Architecture for multi-party synchronization of data sets in a distributed environment

Marcin Marczewski

A dissertation submitted to the University of Dublin,

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

September 12, 2005

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Marcin Marczewski

September 12, 2005

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Marcin Marczewski

September 12, 2005

Acknowledgements

I would like to thank my supervisor Mr. Alexis Donnelly for his guidance and suggestions during the course of this project.

I would also like to thank my parents and family for their support throughout this demanding year.

Special thanks to Katarzyna for her continuous faith in me and frequent visits in Ireland.

Finally, I would like to thank all my NDS classmates for their friendship and making this year more enjoyable.

Abstract

In the world of common mobility, where more and more people and organizations use various mobile equipment such as phones, personal data assistants and other devices the consistency of stored and exchanged data is a very important thing. Today the flow of information, especially in business processes, can be very high and valuable, and deformation of this data is unacceptable because it can lead to numerous misunderstandings.

The use of mobile devices and networked computers makes people want to have the same data on all these devices. In case where a large number of devices stores huge data sets which can be modified by hundreds of people, simple data copying is highly unprofitable and time-consuming.

Nowadays, we can find a lot of various algorithms for set reconciliation designed for various environments, devices. These algorithms have a variety of advantages, but they also have a number of disadvantages which can limit their usefulness.

In this work, a few existing data synchronization algorithms are investigated and compared. Their usefulness for various scenarios is described and analyzed. What is more, this dissertation project also presents various gossip protocols which can be used to disseminate messages in a network environment.

Finally, an architecture for multi-party synchronization of data sets in a network environment is designed, implemented and examined by number of experiments with various settings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the world of common mobility, where more and more people and organizations use various mobile equipment such as phones, personal data assistants and other devices the consistency of stored and exchanged data is a very important thing. Today the flow of information, especially in business processes, can be very high and valuable, and deformation of this data is unacceptable because it can lead to numerous misunderstandings.

The use of mobile devices and networked computers makes people want to have the same data on all these devices. In case where a large number of devices stores huge data sets which can be modified by hundreds of people, simple data copying is highly unprofitable and time-consuming. Instead, some synchronization protocol should be used.

Nowadays we can find a lot of various algorithms for set reconciliation designed for various environments and devices. These algorithms have a variety of advantages, but they also have number of disadvantages, which can limit their usefulness.

The increasing number of used mobile devices caused many researchers to work on this approach. Most of these methods (for example The Palm HotSync [1]) make it possible to synchronize data between two devices, but they are not so efficient,

because in many cases we have to transmit not only modified records but the whole data. What is more, most of existing protocols are not dependent on the number of differences between synchronizing data sets, but depend on the size of the data.

This dissertation project will investigate various methods of data sets reconciliation and will present a set of gossip protocols which can be used to disseminate messages in a network environment. Basing on these analyzes, an architecture for networked synchronization will be designed, implemented and examined.

1.2 Objectives

The following objectives are going to be achieved along with the completion of this dissertation project:

- Review and analysis of chosen existing data synchronization methods. Comparison of these methods and their complexity, efficiency, usability and suitability for network environments,
- Investigation of advantages and disadvantages of various types of distributed systems. Comparison of their similarities and differences in connection with efficiency, scalability and network usability,
- Analysis and comparison of various gossip protocols. Investigation of their complexity and usability for different network structures,
- Design of the multi-party data sets synchronization system based on an efficient algorithm dependent on the size of the number of differences between synchronizing data sets. Detailed architecture and generalization for various environments and devices,
- Implementation of the proposed system,
- Number of experiments with the implemented tool. Tests performed in various

environments with different number of devices. A legible presentation of experiments' results and their interpretation,

- Consideration on other possible scenarios,
- Determining goals for the future work.

1.3 Roadmap

This dissertation report consists of the following parts:

Chapter 2: State of the art. This chapter provides general information how handheld computers synchronize their data today. Various synchronization algorithms are analyzed and their advantages and disadvantages are presented. This part also includes a brief overview of distributed systems with focuses on various types of peer-to-peer systems. A few case studies are described,

Chapter 3: Gossiping - the way of message dissemination. This chapter includes a presentation and analysis of various gossip schemes used for message dissemination in various network topologies,

Chapter 4: Design. This chapter gives an overview of the design of the proposed data synchronization system. The key decisions made during designing process are presented and final algorithms are provided,

Chapter 5: Implementation. This part presents details of implementation of the proposed system design in Chapter 4. Main components and functions are described in detail,

Chapter 6: Simulation results. This chapter discusses the results of various experiments which were carried out in different environments with different data sets and a number of machines,

Chapter 7: Conclusions and evaluation. In this chapter conclusions and directions to the future work are presented.

Chapter 2

State of the art

The first part of this chapter describes the current use of synchronization protocols and limitations of these methods. The short analysis and comparison of chosen protocols is provided.

