplines interested in how people work together and how technology could be used as an aid for this purpose. Earlier there had been attempts to create applications with group support, for example "Office Automation", but they had not reached success mainly because of the developers lack of understanding of how people work in groups and how organizations are built [12].

The people behind the concept of CSCW realized that the main problem was to understand the dynamics behind human collaboration. They gathered economists, social psychologists, anthropologists, organizational theorists, educators and others that could provide useful knowledge on group activity. This created a platform where also system developers could inform each other on technical possibilities and constraints when developing CSCW systems such as: desktop conferencing and videoconferencing systems, collaborative authorship applications, electronic mail and its refinements and extensions, and electronic meeting rooms or group support systems [12].

The first workshop where CSCW was first defined resulted in annual conferences in both Europe and North America which have become important meeting places for the CSCW community. On these conferences there are presentations of research papers, demonstrations of applications and workshops [16].

"The ACM CSCW conference which is a leading forum for presenting and discussing research and development achievements concerning the use of computer technologies to support collaborative activities, as well as the impact of digital collaboration technologies on users, groups, organizations and society." [16]

2.4.2 Main features of CSCW systems

The following features are the most important features of a CSCW system; shared context, awareness of others, negotiation and communication and flexible and multiple viewpoints [4]:

- Sharing context means having knowledge of other participants' current and past activities and knowing how different objects and environments are shared. This is needed in order to communicate with the correct participants when simultaneously working on a common object.

- Awareness means to understand the activities of others and the intention of the activities being performed. When a participant is aware of the other team members' activities and intentions it is easier to choose meaningful activities that contribute to a general objective.

- Negotiation and communication is crucial for creating a task structure, in terms of roles and activities, which is essential when collaborating with complex tasks.

- Flexible and multiple viewpoints are essential because information must often be represented in various ways in order for certain tasks to be done. In some situations it is also important to be able to customize the information representation to distinguish information that is specific for a task.

2.4.3 Technical issues in CSCW systems

To create computer supported collaborative work environments that are easy to use, adaptable to many types of collaborations and that scale well, there are many technical challenges to overcome. Here follows a list of some important technical issues in the opinion of the author:

- Support for people-awareness in CSCWs [4]. A part of this field is that users have a way of finding out who is browsing the same document as them or if someone is modifying an object that they are currently working with. This real-time feature has the potential to increase the quality of collaboration by making the environment more interactive and dynamic.

- Creating tools that can be used by several users at the same time and how to approach this problem for the general case.

- Handling concurrent interactions between users, since rolling back the system state when humans are involved is complicated because the user does not forget what happened. Furthermore, locking an object for mutual exclusion can make the system more difficult to use.

- Updating system state efficiently across all nodes without overloading the nodes with traffic and making the system less responsive.

- Managing the consistency level in an efficient manner, i.e. choosing the synchronization policy that fits the current collaboration scenario best considering the behavior of the users. This thesis intends to provide a useful tool and a good analysis of the dynamics behind the synchronization policies and the user behavior, for solving this problem in the general case.

2.5 Relaxation of consistency

This chapter discusses how one could vary the strictness of synchronization policies and how to relax different performance related parameters in the simulator.

2.5.1 How to vary the level of consistency

The least strict policy is one where there is no synchronization at all. It simply lets the participants do whatever they want without any restriction. The policy can be useful in a real application if real-time constraints are high and the need for consistency guarantees is low. It can also be used to compare performance with other stricter policies as it represents the best case scenario. The comparison can show how close the compared policy is to the ideal solution.

The above mentioned synchronization policy can be made stricter by modeling a synchronization need at each participant. For example, one could model that each participant needs to synchronize every time it has processed a certain number of events (this number could be randomly generated within a given a range of parameters) or after a number of time units in the same manner. When synchronization occurs the synchronizer must wait until all other participants have reached the synchronizing state and then rollback those who have advanced further. When doing a rollback one could introduce a rollback penalty, meaning that the system halts for a number of time units modeling the cost of doing a rollback in a real system. This is done to model the time it takes for a real system to restore a state.

Another important thing to consider after a participant is rolled back is if one should let it continue where it was before the rollback after processing the synchronizing event or just let it continue from there afterwards. The main approach is to let the participant who is rolled back restart from the synchronizing event as the document could have been changed during synchronization.

One way of making a policy even stricter is by introducing synchronization barriers at fixed places in a document (for example after a certain amount of events), meaning no participant can advance to the following event until all other participants are done with that event (i.e. have reached the synchronization barrier). One could also make all the participants periodically synchronize after a certain amount of time units.

2.5.2 How to vary performance related parameters in CSCW systems

There are various ways of enhancing the performance of a CSCW system other than modifying the synchronization policies, which is discussed above. When talking about consistency management in general there are some basic assumptions about the system in order to keep things simple. According to [7], these assumptions are that the virtual world is (although there are several CSCW systems without replication) fully replicated, all communication is reliable, all events are seen by all participants in exactly the same way, all objects have a single history of modifications, all participants wish to see the worlds content in the same way and there is no world versioning. Each of the stated assumptions can be relaxed in order to optimize the system's usage of resources and reduce consistency and at the same time fulfill the needs of the users. Here follows an explanation of how each of these properties can be relaxed.

- The replication can be relaxed by using partial replication where the virtual world is divided into segments and then each participant joins a subset of these segments which becomes its entire virtual world. With this technique the participants only need to be aware of information that is relevant to them and thus reducing the usage of system resources considerably. This comes at the price of increased complexity in respect to handling new segments, consistency between segments that have the same participant as member and not creating dependencies on segments where a participant is not a member.

- In some applications (for example multimedia streaming) it is more important to have a good flow of information with an occasional loss of packets than having a reliable communication with high delay. Therefore, it is common to use unreliable transmission protocols as UDP in applications that send and receive streams of data. Using this kind of protocols can threaten the consistency of the system since there is no guarantee that all participants receive the same messages. Therefore, it is important to combine unreliable transmissions with a control mechanism that can restore the system's state if it becomes inconsistent.
- A common way of reducing communication is having a behavior approximation mechanism that approximates the behavior or movement of a participant and thus requires less information at the cost of marginal inconsistency. As with unreliable communication, this must be combined with a mechanism that can restore consistency.

- When it comes to object modification it is common that an object may have only one owner at a time, which is the only one authorized to modify the object. This policy is good for maintaining consistency but does not allow concurrent modification, which in many cases can hurt performance. To relax the requirement of a single history of modifications we introduce an alternative, which is to use optimistic updates instead. This lets several participants modify an object concurrently although it needs a mechanism to restore the object from an inconsistent state. In order for this approach to be beneficial it is important that the concurrent modifications not often result in conflicts and that the overhead for restoring state is small.

- Sometimes participants in the same shared context need to emphasize on different details. To free resources the system should filter unnecessary details from each participants view creating several subjective views. When using subjective views it is important that there are possibilities to share, modify and reason about different subjective views among the participants.

- In some cases there is a need to create a new version of the virtual world where participants want to modify it in different directions concurrently. This can be accomplished by having different versions of the virtual world.

2.6 Consistency differences between CVEs and other systems

Collaborative Virtual Environments (CVEs) are systems which normally have audio-graphical interfaces where multiple users simultaneously can explore, communicate and collaborate with one another and interact with the virtual environment. Normally each user accesses the CVE through their own computer and is connected to the rest of the participants through one or several networks.

Consistency management has been studied in many areas of computer science, especially database systems. But since CVEs distinguish themselves from traditional systems in many aspects, there is a need to change some of the traditional ideas and techniques when it comes to managing consistency. According to [7] some of the most important characteristics of CVEs are the following:

- The requirement for short delays, in contrast to database applications that normally have focus on high throughput instead.

- The difficulty of rolling back states, since CVEs involve human users which receive output in real-time from the CVE and who also give input in real-time based on what they perceive. The state of the users' perception cannot be rolled back since we cannot forget on demand, and changing something in the virtual environment unexpectedly could cause confusion for the users.

- Some CVEs have aspects linked to the “wall clock time” of the physical world and must update their state consistently even in the absence of explicit events. The state of computer simulations and database systems remains static when no explicit events are processed.

- In some cases shorter delays are more important than absolute correctness, mostly because humans in some cases can tolerate a certain lack of correctness better than unexpected delays. For example, when receiving real-time video streams where some packets contain structure of image and other image colors, it is generally better to lose a limited number of packets at the cost of video quality (color information) than to stop the video stream until all the lost packets have been re-sent.

2.7 Related work

Chapter 2.7 discusses aspects of existing systems which are related to CSCW systems and consistency management. The main criteria for selecting the following systems has been that they all illustrate difficulties of CSCW systems. By discussing existing CSCW systems and how their consistency issues were solved it becomes easier to understand how these systems could benefit from the results of this thesis.

2.7.1 Development toolkits for CSCW systems

Many CSCW systems have been developed but not many have yet been really successful. One of the main reasons is the complexity of implementing key mechanisms as consistency and group management. One of the reasons for these difficulties is the lack of supportive frameworks and software when it comes to implementing consistency in CSCW systems.

According to [2] one of the key aspects of toolkit design is the reusability of components in several applications and circumstances. But it has been proven hard to create reusable shareable objects for CSCW system toolkits. Two reasons for this are that an object can be shared in many different ways and the that users may have different behaviors. The choices of replication strategies, locking, conflict management and other crucial mechanisms for a shared object define in what situations and in which way the object can be shared. This makes it difficult to create toolkits supporting universally shareable objects.

The Prospero toolkit for designing CSCW systems has addressed this problem by providing a metalevel interface which gives the application programmer control over synchronization mechanisms, replication strategies and other important properties of shared objects. This way the programmer can specify characteristics of the shared objects and create an application that is more flexible for different situations and that to a greater extent fulfills the needs of its users.