Secondly, two main types of distributed systems are presented and compared. The main focus is on peer-to-peer approach - the most popular systems are analyzed as case studies.

2.1 Handheld computers' synchronization today

2.1.1 Definition of data synchronization

In this dissertation project a data synchronization process is considered as exchanging information between two or more devices. These devices can store different number of information and some of these messages can be common for chosen group of hosts. As a result of data synchronization process all participating nodes store exactly the same set of information.

More formalized definition is presented in Table 2.1.

Definition of a synchronization process
Given two machines with largely overlapping data sets: respectively **A** and **B**. **Synchronization** is a process of data exchanging which ensures that both machines finally have copies of $\mathbf{A} \cup \mathbf{B}$. Ideally, only the symmetric differences ($\mathbf{A} \backslash \mathbf{B}$ and also $\mathbf{B} \backslash \mathbf{A}$) should be exchanged to minimize network traffic.

Table 2.1: Definition of a synchronization process

2.1.2 Data synchronization today

Nowadays we can find a lot of various algorithms for set reconciliation designed for various environments and devices. These algorithms have a variety of advantages, but they also have a number of disadvantages such as complexity or data-size dependency, what can limit these protocols' usefulness.

The increasing number of mobile devices caused many researchers to work on this approach. Most of these methods (for example The Palm HotSync [1]) make it possible to synchronize data between two devices, but they are not so efficient, because in many cases we have to transmit not only modified records but the whole data. What is more, most existing protocols' performance depends not on the number of differences between synchronizing data sets, but on the size of the data.

Another problem in data synchronization process appears when we want to synchronize our information between various types of devices - for example between our personal computer, our personal data assistant and mobile phone. A need of some kind of standardization induced the biggest companies on this market (Ericsson, IBM, Lotus, Motorola, Nokia, Palm Inc., Matsushita Communications Industrial, Psion and Starfish Software) to find a common solution which could be used as a standard by various types of devices. The effect of this work is a SyncML initiative [4]. Today this protocol is used by number of mobile phones and PDA devices but is still under development. This solution uses timestamps which are used to control which records were modified after the last synchronization. Thanks to it this protocol can work in a network environment, but it requires every machine to store information about other

devices in this network. This has a very negative impact on the scalability of this protocol and makes it impossible to use this algorithm in peer-to-peer infrastructure where new hosts can join or leave the network intermittently.

2.2 Synchronization algorithms

2.2.1 Palm synchronization protocol

Palm HotSync [1, 9, 10] is a first method of set reconciliation used to synchronize data between palmtop and desktop computer. The algorithm is very simple and works on data which is stored as a data base - each information as a new record. These records have an additional field which is used as a flag. There are three types of flag: *modified*, *inserted*, and *deleted* which are set up according to the changes introduced to the data.

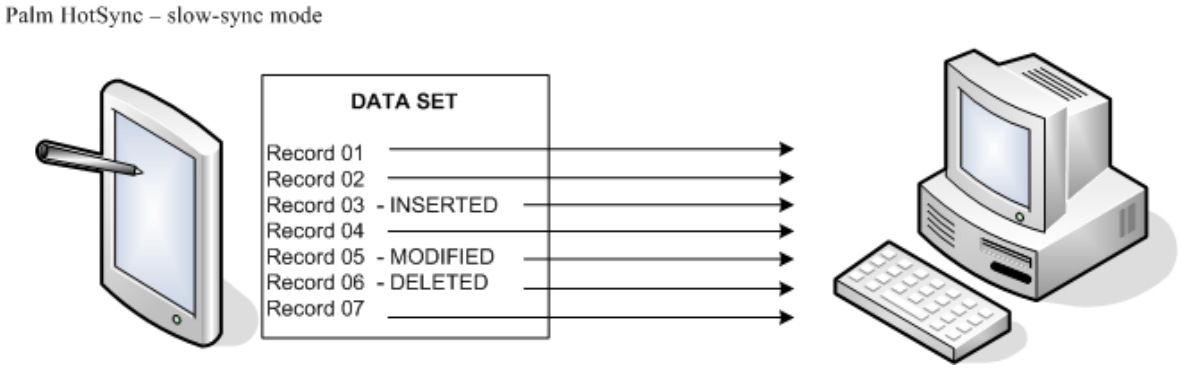

Figure 2.1: Palm HotSync: slow-sync mode

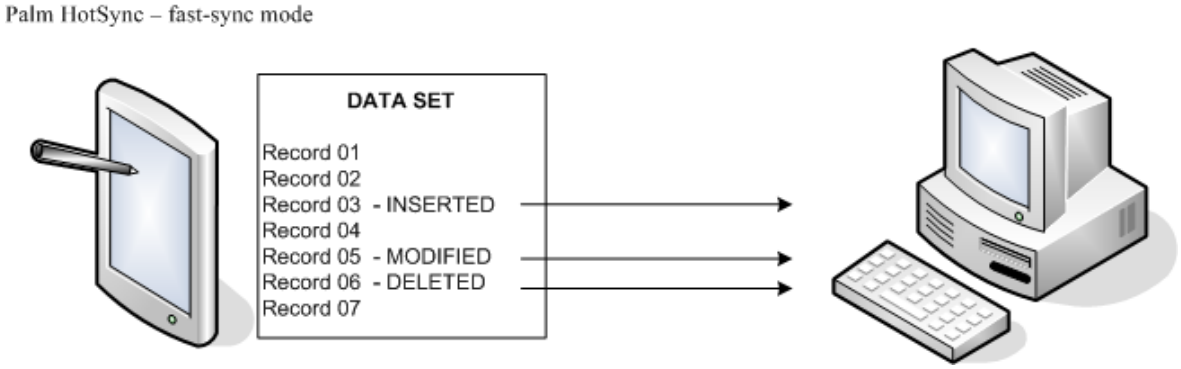

Figure 2.2: Palm HotSync: fast-sync mode

The HotSync algorithm works in two modes: *slow-sync* and *fast-sync*. The first one is executed when a palmtop synchronizes with a new computer - all the records are transmitted to the PC (Figure 2.1). The second solution is used when palmtop wants to synchronize with the same machine it did last time. Then only modified records are sent to the desktop machine. After sending the data all flags are cleared. See Figure 2.2.

This solution is pretty simple and easy to implement, but has a lot of disadvantages. First of all, it is highly inefficient when we want to synchronize our data with many computers - then always *slow-sync* mode has to be performed and the whole data has to be copied many times.

Advantages	**Disadvantages**
- easy to implement	- not efficient if synchronization is not performed with one (the same) machine - not scalable - depends on the size of data - one-way synchronization

Table 2.2: Palm HotSync: Advantages and disadvantages

2.2.2 Timestamps

Another solution for data synchronization is using timestamps. Similarly to Palm HotSync this method requires maintaining additional field with metadata with each record in the data base. Time of the last data synchronization or modification is written there. When data records are modified, deleted or inserted current time in this metadata field is set. In case of next synchronization process, only records with higher time than than the time of last synchronization are transmitted. This method can work in two directions. What is more, when the same records were modified on both handheld device and desktop computer, after synchronization process both machines have the newest version.

Although time stamping method can be used in the synchronization between high number of devices in some specific situations it can provide inefficient effects. Figure 2.3

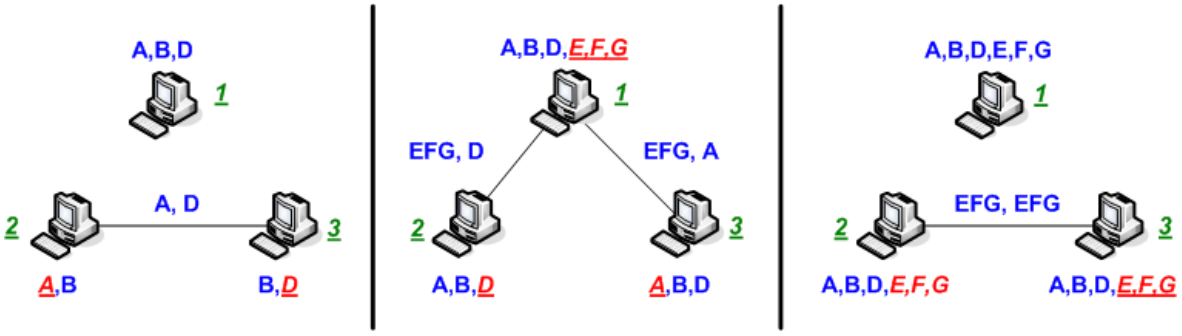

Figure 2.3: Inefficiency of the synchronization using timestamps - during the third step of synchronization process machines 2 and 3 exchange exactly the same data sets

shows an example of inefficiency of use time stamping. Assuming that initially devices 1,2,3 store set of records, respectively: A,B,D, B and D in the effect of some synchronization process performed earlier, when record A is added on machine 2 and records D is inserted on computer 3 these two, newly added information have timestamp higher than the time of the last synchronization performed between nodes 2 and 3. So, in case of synchronization between these two machines computer 2 receives record D and device 3 receives information A - see Figure 2.3A. After this, when new records (E,F,G) are added to the machine 1 (see Figure 2.3B) in case of synchronization process between machines 1 and 2 device 1 receives record D and computer 2 receives set of records E,F,G. Similarly, when machine 1 synchronizes with device 3 computer 1 receives information A and computer 3 receives set of information E,F,G. Here an inefficiency of the synchronization process is noticed, because machine 1 receives records D (from device 2) and A from computer 3 although it already stores these both information. Finally, when machines 2 and 3 synchronize again (Figure 2.3C) both send to each other set of information E,F,G, because these records have timestamps higher than the time of the last synchronization between these two computer. The inefficiency is obvious, because both of these devices have already stored received information before the set reconciliation process.