2.7.2 Consistency management in Sombrero

The distributed operating system Sombrero [5] uses several consistency policies in order to maintain the data integrity. Among the supported policies we find the strict policy which is the strongest possible consistency model and that applies all changes conservatively. Then there is the sequential policy which appears like the strict but is easier to implement since it only makes sure that all nodes see exactly the same write sequences. We also find weaker consistency policies that only guarantee consistency after certain barrier synchronization. Between these barriers each node's writes are invisible to other nodes and it is the responsibility of the application to merge the writes in a correct way at the barriers. One of these weaker concurrency control models is called Concurrent Read Exclusive Write (CREW) which gives several nodes the possibility of concurrent writes on the same memory space without the classical mutual exclusion. Consistency is achieved by assigning unique memory areas to each writer which are not visible to other writers. When the application reaches a synchronization barrier the writes are merged with preserved consistency. This policy boosts performance by allowing simultaneous writes to the same page, but needs a mechanism to merge the writes correctly.

Sombrero supports choosing a specific consistency policy for each object; this tends to work well with objects that are accessed in a predictable manner because one is able to choose the optimal policy for the expected behavior. But since it is not possible to switch policy during runtime this does not work as well with objects that are accessed in an unpredictable manner.

2.7.3 Consistency management in a Wide-area caching Data Store

In the wide-area data store Khazana, described in [6], it is crucial to maintain data consistency across all nodes holding replicated data. To achieve consistency Khazana implements several consistency policies that are suitable for different situations.

The nodes in the data store system are hierarchically organized with child and parent nodes, at the top of the hierarchy there is a root node which contains the master copy of all the different files. Changes in the system are propagated upward until they reach the root node. Then, if they are accepted, the changes are propagated downward to the whole system. The following policies are applied on per-file basis in the system based on the properties and needs of each file.

- The first synchronization policy is the strict pessimistic which only allows one writer in the system at the time.

- The second is the optimistic last-writer wins which allows reads and writes at any time and at any replica, and then the changes are propagated to the root replica and write conflicts are resolved along the way. The last write to reach the root replica is accepted and propagated to all the nodes. When propagating changes sometimes write conflicts occur. They can be handled in many ways; either they require manual intervention, rely on resolution rules applied by the system or require application specific techniques. Specifically for this policy, incoming changes are incorporated (i.e. merged) with the local copy.

- The third policy is called append-only consistency and allows appends to files at any time. If there are two concurrent appends to replica (from child nodes) they are applied in the order of arrival from the child nodes at the parent node replica.

2.7.4 DIVE

The Distributed Interactive Virtual Environment (DIVE) [18] is an Internet-based multi-user virtual reality system where participants navigate in 3D-space and see, meet and interact with other users and applications. DIVE used full world replication in its first implementations but because of scalability difficulties the policy was changed to partial world replication. This means that a participant requests the objects that it wants to "exist" in its perception of the world, normally all objects in sight are requested. When an object is requested the system finds the participant who is closest to the requester and that has the most recent version and makes it send an up-to-date replica of the object to everyone who has the object in their world through a multicast message. There is also a world server listening to all the multicast messages which is responsible of making all the local changes global and persistent. The consistency approach is conservative and is implemented through a per-object token-passing algorithm and is best suited for an environment where participants own objects for a long time and concurrent modifications seldom occur.

2.7.5 CVE

CVE [13], also known as Massive-2, is a virtual environment based on the main concepts world, artefact and group. An artefact is defined as an object that is able to change world and change group within a single world. A group is defined as a group of artefacts or a region within a world.

When a process starts observing an artefact a local copy of the artefact is created, but there is only one master object residing in the process that owns the artefact. There is no general locking policy in the system, so each of the master objects representing the artefacts must implement their own synchronization policies. The master object handles all changes and performs source ordering on the incoming messages before applying the changes. Then all the local copies mimic the behavior of the master object as soon as they receive updates from it. There is no global clock in the system, so when delays between master and local copy occur the system state may become inconsistent.

2.7.6 MiMaze

MiMaze [3] is a multi-user maze game where the users control pacman-like figures and score points by eliminating each other. The software is totally distributed and the peers use unreliable multicast to communicate with each other in order to minimize message delays (below 100 ms) and to support a good scalability. Since the transmission protocol is unreliable the information sent is not incremental but continuous, meaning that the system tolerates a certain amount of dropped packets due to redundant information in subsequent packets.

The bucket synchronization mechanism is used to process commands that have been issued at the same time. By dividing time into evaluation periods and associating a bucket with each period, all commands issued at the same time fall into the same bucket. Sometimes commands are delayed to allow synchronization with other commands issued at the same time that have been delayed due to network lag. By ordering all the commands in this fashion the synchronization mechanism provides a good level of consistency. Also the buckets can be updated at different frequencies, for the illusion of movement to the human eye at least 25 Hz (updates every 40 ms) are needed.

Since the bucket algorithm does not handle collisions efficiently, they have to be detected before they happen. This is done by having the system calculate the distance that the players can cover within one time period based on their speed. The space within this distance is called the collision domain and if several users share one of these collision domains the system prepares itself to handle a collision. The system also computes the expected network delay of messages coming from different sources by using a global clock that is calibrated with an NTP[1]-like algorithm based on round-trip time.

[1] The *network time protocol* (NTP) is a protocol used to synchronize the internal *clocks* of *hosts* and *routers* on the Internet.

3 SynchSim – A platform for experimenting with various synchronization policies in CSCW systems

3.1 Design

The SynchSim simulator is developed for testing how selected synchronization policies perform under different circumstances. The main goal is to develop a general simulation platform for simulation of synchronization policies in collaborative environments that is able to model many different behaviors and environments. Another goal is to produce a powerful tool for making a real-time multi synchronization policy selector, which changes the synchronization policy of a CSCW system based on the user's interactions with the ambition of optimizing delays, throughput, etc.

The main abstraction is that all participants are sharing a document that is divided into pages which are read simultaneously and when there is a need for synchronization all participants must be on the same page before being able to continue. This is achieved by awaiting participants who haven't reached the synchronizing page or rolling back those who have gone further in the document or a combination of both. The idea with this abstraction is to make it as general as possible in order for the simulator to be able to model many different CSCW systems.

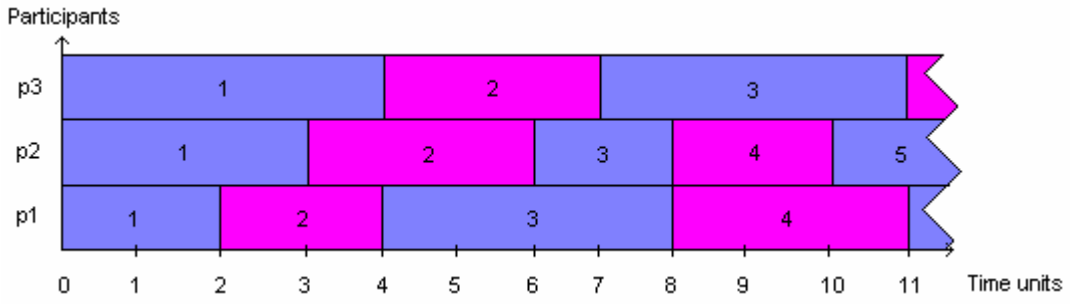

Figure 3.1

Figure 3.1 is an illustration of the page abstraction with three participants that are reading a document without synchronization. The numbers inside the boxes represent the page numbers.

Using this abstraction, it is the event that takes time (i.e. reading the page) and the event switching (continuing to next page) is done instantly. This is equivalent with another common abstraction, illustrated below, where the time between events is measured and the events themselves are executed instantly.

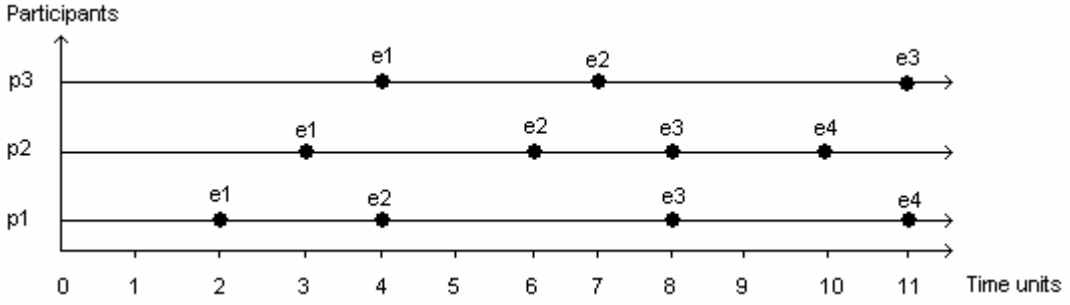

Figure 3.2

Figure 3.2 is the equivalent of 3.1 but the events are the page switches and the time between the events is the time it takes to read the pages.

In the simulator each participant has its own reading velocity, given as range of values and a statistical distribution used to randomly generate the reading time of each page. For some synchronization policies (such as the relaxed) each participant also has its own interaction need, which shows how often a participant needs to synchronize. The interaction need is defined as a range of values and a statistical distribution used to randomly generate the amount of pages between two synchronizations during runtime.

3.1.1 Choice of synchronization algorithms and policies

Conservative synchronization algorithms avoid violating consistency whereas optimistic allow violations to occur but provide mechanisms to recover to a consistent state. They are both useful in different circumstances. For example, if there is much interaction and one could expect that the local causality constraint would be broken often or if the overhead of the recovering mechanism is large, one would probably benefit from using a conservative synchronization algorithm, since every violation of the causality constraint would cause the system to perform expensive recovering actions. On the other hand, if there is little concurrent interaction and the overhead of the recovering mechanism is small, it is probably beneficial to use an optimistic synchronization algorithm. Both kinds of synchronization algorithms have been included in the simulator.

3.1.2 Simulation measurements

When simulating different scenarios there are some key measurements. The first one is the time that it takes for the last participant to complete the document. The second measures how long it takes for each participant to complete the document. The third is the number of time units each participant has been forced to wait for others when synchronization occurred. The fourth, which is special for the relaxed and hybrid policy, is the number of rolled back pages for each participant during the advancement of the document. The fifth, also specific for the relaxed and hybrid policy, is the amount of lost time units on rollbacks, meaning the sum of time units passed since a rolled back participant started on the page it is rolled back to. Additional measurements could be added in the future in order to obtain more detailed information of the participants' behavior.