Advantages	Disadvantages
- easy to implement - two way synchronization	- scalable, but can be confusing in specific situations - not efficient if synchronization is not performed with one (the same) machine

Table 2.3: Timestamps: Advantages and disadvantages

2.2.3 Characteristic Polynomial Interpolation-Based Synchronization

CPIsync (Characteristic Polynomial Interpolation-Based Synchronization) a is a method of data sets synchronization proposed by Starobinski, Trachtenberg and Argawal [1, 2, 3]. In comparison to many other synchronization algorithms CPIsync is linearly depended on the number of differences in data sets stored by various hosts. What is more, CPIsync does not require to store any information about previous synchronization or about other devices (like in Palm HotSync protcol), so it can operate in dynamic networks where the number of participating nodes is changing intermittently.

In a CPIsync protocol data sets are stored in a form of a data base - each information in one, separate record. Each record is represented by one, unique integer.

The main assumption of this method is this that all information stored can be presented as a **characteristic polynomial** representing all the integers:

$$\chi_S(Z) = (Z - x_1)(Z - x_2)(Z - x_3)...(Z - x_n)$$

where x is one information record (integer)

Starobinski, Trachtenberg and Argawal in their research present an observation that for two data sets S_A and S_B following equation can be drawn:

$$\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)} = \frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}$$

where Δ_A and Δ_B represent differences sets respectively $S_A - S_B$ and $S_B - S_A$.

In the equation presented above the common elements for both data sets stored by both synchronizing devices are cancelled out, so the remaining elements are exactly the differences between original data sets. To find these elements another mechanism is used.

As presented in [2] the general CPIsync algorithms works as following:

1. Hosts A and B evaluate their characteristic polynomials on $\bar{m}$ sample points (over a chosen finite field). Host A sends its evaluations to host B.

2. The evaluations are combined to compute $\bar{m}$ sample points of the rational function $\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)}$, which are interpolated to determine $\frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}$

3. The numerator and denominator of the interpolated function are factored to determine the differences between S_A and S_B. The zeroes of $\chi_{\Delta_A}(Z)$ and $\chi_{\Delta_B}(Z)$ are the elements of Δ_A and Δ_B.

Thanks to these assumptions the presented algorithm can be very effective and can limit the data transferred between synchronizing hosts. Although the degree of $\chi_{S_A}(Z)$ and $\chi_{S_B}(Z)$ can be highly depending of the size of the data set the degree of the $\chi_{\Delta_A}(Z)$ and $\chi_{\Delta_B}(Z)$ can be small (depending on the number of differences between both hosts).

This presented version of CPIsync assumes that the number of differences between synchronizing hosts - $\bar{m}$ is known a priori, but Starobinski, Trachtenberg and Argawal also introduce a probabilistic scheme for the situation where $\bar{m}$ is not known. It is much more complex consideration and requires additional communication between synchronizing nodes to determine appropriate upper bound, but the disadvantage of this scheme is that the higher the number of differences between original data sets is the lower is the accuracy of determining these differences.

Characteristic Polynomial Interpolation-Based Synchronization algorithm is a very fast method of set reconciliation, especially when the data sets are very large and the number of differences between synchronizing devices is relatively small. But for

the scenario where the same nodes exchange their data sets and/or the number of differences is high and where the number of messages sent (not their size) is important implementation of CPIsync can be ineffective and not accurate.

Table 2.4 summarizes the advantages and disadvantages of presented CPIsync algorithm.

Advantages	**Disadvantages**
- depends on the number of differences - scalable - two-way synchronization - data rows represented by integers - very fast when the number of differences is small	- complex - difficult to implement - cost of determining an upper bound can be inefficient in case when data sets are really small or the number of differences is high - when the number of differences is high in the relation to the whole data set determining the number of differences using the probabilistic scheme can be inaccurate

Table 2.4: CPIsync: Advantages and disadvantages

2.2.4 Other methods

During the course of this project a lot of research was done and various methods were analyzed. There also exist other data synchronization protocol and related algorithms which could be used for finding modified or new records in set of information records, but because of space and time limitation and also little interest in them they are not described in this dissertation report. These are e.g. Bloom filters described in details in [13] and also in [1, 2, 3], Rsync algorithm which was a subject of Andrew Tridgell's PhD thesis [12] or nsync presented in [11] - a method for group synchronization with gossip scheme adoption.

Some of algorithms found an application in industrial projects. Two of these projects - a commercial example of synchronization platform and an initiative for creating an open synchronization protocol are described in brief.

2.3 Data synchronization industrial projects

2.3.1 SyncML

SyncML (Synchronous Mark Up Language) described in [4, 5] is an initiative started by a few biggest IT companies such Ericsson, IBM, Lotus, Motorola, Nokia, Palm Inc., Matsushita Communications Industrial, Psion, Starfish Software and other to find a common solution which could be used as a data synchronization standard protocol on various types of devices. The main assumption is that using SyncML protocol any device can synchronize with any other device over any network.

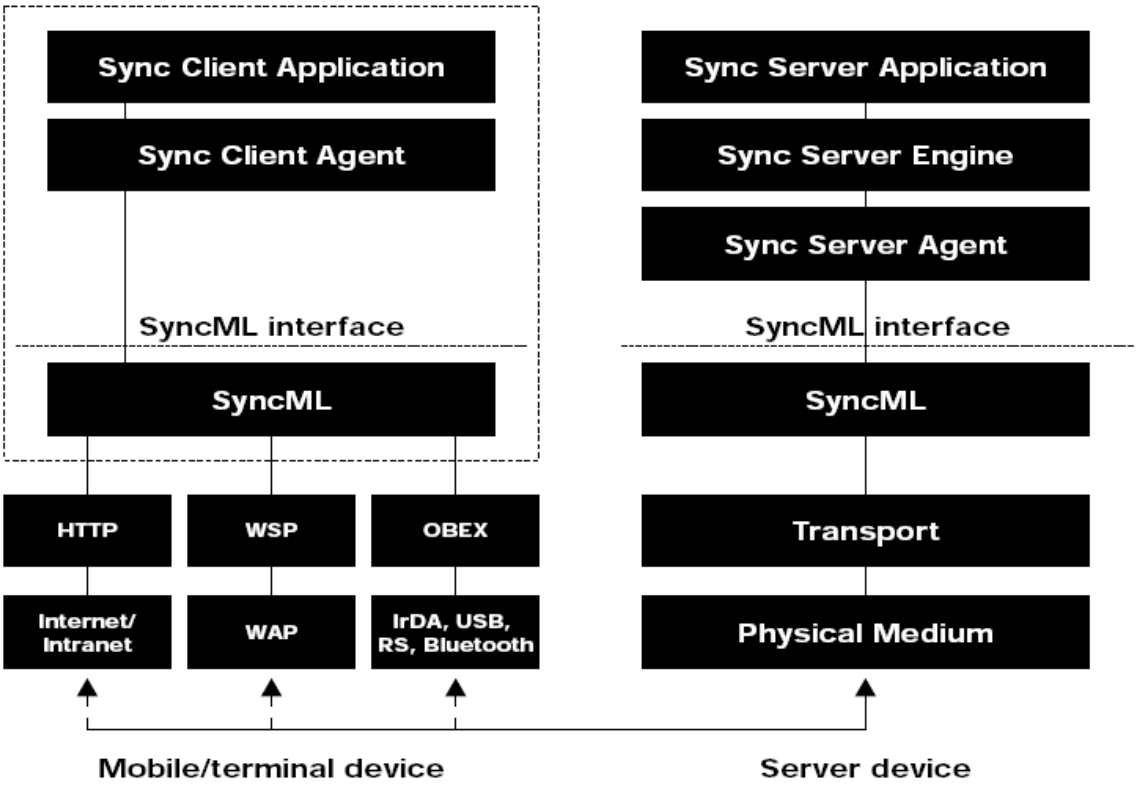

Figure 2.4: Architecture of SyncML protocol (picture from SyncML WhitePaper[4])

The effect of this work is a SyncML protocol [4]. SyncML defines the representation protocol and synchronization protocol as also a transport protocol and binding.

The representation protocol is defined by set of messages consisted of user data, commands and other meta data. All these massages are prepared using XML or WBXML (Wireless Binary XML).

Advantages	Disadvantages
- easy to implement - an open protocol developed by the biggest companies - common data and message format using XML - a few various modes of synchronization	- not efficient if synchronization is not performed with one (the same) machine - limited scalability - can depend on the size of data (depending on the chosen mode) - still under development

Table 2.5: SyncML: Advantages and disadvantages

The synchronization protocol defines the message flow between synchronizing hosts to perform set reconciliation.

Figure 2.4 from SyncML White Paper[4] presents the very high level view of the architecture of SyncML initiative.

Today this protocol is used by a number of mobile phones and PDA devices but it is still under development. This solution uses timestamps which are used to control which records have been modified since last synchronization. Thanks to it this protocol can work in a network environment, but it requires every machine to store information about other devices in this network. This has a very negative impact on the scalability of this protocol and makes it impossible to use this algorithm in a wide peer-to-peer infrastructure where new host can join or leave the network intermittently.

2.3.2 Intellisync Data Sync

Intellisync Data Sync[6] is a part of Intellisync Corporation's product called Intellisync Mobile Suite[7]. This software dedicated for large companies provides a tool for data exchanging between laptops, handheld devices, desktop PCs, local databases, file or mail servers etc. This solution provides a centralized system architecture with the Intellisync Data Sync server in the middle. Figure 2.5 shows the high level view on the architecture of this plaform.