3.1.3 Validation of simulator correctness

In order to be able to trust the output of a simulator it is important to do many experiments testing different scenarios and manually control that the simulator has done exactly what it was supposed to. It is also of importance to analyze the results in general and check if they match what is expected.

For testing the simulator two features have been implemented. The first is a debugging mode where every action taken by the simulator is logged so one can trace the source of possible bugs and also follow the simulation manually and double check the computation. The second feature provides the possibility to feed the same random number stream into different simulations which can be used to find scenarios where synchronization policies behave abnormally.

3.2 Implemented policies

3.2.1 Strict

This policy uses a conservative synchronization algorithm that enforces synchronization on every page. No participant may start on a new page until all other participants are done with the current page. Using this policy the pace for traversing the whole document is set by the slowest reader. But since the policy is conservative there is never a need to rollback any participant, so it can be beneficial when rollback costs are expensive or when the participants need to synchronize often. In the figures 3.3-3.6 the white spaces between the pages represent the time a participant waits for synchronization.

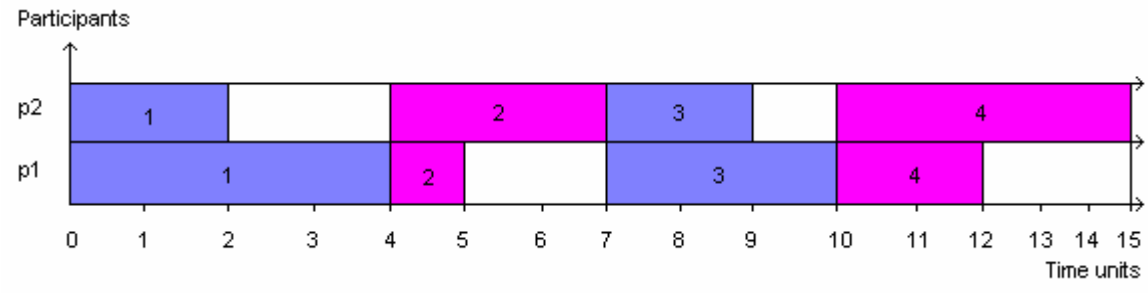

Figure 3.3

Two participants reading a shared document using the strict policy with the reading times shown in table 3.1.

Table 3.1

The reading times for the two participants illustrated in figure 3.3 and the barriers created by the strict policy.

Participant 1	Participant 2	Barrier
4	2	4
1	3	7
3	2	10
2	5	15

3.2.2 Relaxed

When using this policy the participants make synchronizations on demand. Otherwise, they are free to advance. This policy uses an optimistic approach which works best when there are few rollbacks and little overhead. Every participant has a certain need for synchronization which is modeled by generating random numbers within a given range and letting these numbers be the gaps between synchronizing pages. When synchronization occurs (when a participant reaches a synchronizing page) the participant who triggered it waits until all other participants have reached that page and then forces those who have advanced further to rollback to the synchronizing page. The only exception is when another participant forces the awaiting participant to rollback further back. In this case the first synchronization is canceled.

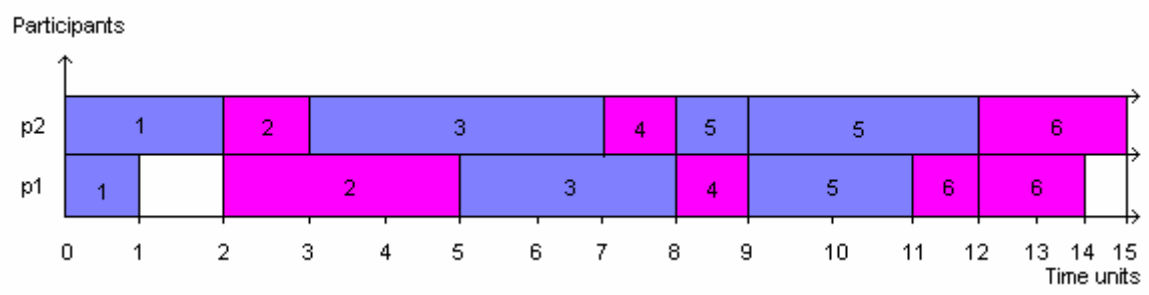

Figure 3.4

Two participants reading a shared document using the relaxed policy with the reading times and synchronization needs shown in table 3.2.

Table 3.2

Reading times and interaction needs for the two participants illustrated in figure 3.4.

P1 read time	P1 synch. page	P2 read time	P2 synch. page
1	2	2	2
3	5	1	6
3	7	4	8
1	-	1	-
2	-	3	-
2	-	3	-

3.2.3 Hybrid

This policy uses mechanisms from both the strict and relaxed policy. It has barriers where all participants must synchronize and between these barriers it behaves as the relaxed synchronization policy with on-demand-synchronization. The distance between the barriers is variable and can be adjusted to fit different scenarios, in figure 3.5 the distance between the barriers is three pages.

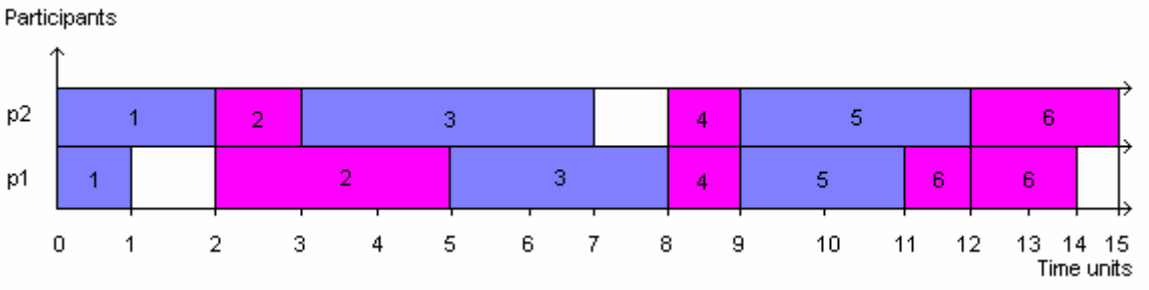

Figure 3.5

Two participants reading a shared document using the hybrid policy with synchronization barriers every 3 pages with the reading times and synchronization needs shown in table 3.3.

Table 3.3

Reading times and interaction needs for the two participants illustrated in figure 3.5.

P1 read time	P1 synch. page	P2 read time	P2 synch. page
1	2	2	2
3	5	1	6
3	7	4	8
1	-	1	-
2	-	3	-
2	-	3	-

3.2.4 Totally relaxed

This policy does not synchronize at all and is included only to show what the best case scenario could look like. Participants advance freely on their documents without regard of what the others are doing.

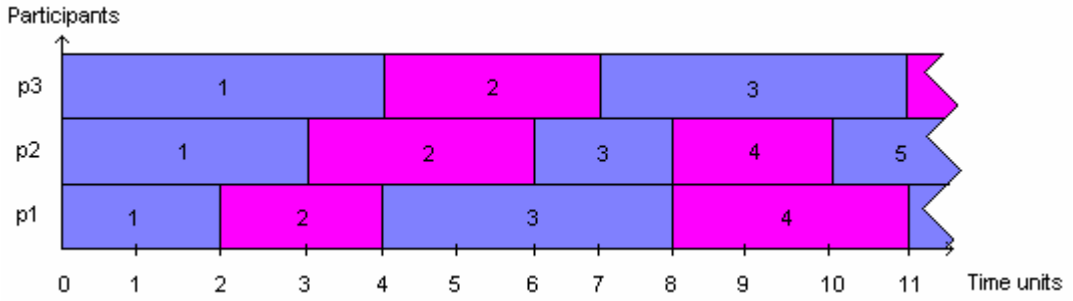

Figure 3.6

Three participants reading a shared document using the totally relaxed policy which doesn't synchronize at all.

3.3 Implementation

The simulator platform and the graphical user interface were implemented using Java (Java 2 SDK standard edition, version 1.4.2).

3.3.1 The simulator platform

The data structures used are global vectors containing information about the participants in specific positions (i.e. the information about the first participant is stored in position zero in each vector and so forth). The application takes a configuration file as input which is parsed and parameters such as reading times, interaction needs, policy and others are extracted and stored internally in variables. After initialization the run method is executed a number of times specified by the iterations variable in the configuration file. After each iteration the measured variables (such as completion time, number of rollbacks etc.) are accumulated in vectors that store the total amount of each measured variable. The vectors are later used to calculate and present the averages in an output window.

In the run method the first thing that is checked is the synchronization policy, if the strict policy is chosen the method enters a different branch than if the relaxed or hybrid policy would be selected. The strict policy is very straightforward in its implementation; reading times are generated for all participants then the clock is incremented with the greatest of the generated values. The following step is to calculate the time period that each participant has waited for the other participants to reach the synchronization barrier; this value is accumulated to the appropriate vector which is presented as output after the simulation. Finally, the page number is incremented and the process is repeated until all the pages in the document are read.

When the hybrid policy is selected, the simulator enters a loop that executes until all participants are done with the document. First the reading times and interaction needs (number of pages between synchronizations) are generated for each participant, next the hybrid barrier is computed which no participant may cross until everyone has reached it. The next step is to check if any participant has finished its page on the current time unit; if so it is flagged in the *doneWithPage* vector. The following step is to see which participants have reached the hybrid barrier; this information is stored in the *needsToWait* vector. The next step is to check if everyone has reached the hybrid barrier, if so a new barrier is computed along with reading times and, if needed, interaction needs. After these steps the *checkSynchNeed* method is called, this method checks if any of the participants that is about to turn a page needs to synchronize. In case of synchronization the method checks if any of the participants hasn't reached the synchronizing page yet; if so, it creates a barrier that cannot be crossed until everyone has reached it else it rolls back those who have gone further and generates new reading times and interaction gaps. Next the simulator checks if there is some participant that has several pages to reach the next barrier, if so it advances those participants if they are done with there current pages and calls checkSynchNeed again to see if any of the newly advanced participants needs to synchronize. The following step is to check if any of the participants have finished the document, if so it is flagged in the vector *doneWithDocument*. The last step is to update the global time, either by advancing to the time for next event or incrementing it by one time unit if the time for the next event is unknown.

The relaxed policy works exactly the same except for the hybrid barriers which are disregarded. During runtime the output is written to the file "C:\tempDir\output" and when the simulation is done it is loaded and presented in the output window.

3.3.2 Graphical user interface

The graphical user interface (GUI) was designed to be as simple and intuitive as possible. It was implemented with NetBeans 4.1, which has good tools for generating GUI's in Java, using the *javax.swing* library. The input for the simulator platform is gathered through a series of *JSpinners* and *JRadioButtons* in the GUI from which a configuration file (as seen in appendix B) is generated when the Run button is pressed. This configuration is stored in the file "C:\tempDir\confFileTemp" (storing the file at this specific location may prevent the application from running correctly on systems with file systems different from Windows') and is later used as input for the simulator class.

3.4 Usage of the simulator platform

Through the graphical user interface the configuration is easily set for a scenario to be run in the simulator. After the simulation is computed the results are presented in a separate output window. In the first part of the GUI the synchronization policy is chosen. The digit next to the Hybrid option is the static distance between barriers as described in 3.2.3. In the second part of figure 3.7 the amount of participants is set in a range between one and twenty. Furthermore, one can choose if all participants should be statistically identical in terms of reading velocity and interaction need or if they should have their own values. If they are set to be statistically identical the ranges for reading velocity and interaction need are set in the second part of the GUI, otherwise these parameters will be set for each participant through dialog boxes.

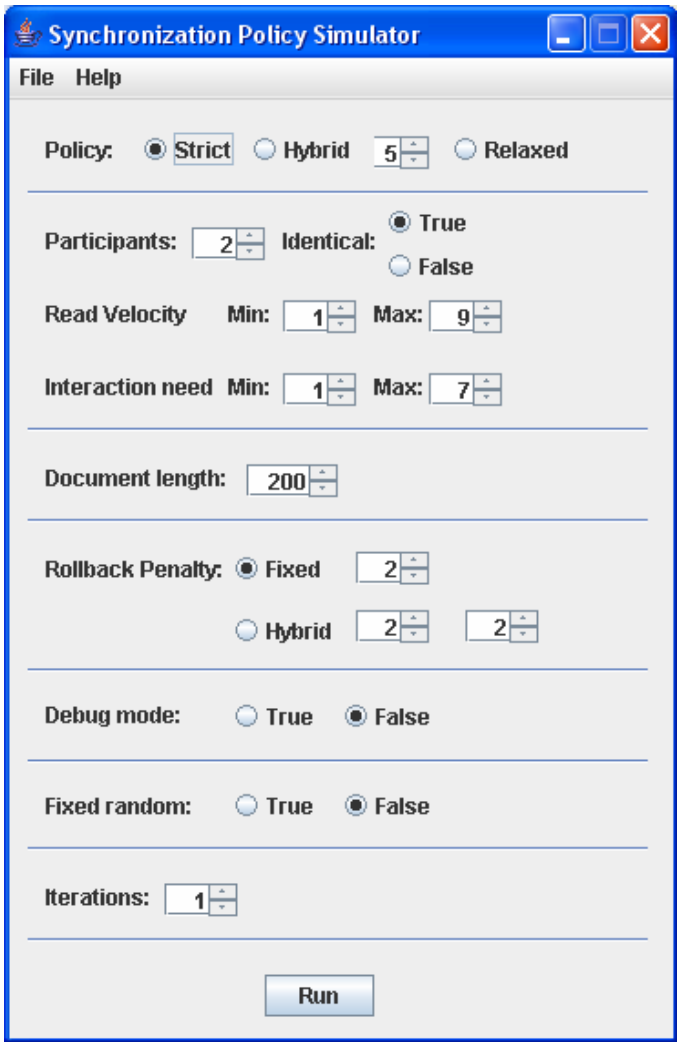

Figure 3.7 – *The graphical user interface*

In the following section the length of document is set in a range between one and 2045 pages. The fourth part of the GUI sets the rollback penalty; the first choice is to select a fixed or a hybrid rollback. The first digit next to the fixed option is a divisor for the mean reading time of a page. For example, if the minimum reading time is one and the maximum nine then the mean reading time would be five. If the divisor is two then the rollback penalty would be computed to 2.5 and then rounded to two time units. The first digit next to the hybrid option is exactly the same as the one next to the fixed explained above; the second digit is the page divisor which divides the number of pages rolled back. For example, if a participant rolls back two pages and the second divisor is three then the partial rollback penalty would be computed to 2/3, then it would be added to quotient of the mean reading time and the first divisor and lastly that sum would be rounded to the nearest integer and becoming the rollback penalty (in time units) to be applied.

In the case where no rollback penalty is wanted one can simply choose fixed hybrid (in the fourth part of the GUI) and set the divisor to a number greater than twice the mean reading time. This will make sure that the rollback penalty always will be rounded down to zero. In the fifth part one can choose whether to run the simulation in debug mode or not; in debug mode every action is logged and can be used to find vicious bugs in the software. In the following part one can choose to use a fixed random stream which can be used to feed the same input to various simulation runs. This can be useful to compare different synchronization policies observing how they perform in the exact same situation. This feature has some limitations:

- The maximum length of a document is 100 pages
- The maximum amount of participants are ten
- It only supports the mean synchronization needs one (1-1) to five (1-9)

The last option is to set the number of iterations to execute in a range between one and 999. It can be useful to run several iterations in order to eliminate statistical anomalies from the results, since it is the mean value of all iterations that is presented.

When choosing “File” in the top menu of the GUI various options are presented. The option “New” resets all values and buttons to their default state which is shown in the image of the graphical user interface above. Next is the option “Load” which lets the user load a configuration previously saved in a file. Further is the option “Save” which lets the user save the current configuration into a file. The last command is “Exit” which simply exits the simulator.

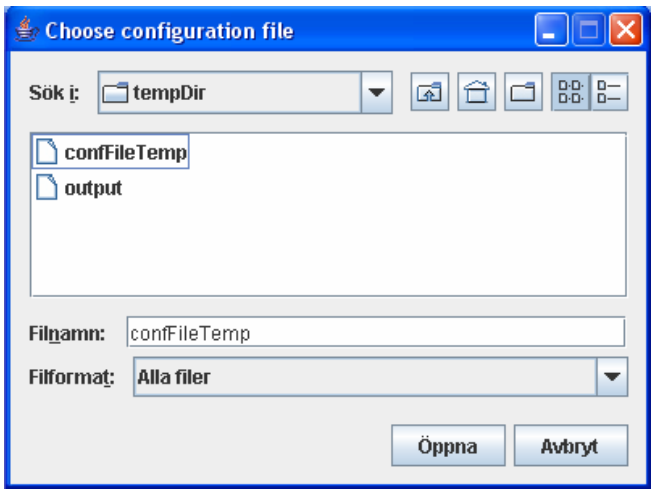

Figure 3.8 – *The load dialog*

Figure 3.9 shows part of the output window from a simulation run. All the statistics shown are the averages over the iterations run. The max finish time refers to the time it takes for the last participant to finish the document. The finish time is the time it takes for each individual participant to finish. Time waited on synch means the time a participant must wait at a synchronization barrier before being able to continue. Pages rolled back are the total number of pages a participant has to go back when another participant that is behind issues synchronization. Lastly there is the number of time units lost on rollback, which is the sum of the differences of the time of rollback and the time where the participant started the page it is later on rolled back to.

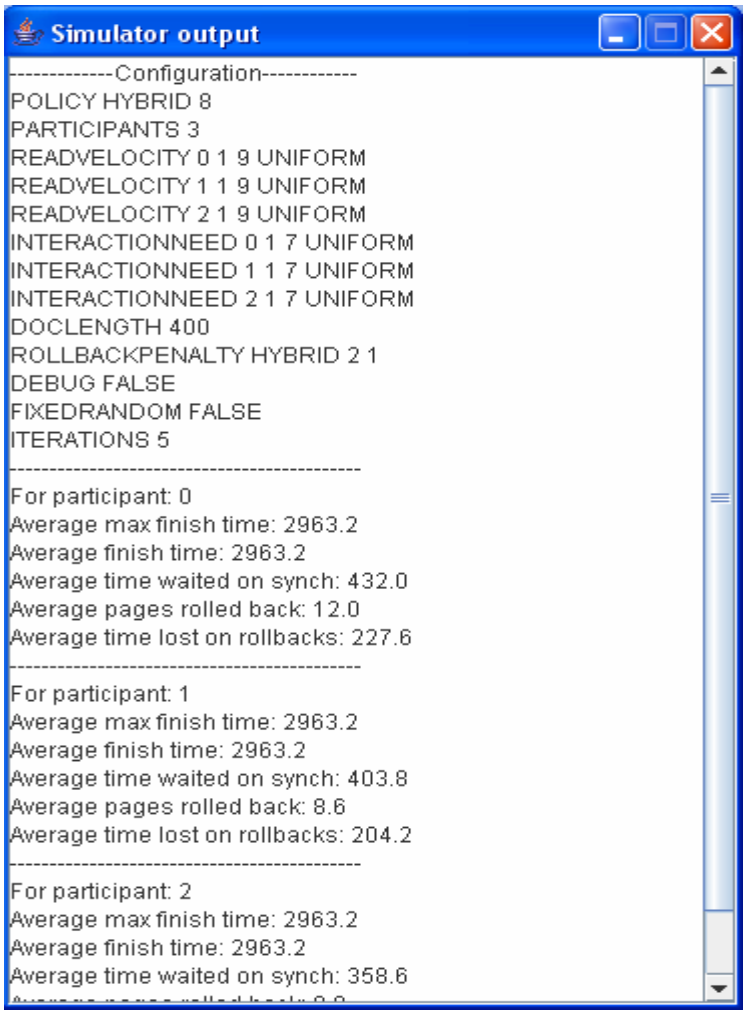

Figure 3.9 – *The output dialog*

Figure 3.10 shows the dialog that is displayed when the participants are configured to behave different. The dialog allows the user to set the read velocity and interaction need for each user in an efficient manner. It is important not to use the button at the right top corner to close these windows; instead the Set button should always be used. Otherwise the application will not receive the configuration for all participants and the procedure will have to be repeated.