On the one side of this Intellisync Data Sync server there are services which are responsible for storing the data - such as data base servers (IBM DB2, MS SQL, Oracle,

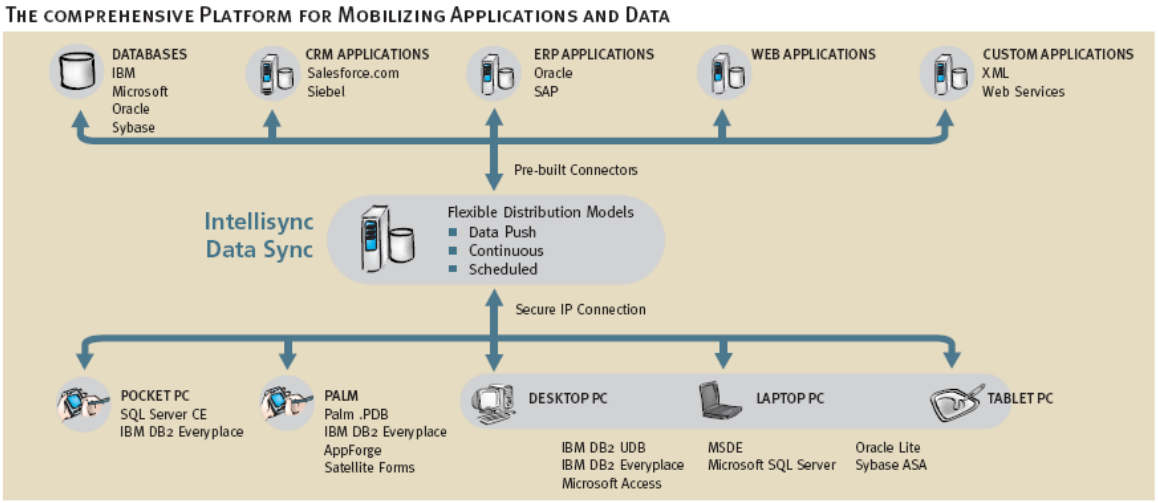

Figure 2.5: High level view on the architecture of Intellisync Data Sync (from Intellisync Technical Datasheet)

Advantages	**Disadvantages**
- easy to use - handles a lot of types of devices - deploys most of the databases used today - provides high level of security	- commercial software (cost) - centralized

Table 2.6: Intellisync: Advantages and disadvantages

MS Access and a few other), ERP/CRM applications (such as SAP) or web services.

On the other side there are all the devices which are handled by the main server. These can be desktop PCs, palmtops, laptops, mobile phones etc.

Intellisync Data Sync is an application which combines a lot of various technologies which can be used to store or retrieve information and thanks to central server it can deliver only changes or new required information to the devices which ask about this. This application can be run on most devices and operating systems used today.

The application run on main server has to maintain information about devices and their last synchronization. In this way algorithm such as fast-sync can be performed most of the time. In other words, when a totally new device connects to the system, the algorithm in slow-sync mode is used for required data sets.

2.4 Synchronization protocols - summary

Since there are many various data synchronization protocols most of them seem to be designed for pair processing. This characteristic has an effect in poor scalability of described data reconciliation algorithms.

Table 2.7 presents the overall comparison of three chosen synchronization protocols which seem to be the most relevant for this project. What is more, these algorithms have some potential to be adopted in a network environment.

2.5 Networked systems architectures

Currently there a lot of various structure models used in today's networking. This part of this chapter brings closer and compares advantages and disadvantages of two models: client-server and peer-to-peer network.

2.5.1 Client-server model

Client-server is an example of network model where most operations important for the system are processed by one, central machine or process (server). This machine (process) takes control of the current view on the whole system and manages all operations. All other machines (clients) connect to the central server if they have some requests and force the central machine to do some task for them.

Client-server architecture is very simple to implement and manage, because there is only one point which is responsible for performing tasks of the whole system. What is more, the number of points of failures important for the system servicing is limited only to the central server, so improving security and fault-tolerance on the central server can provide a very effective and reliable environment for networked task processing.