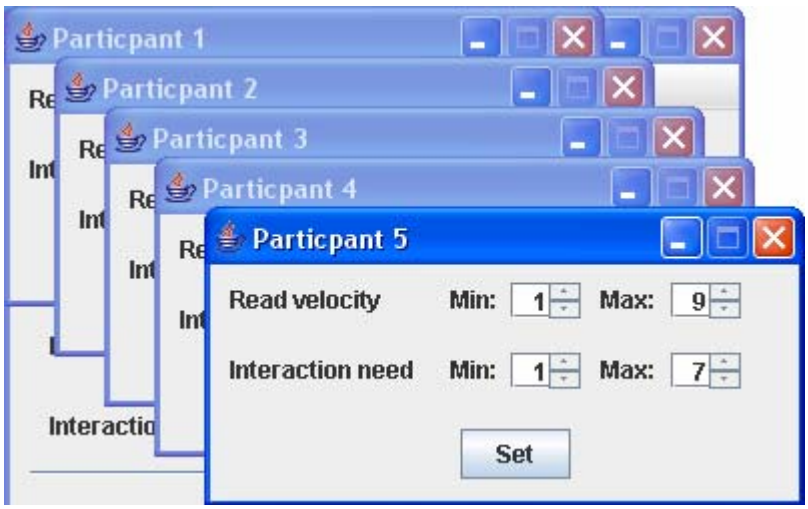

Figure 3.10 – *The heterogeneous participant configuration*

Installation instructions for the simulator platform can be found in appendix A.

4 Experiments and results

In this chapter various scenarios are simulated. In each experiment one parameter is varied to investigate its impact on the performance of the different synchronization polices. The purpose of these experiments is to combine the results into guidelines which could explain the dynamics behind the different policies. The guidelines could be used for design and implementation of a real-time synchronization policy chooser. In particular the performance of three policies ("Strict", "Hybrid 5", and "Relaxed") will be evaluated in the following experiments. At first the policies "Hybrid 3" and "Hybrid 8" (barriers every three or eight pages) were also tested. But they behaved very much like "Hybrid 5" and did not provide interesting information and thus were removed in order to make the graphs easier to read. In the following experiments the rollback penalty will be denoted in terms of T and N where T equals the mean reading time and N is the amount of pages that a participant rolls back.

4.1 Homogeneous participants

4.1.1 Rollback penalty

The purpose of this test is to analyze the impact of rollback penalties on the total completion time (the time it takes for all participants to finish the document). In this experiment we have three users that concurrently read a document of 200 pages with an average reading time of five (1-9) time units per page and with an average synchronization distance between one (1-1) and seven (1-13). The completion times shown in the figure 4.1 are an average measured over 20 iterations and the rollback penalties are based on average reading time rolled back pages.

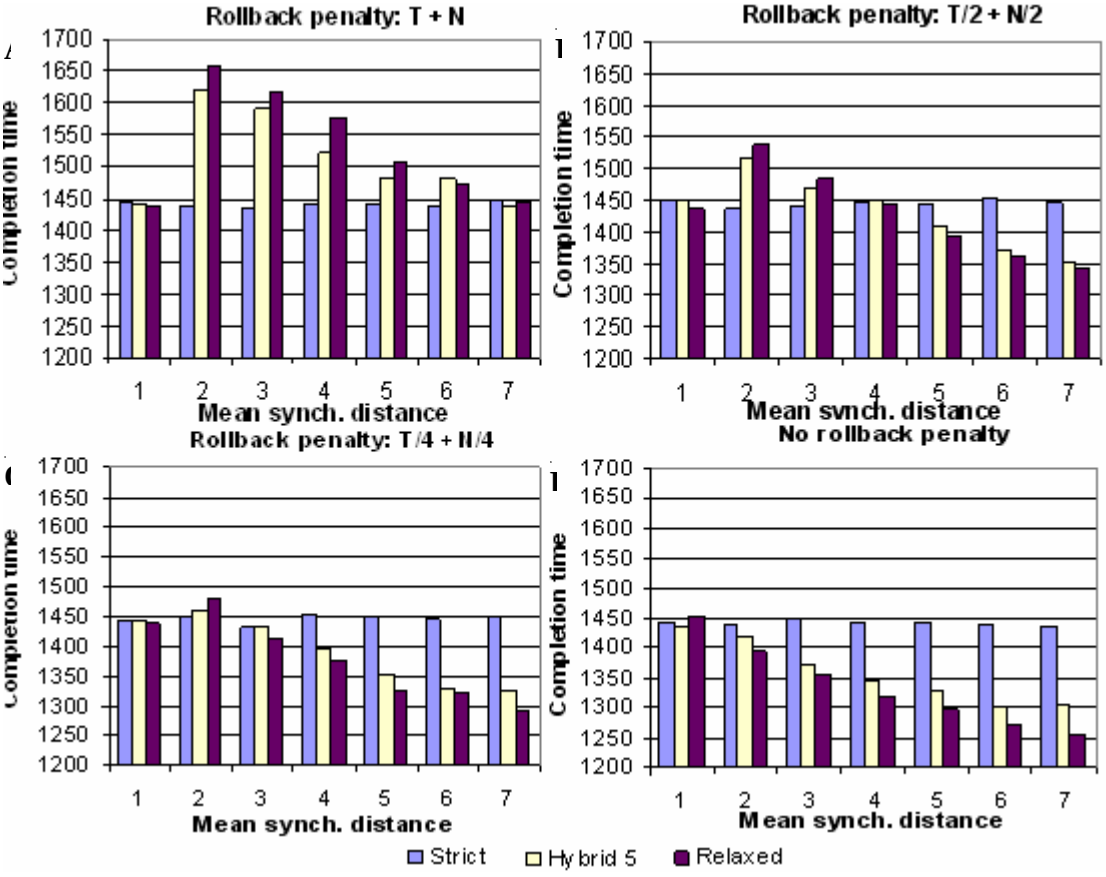

Figure 4.1

Completion times for various policies with different synch. distances and rollback penalties.

For all the graphs above we can see that the completion time is constant (around 1450 time units) when using the strict synchronization policy due to the fact that the policy is not influenced by rollback penalties or synchronization distances. Observing graph A we can see that there is an advantage using a strict policy when the rollback penalty is large, especially when the synchronization distance is small since that causes more synchronization situations where rollbacks can occur. We can also see that when the rollback penalty becomes small as in graph C and D there is a benefit in relaxing the synchronization policies since the time lost on rollbacks is less than the time gained when having less synchronization barriers where participants have to await each other.

4.1.2 Number of participants

This experiment aims to analyze how different synchronization policies perform when there are various numbers of participants reading a document concurrently, the rollback penalty for the participants is also varied to further investigate its importance. In the experiment the participants concurrently read a document of 200 pages with an average reading velocity of five time units and the average synchronization distance of seven time units. Completion times shown in the graphs are averages computed over 20 iterations for each scenario and the rollback penalties are based on the average reading time (denoted as T in the graphs) and the amount of pages rolled back (denoted as N in the graphs).

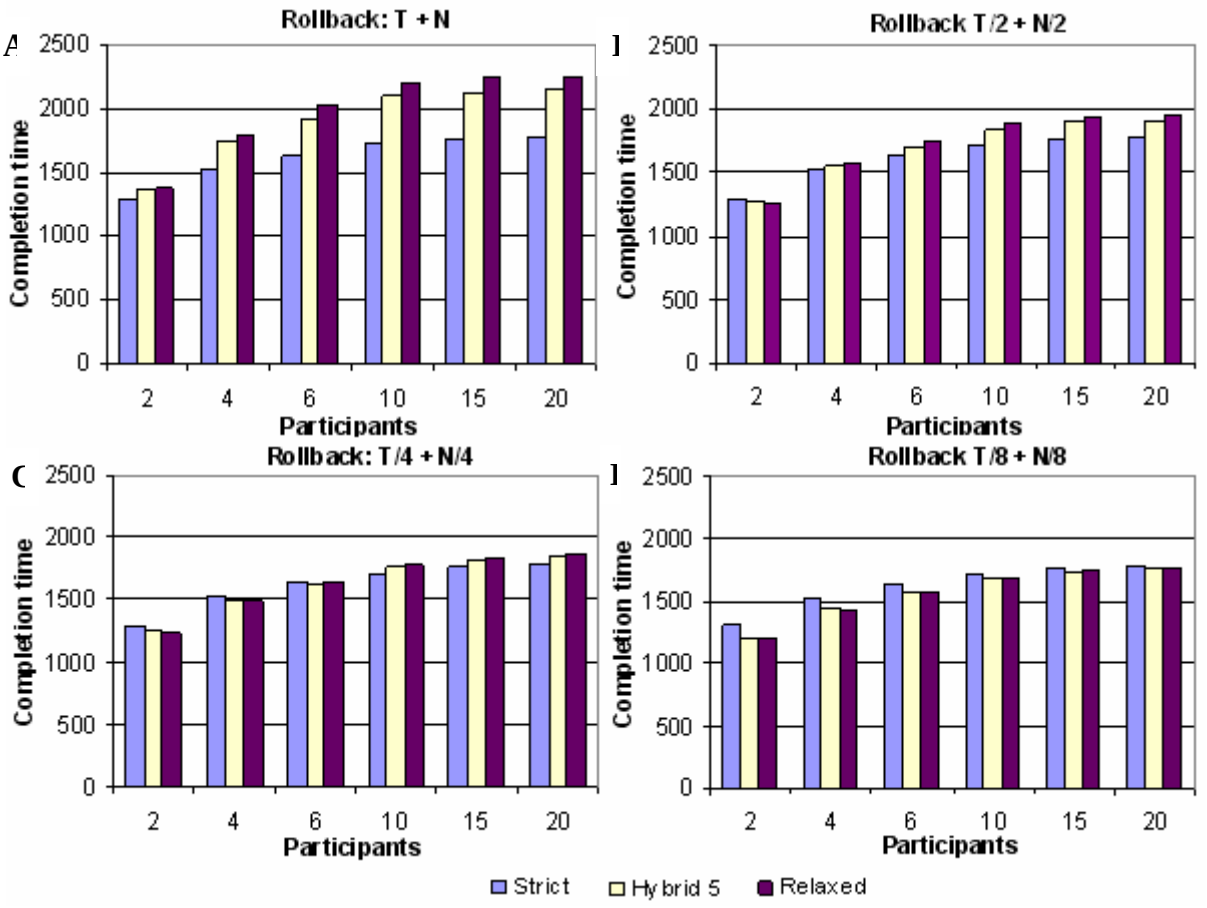

Figure 4.2

Completion times for various policies with different amount of participants and rollback penalties.

In graph A in figure 4.2, where the rollback penalty is large, it is clear that it is an advantage to use the strict policy since there seems to be a correlation between the number of participants and the number of synchronizations which are directly related to the amount of rollbacks. Graph B shows the same tendencies as graph A although the different policies are fairly equal when there are few participants. In graph C and D all policies perform roughly equally well although there is a slight advantage to use a relaxed policy when there are few participants.

4.2 Heterogeneous participants

In the previous experiments all participants had the same behavior in terms of reading velocity and interaction need. Since this is not very likely the case in a real CSCW system this chapter will experiment with scenarios where participants have a heterogeneous behavior to see which policies are best suited.

4.2.1 Homogeneous group and a dynamic participant

Experiments in this chapter will have a group of three participants with homogeneous behavior and one single participant behaving in an opposite manner (in terms of interaction need and read velocity) and then gradually approximating the groups. The rollback penalty in experiments is T/4 + N/4 (where T: average reading time, N: number of pages rolled back). Figure 4.3 illustrates experiment 4.2.1.1 where the group resides in the fourth quadrant and the single participant starts in the first quadrant and approximates the group in five steps as shown in figure 4.3.

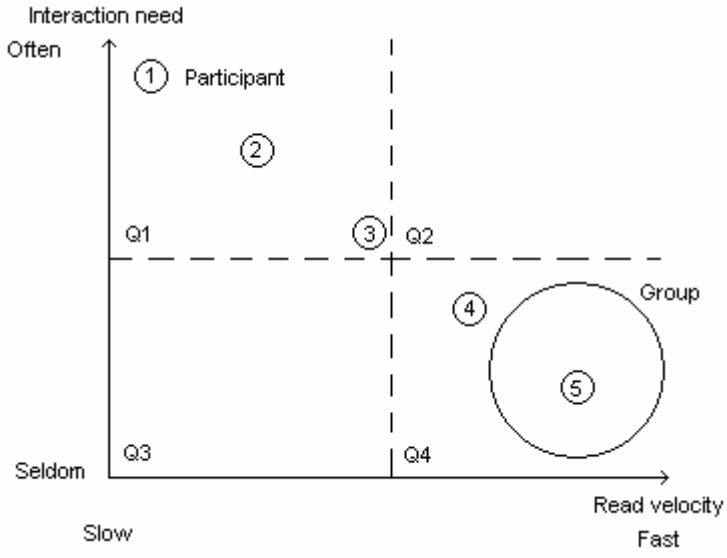

Figure 4.3

Figure 4.3 shows how a group of three participants have the same behavior (they read fast and synchronize seldom) and how one participant that in the first scenario behaves in an opposite manner approximates the group's behavior in five steps.

This chapter contains four experiments, the first is depicted above and has the configuration [Q1, Q4] (i.e. [<start quadrant of single participant>, <quadrant of group>]). The other three experiments have the configurations [Q2, Q3], [Q4, Q1] and [Q3, Q2] and have the single participant approximating the group in the same way as the first experiment. The read velocity and interaction need for the quadrants are the following: Q1: RV: 6-8, IN: 1-3; Q2: RV: 1-3, IN: 1-3; Q3: RV: 6-8, IN: 6-8 and Q4: RV: 1-3, IN: 6-8. The values of the start configuration for the single participants are equivalent with the groups' and the configurations of the following steps are specified below the x-axis in the graphs.

4.2.1.1 Group reads fast and synchronizes seldom, single participant with varied behavior

In this experiment we have three participants who read fast with an average reading velocity of two time units per page (1-3) and that seldom synchronize, having an average gap of seven pages (6-8) between synchronizations. The fourth participant (starting in the upper left corner in figure 4.3) gradually changes its behavior from being a slow reader that often synchronizes to the behavior of the other three participants. The participants read a document of 200 pages with the rollback configuration (T/4 + N/4). In the graph below we see the completion times of five experiments for different synchronization policies where the reading velocity and interaction need of one participant has been varied as stated below the x-axis (where rv: read velocity, in: interaction need).

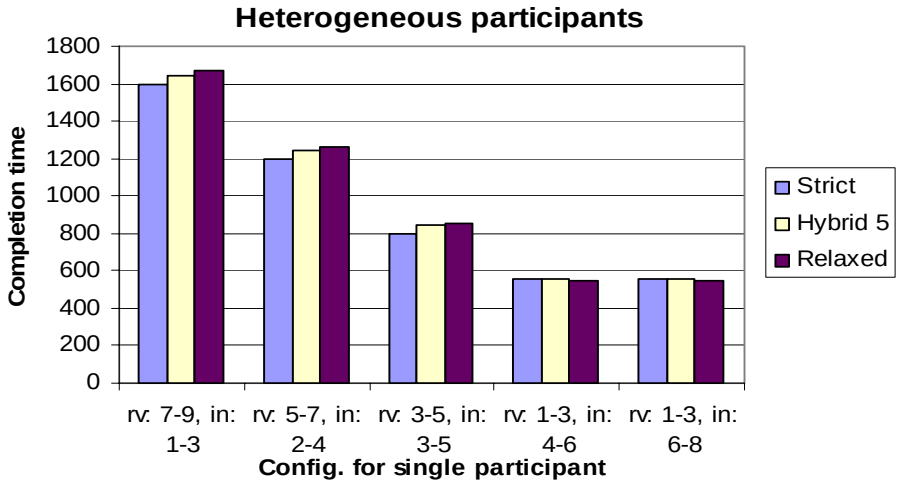

Figure 4.4

Figure 4.4 shows the completion times for different configurations of a single participant as depicted in figure 4.3.

We can observe from the graph that when the differences between the single participant and the homogeneous group are large it is beneficial to use a strict policy while it is slightly better to use a relaxed policy when they all behave alike. The reason behind this could be that when one of the participants needs to synchronize more it often leads to more rollbacks which increase the completion time due to the rollback penalty.

When the behavior is homogenous as shown in the rightmost part of the graph all policies seem to perform equally well. If we compare this to a similar experiment in chapter 4.1.1 with the same rollback penalty there is a pretty big difference since the policies there do not perform equally well. This could be due to the difference in the reading velocity which is faster in this experiment and the low reading time spread (1-3) which benefits the strict policy.

4.2.1.2 Group reads slow and synchronizes seldom, single participant with varied behavior

In this experiment the group reads slowly and synchronizes seldom. It seems as if the more the single participant (starting in the upper right corner in figure 4.3) approaches the group's behavior the worse the relaxed and hybrid policies performed, as can be seen in figure 4.5. The reason for these policies performing worse than the strict could be due to the fact that the participants reading times have a small spread as the single participant's behavior approaches the groups.

For example, in the rightmost part of the graph below all the participants have a reading time of 6-8 time units. This means that when the strict policy is applied, the maximum time a participant has to wait for synchronization is two time units. When the hybrid or relaxed policy is used the rollback penalty is around two time units and there isn't the same guaranteed advancement of the strict policy and often participants have to read parts of the same page several times. On the other hand, the interaction needs are low which benefits the hybrid and relaxed policy since it generates fewer rollbacks and allows those policies to perform better. But clearly in this case the low spread of reading times outweighs the small amount of rollbacks with rather low rollback penalty and the strict policy is benefited the more homogeneous the behavior becomes.

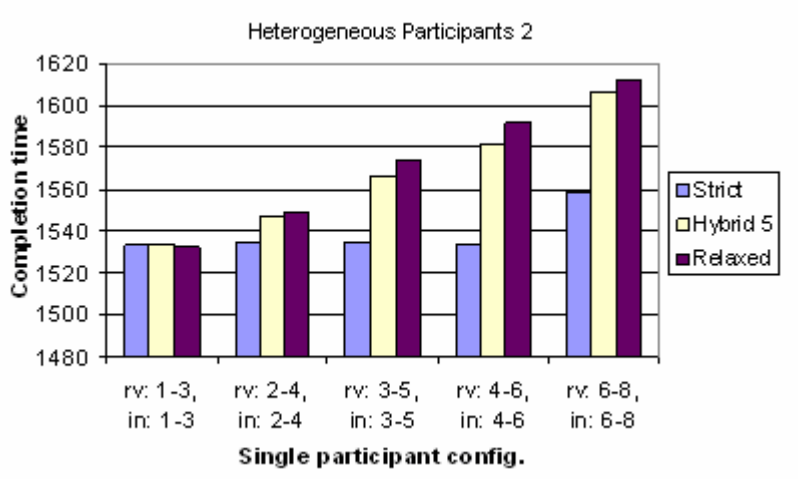

Figure 4.5

Figure 4.5 shows the completion times for different configurations of a single participant which approximates its behavior towards the group's as shown below the x-axis.