On the other hand this only one point of management can lead to various problems. First of all, the scalability of client-server model can be limited depending on the power of central server and link performance between server and its clients. What is more,

	Palm HotSync	**Time stamps**	**CPIsync**
easy to implement	yes	yes	no
depends on the size of data	yes (slow-sync mode), no (fast-sync mode)	no	no
depends on the number of differences	yes (fast-sync mode), no (slow-sync mode)	yes, but it can be confusing if used in network environment in appropriate order of synchronization - some data records can be transmitted a few times what reduces the efficiency of the algorithm	yes, but the probabilistic scheme used for discovering set differences is effective when the number of differences is low in the relation to the whole data set, otherwise it can be inaccurate
one-way / two-way synchronization	one-way/two-way	two-way	two-way
easily scalable	no, because it requires to maintain information about the last synchronization, otherwise slow-sync mode is used and the whole data sets are transmitted	yes, but inappropriate order of synchronizing devices can effect in a reduced efficiency, because some records can be transmitted repeatedly	yes
other characteristics	two modes of use, requires to maintain additional data about data set changes; Palm HotSync is the first and the simplest synchronization algorithm used; algorithm applied in various industrial projects (e.g. Intellisync Data Sync)	wrong order of synchronization can provide inefficiency; applied in various industrial projects	mathematically complex, when the number of differences is high determining upper bound can be expensive

Table 2.7: Comparison of various synchronization algorithms

high number of clients cooperating with the central server can lead to bottlenecks. Finally, the failure of the central point in this architecture immobilizes the whole system.

2.5.2 Peer-to-peer networks

Introduction to peer-to-peer network model

Nowadays peer-to-peer systems (P2P) are more and more popular. The increasing number of people connected to the Internet and their bigger and bigger needs concerning downloading a lot of data from other users caused many scientists to work on developing this approach.

Peer-to-peer is also a very controversial network model, because is scalable very well and distribution of data among thousands of connected peers can be performed quickly and with a high level of anonymity, so sources of illegal software or music files cannot be detected and eliminated in an easy way.

Definition of peer-to-peer (p2p)

As the name suggests in peer-to-peer networks every machine has equal rights. It means that every peer can act as a server and as a client at the same time. Peers can establish connections with other peers and transmit data without the presence of any centralized server. What is more, the stored data can also be distributed among participating nodes, what can have a very positive impact on such systems' performance. Figure 2.6 shows how the connection between nodes is maintained in both centralized and peer-to-peer network.

Considering peer-to-peer systems a few characteristics (derived from [8]), which can have influence on effectiveness of P2P networks, should be analyzed. These are:

- **decentralization** - in peer-to-peer systems data resources are not only at one point of the network, but on a few machines what can have a very positive impact on performance and prevent to bottlenecks or hardware failures;

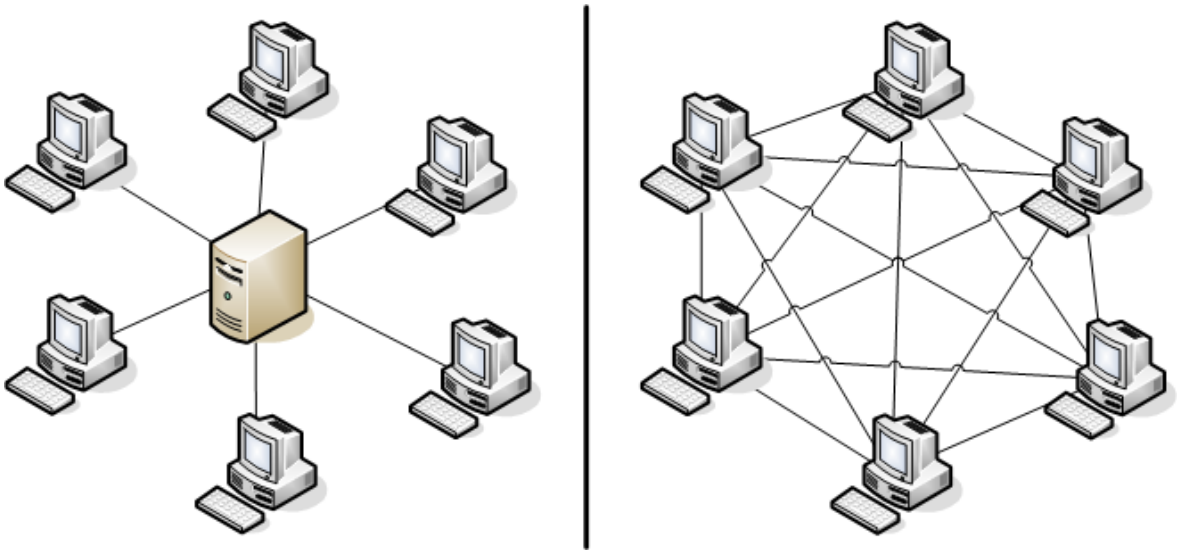

Figure 2.6: Centralized systems versus peer-to-peer model

- **scalability** - P2P networks are highly scalable thanks to decentralization and data sharing; networks can consist of thousands of nodes and the number of peers should not have negative impact on network availability;
- **anonymity** - peer-to-peer systems can be easily enriched in solutions which provide high level of anonymity, what can be desired in case of data transmission; it is also very controversial, because anonymity can make it possible to send and receive illegal data such mp3 files or software;
- **self-organization** - there have to be applied a mechanism which can provide automated peers management, because in peer-to-peer networks nodes can join and leave intermittently and the network can grow unpredictably;
- **cost of ownership** - thanks to decentralization and self-organization costs of ownership peer-to-peer networks are pretty low, because the ownership in such systems is shared between participating nodes;
- **ad-hoc connectivity** - contrary to centralized networks in peer-to-peer systems using mobile devices is not so important problem, changing accessibility of networks nodes can be handled in an easy way;