4.2.1.3 Group reads slow and synchronizes often, single participant with varied behavior

In this scenario the group consisting of three participants reads slowly and synchronizes often and the single participant (starting in the lower right corner in figure 4.3) reads fast and synchronizes seldom. This generates a lot of rollbacks that increase the completion times of the relaxed and hybrid policies which can clearly be seen in figure 4.6. One odd thing is that the differences between the policies' performances seem almost independent of the single participant's behavior; this indicates that the group's behavior is so dominant that the single participant hardly can influence the outcome.

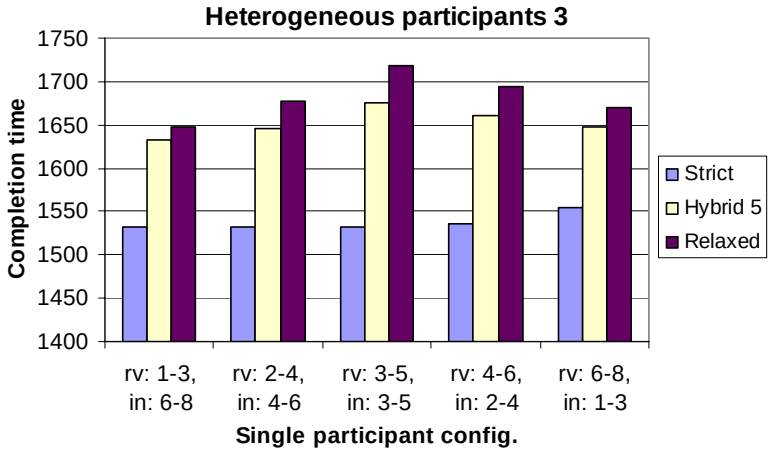

Figure 4.6

Figure 4.6 shows the completion times for different configurations of a single participant which approximates its behavior towards the group's as shown below the x-axis.

4.2.1.4 Group reads fast and synchronizes often, single participant with varied behavior

In figure 4.7 we can see how the synchronization policies behave when a group of participants read fast and synchronize often with one participant (starting in the lower left corner in figure 4.3) behaving in an opposite manner and then gradually approximating to the behavior of the group. The first thing we can observe is that all the policies perform nearly equally well although the strict policy performs marginally better. This indicates that the group's behavior is dominant in this scenario and that the behavior of the single participant does not affect the relation between the policies' performances. In this experiment many rollbacks were generated since most participants synchronized often. The reason that this does not affect the relaxed and hybrid policy much is the generally low reading time which makes the rollback of a page a small loss in time. The low spread of reading times in general (1-3) benefits the strict policy which can be seen in figure 4.7.

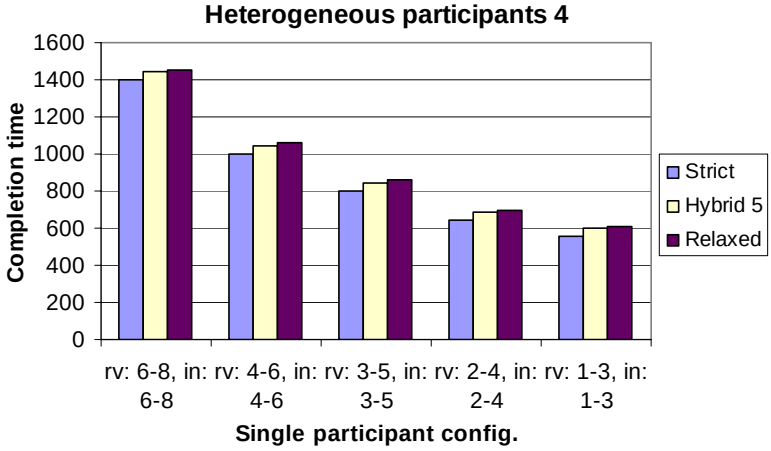

Figure 4.7

Figure 4.7 shows the completion times for different configurations of a single participant which approximates its behavior towards the group's as shown below the x-axis.

4.2.2 Static heterogeneous behavior

In this subsection the participants have the same configuration throughout the whole experiment; the varied parameters are the mean synchronization distance and the rollback penalty. In both experiments three participants with different attributes are tested in different scenarios both with a high rollback penalty and without rollback penalty. In the first configuration the participants have the following reading times (1-3, 2-4 and 3-5) and in the second (1-7, 1-13 and 1-19). The experiment attempts to show how the policies perform when participants have different reading times, which should be the case in most real life situations. The purpose of having two different sets of reading times is to see how the narrow and wide reading time ranges influence the performance.

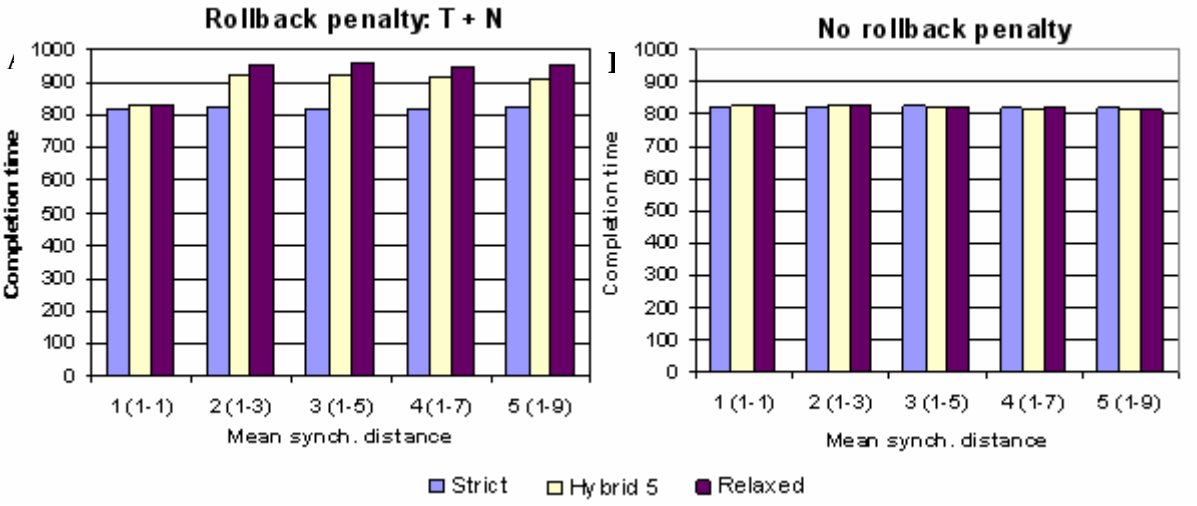

Figure 4.8

Figure 4.8 shows two experiments with different rollback penalties and with a low spread in reading times for all participants.

In figure 4.8 we can clearly see that the hybrid and relaxed policies perform poorly compared to the strict when the rollback penalty is high and only marginally better when there is no rollback penalty. This indicates that the strict policy generally works better for heterogeneous behavior, especially so when the reading time spread is narrow which can be seen if one compares with the two graphs below that have a broader spread.

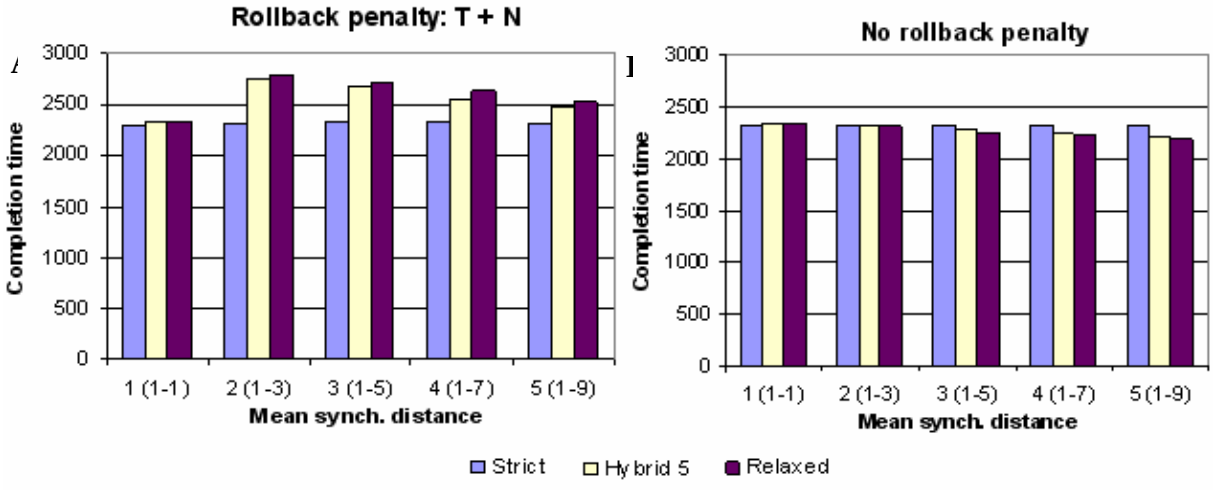

Figure 4.9

Figure 4.9 shows two experiments with different rollback penalties and with a high spread in reading times for all participants.

In figure 4.9 the second set of reading time was used and it can be seen that the higher spread of reading times marginally benefits the hybrid and relaxed policies. Probable reasons for this could be that fewer rollbacks are generated in total and that the higher spread in reading times gives the strict policy a higher average because it always has to adapt to the slowest reader.

4.2.3 Static heterogeneous behavior 2

This experiment is similar to the previous but with a configuration where the slowest reader synchronizes the most frequent and vice versa. In the first configuration the readers are fast with reading times (3-5, 2-4 and 1-3) and with mean synchronization distances (4, 3 and 2). In the second configuration the reading times are (1-19, 1-13 and 1-7) with the same synchronization distances.

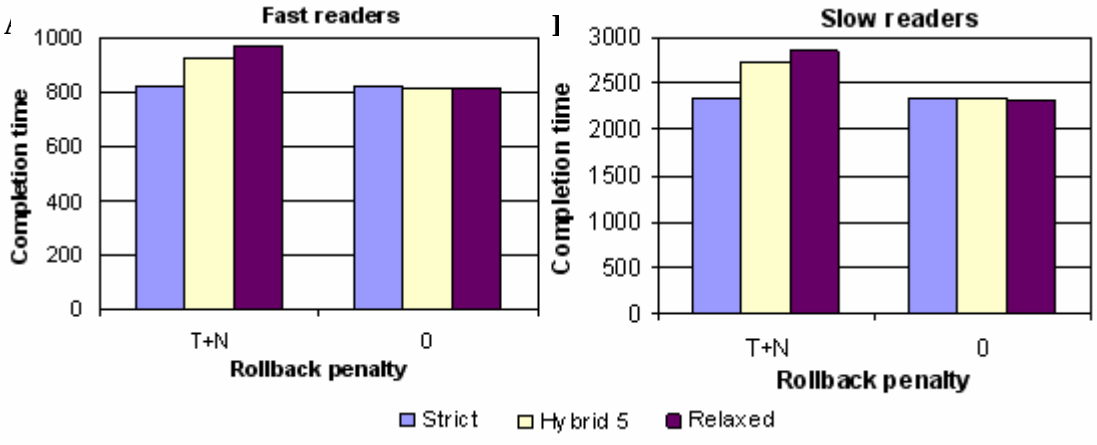

Figure 4.10

Figure 4.10 shows two experiments with different rollback penalties and fast and slow readers.

When comparing the graphs A and B in figure 4.10 the relations between the different policies seem nearly identical this indicates that a difference in reading time spreads hardly makes a difference under these circumstances. Furthermore, it seems that the hybrid and relaxed policies perform about the same as the strict without the rollback penalty which shows that the strict policy generally is better for heterogeneous participants.

5 Conclusions and future work

5.1 Conclusions

Choosing the optimal synchronization policy for a given scenario is a complicated process since there are many factors that influence how well a synchronization policy performs. Through a variety of simulated scenarios presented in chapter 4 the author has found that the following parameters have an impact on the efficiency of the tested policies:

- **Number of participants** – as we can see in subsection 4.1.2 having many participants benefits from using a strict policy which is comprehensible since having many participants generates rollbacks on almost every page with the relaxed and hybrid policy. Furthermore, it is easier to predict the synchronization behavior with the strict policy when there are many participants; the average reading time for a page simply approximates the upper bound of general reading times whereas the relaxed and hybrid policies potentially could have a rollback penalty on every page.

- **Reading velocity in general –** this parameter obviously has a great impact on the completion times of the documents in general but it does not seem to have an important influence on the relation between the different synchronization policies. One important detail is that fast readers automatically have narrow range in reading velocity which can have an impact on the performances of the policies as described below.

- **Interaction need in general –** in experiments where the interaction need is studied it can be observed that frequent synchronization makes the relaxed and hybrid policies perform worse than the strict. Frequent synchronization generally generates more rollbacks since there simply are more occasions where rollbacks can occur. A high interaction need in general does not allow the hybrid and relaxed policies to perform at their full potential since the rollback penalties outweigh the benefits of having relaxed and hybrid policies.

- **Rollback penalty –** the penalty that is issued at each rollback to model the time it takes a normal system to find and load a previous state. It can clearly be observed in most experiments that this parameter has a great impact on the performance of the hybrid and relaxed policies.

- **Heterogeneousity –** the difference among the participants in terms of reading times and interaction needs. In most of the experiments with heterogeneous participants the relaxed and hybrid policy performed worse than the strict policy and in the cases where there was no rollback penalty the performance was marginally better than the strict at best. On the other hand in section 4.1 where the participants are homogeneous we can clearly see that there are scenarios where the relaxed and hybrid policy perform better than the strict.

- **Narrow ranges in reading time –** in subsection 4.2.2 and 4.2.3 there are two scenarios where the participants either have wide or narrow ranges in reading velocity. These experiments show that there is a benefit, although marginal, for the relaxed and hybrid policies when the range is wide. When the reading time range is narrow the strict policy is benefited since the average waiting time for each page will be low and also the hybrid and relaxed policies have little to gain under these circumstances but have an additional risk of rollback penalties.

Observing the influences of the parameters above, it can be seen that strict policy works better than the other policies when there are many heterogeneous participants that have a high interaction need with a high rollback penalty and with a narrow ranged reading time. Based on the experiments in chapter 4 the relaxed and hybrid policies tend to work best when there are few participants that have heterogeneous behavior and when the rollback penalties are low.

Since the behavior of the participants in an arbitrary real-life scenario is unlikely to be homogeneous it is probable that the strict policy suits most of the scenarios in CSCW systems best. Furthermore, using a strict policy is usually better for human interaction since we can not rollback something that we have perceived, as a machine can do. Instead we have to try to disregard that particular change and thus the system becomes more difficult and unintuitive to use. Even though the strict policy seems to be the better one in most scenarios there are some situations where a relaxed or hybrid policy could shorten the completion times, especially in a system that has low rollback costs and where the participants have a somewhat similar behavior. Therefore, most CSCW systems could still benefit from implementing all the tested synchronization policies and have a dynamic mechanism that changes the policy based on the behavior of the participants.

5.2 Future work

The main objective of this master thesis project was to implement a simulator platform that could be used to analyze and evaluate synchronization policies in different scenarios. The project also aimed at experimenting with various scenarios in order to create guidelines describing the dynamics of the tested policies. These guidelines could potentially assist the possible development of a mechanism that could choose synchronization policy in real-time in a CSCW system. With this in mind, the most interesting future work would be to develop this mechanism with the help of the simulator platform and the guidelines in an attempt to improve today's CSCW systems.

To improve the work done in the thesis the simulator platform could be further developed. For instance, additional synchronization policies could be implemented and more parameters could be measured during the simulation to improve the quality of the output. Furthermore, experimenting could also be done to gain more understanding the dynamics of the different policies in order to make a policy chooser mechanism more efficient.

References

[1] Ayani, R. and Ulriksson, J., ***Consistency Overhead using HLA for Collaborative Work***, Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2005, Montreal, Canada.

[2] Dourish, P., ***Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaboration Toolkit***, Proceedings of the 1996 ACM conference on Computer supported cooperative work, Boston, United States.

[3] Gautier, L., Diot, C. (1997): MiMaze, a Multiuser Game on the Internet. INRIA Sophia Antipolis unit - RODEO project - Research report 3248 - September 1997. http://www.inria.fr/rodeo/MiMaze.

[4] Churchill, E.F., Snowdon, D.N. and Munro, A.J., ***Collaborative Virtual Environments: Digital Places for Interaction***, ISBN 1-85233-244-1, Springer-Verlag London 2001.

[5] Olson, J., ***Distributed consistency management in Sombrero, a single address space distributed address system***, technical report, 2002, http://www.eas.asu.edu/~sasos/docs/j_olson_thesis.pdf, 2006-02-09.

[6] Susarla, S., ***Flexible consistency management in a wide-area caching data store***, technical report, http://www.cs.utah.edu/~sai/papers/proposal, 2006-02-09.

[7] Greenhalgh, C. and Vaghi, I., ***Demanding the Impossible: Data Consistency in Collaborative Virtual Environments***, technical report version 1.2, 2002, Department of Computer Science, university of Nottingham UK.

[8] Roberts, D. and Wolff, R., ***Controlling Consistency within Collaborative Virtual Environments***, Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2004, Budapest, Hungary.

[9] Ayani, R., lecture slides in the course Distributed systems at The Royal Institute of Technology, Sweden. http://www.imit.kth.se/courses/2G1509/, 2006-02-09.

[10] Picco, G., P., lecture slides in the course Distributed computing systems given at Politecnico de Milano, Italy. http://www.elet.polimi.it/upload/picco/Teaching/UIC553/slides/synch1.pdf, 2006-02-09.

[11] Fujimoto, R. M., ***Parallel and Distributed Simulation Systems***, ISBN 0-471-18383-0, John Wiley & Sons, 2000.

[12] Grudin, J., ***CSCW: History and Focus***, technical report, Information and Computer Science Department, University of California, http://www.ics.uci.edu/~grudin/Papers/IEEE94/IEEEComplastsub.html, 2006-02-09.

[13] Greenhalgh, C., and S. Benford (1997) "A Multicast Network Architecture for Large Scale Collaborative Virtual Environments", in Multimedia Applications, Services and Techniques - ECMAST'97, Proceedings Second European Conference, Serge Fdida and Michele Morganti (eds.), Milan, Italy, May 21-23, 1997, pp. 113-128, Springer.

[14] Roberts, D., ***Principles of Collaborative Virtual Environments***, technical report, University of Salford Manchester, http://info.nicve.salford.ac.uk/web/files/uplink/Roberts_IIS2003.pdf, 2006-02-16,

[15] Pham, C., D., lecture Slides about parallel simulations on high-performance clusters, University of Lyon, France, http://www.univ-pau.fr/~cpham/Paper/TalkLINZ.ppt, 2006-02-19.

[16] The ACM CSCW conference web site, http://www.acm.org/cscw2004/, 2006-02-20.

[17] Ramage, S., slides in the course Computer Supported Co-operative Working given at Lincoln University, UK, http://hemswell.lincoln.ac.uk/~cboldyreff/CSCW/new-cscwlect.html, 2006-02-24.

[18] Carlsson, C., and O. Hagsand (1993) "DIVE - A Multi-User Virtual Reality System", in Proc. VRAIS'93, IEEE Virtual Reality Annual International Symposium, pp. 394-400.

Appendix A – Installation

- Make sure the Java 2 SDK is installed.
- Download the source code and the class files from:

  http://web.it.kth.se/~it00_mma/classes_and_source.zip
- Unzip the content and go to: “<unzip-directory>/build/”
- Type “java gui” in the command prompt to execute the GUI.