- **high performance** - thanks to decentralization the performance can be much higher in P2P networks, it can be achieved by data replication and/or caching on various nodes and by selection of appropriate efficient routing algorithms;
- **security** - in peer-to-peer systems security is limited, because of number of connection between nodes, so some mechanisms such as key encryption have to be implemented, especially in files sharing systems, where various parts of files can be transmitted among various peers;
- **fault-tolerance** - thanks to decentralization peer-to-peer systems are resistant to nodes or connection failures and the whole system can continue servicing even if a couple of peers are disconnected.

Pure vs. hybrid model

There can be distinguished two models of peer-to-peer systems. The first one - *pure* peer-to-peer model does not include any centralized element. In a *hybrid* model there is one centralized element which is usually used by peers to find some metadata about connected peers and resources they store. Thanks to this metadata a new peer can directly connect to the chosen peer which stores the requested resource.

As Example of pure P2P systems can be given Gnutella and Freenet. As a hybrid systems we can consider Napster and BitTorrent.

Hybrid model

A new node initializes connection with centralized point (some server - see Figure 2.7A) which stores metadata about peers (such as peer identifier, peer address, information about resources stored etc.). After receiving metadata about other peers it can connect to the chosen peer and join the network (Figure 2.7B).

This solution has obviously advantages and disadvantages:

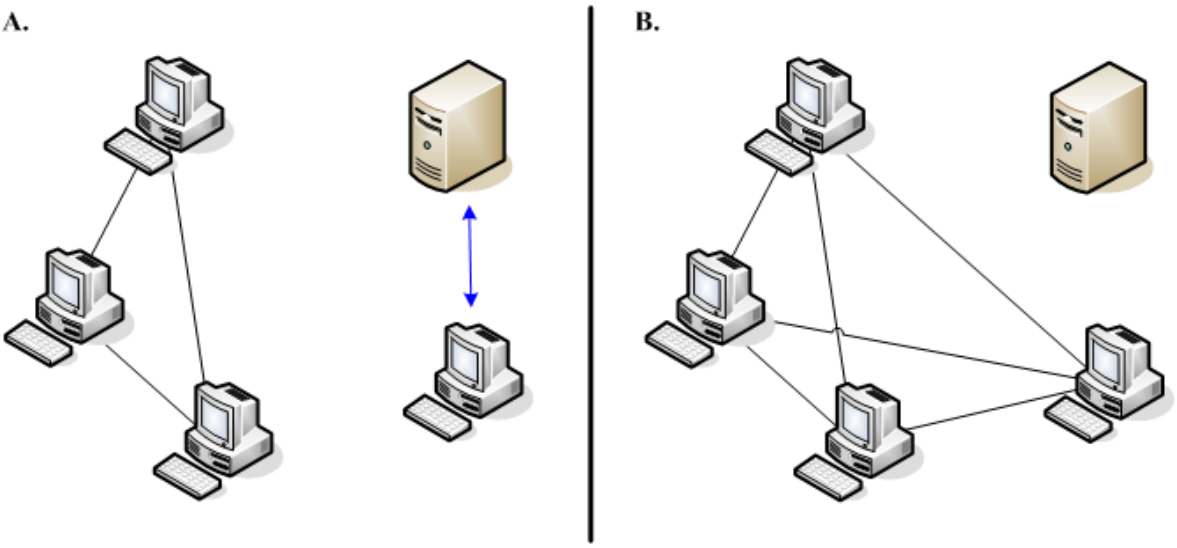

Figure 2.7: A new peer joins the network (hybrid peer-to-peer model)

Advantages	**Disadvantages**
- some control of resources and peers - all joining peers connect to the same point - they do not have to look for anyone in the network, they just remember the address of central point - easy to implement better security	- central point - in case of its failure no new connection can be established - possibilities of bottlenecks

Table 2.8: Hybrid peer-to-peer model: Advantages and disadvantages

Pure model

In pure peer-to-peer model if a new node if wants to join the network, it has to know an identifier (for example: IP address) on one active node. It can be difficult to achieve, because in P2P network model peers can join and leave intermittently and they do not have to be active all the time. A new peer connects to one chosen active peer (see Figure 2.8A) and receives from it a metadata file with information about other peers (or chosen group of them) and available resources. Then it can establish connections with other peers (Figure 2.8B).

Peer-to-peer today - case studies

This section describes in a briefly three chosen peer-to-peer systems.

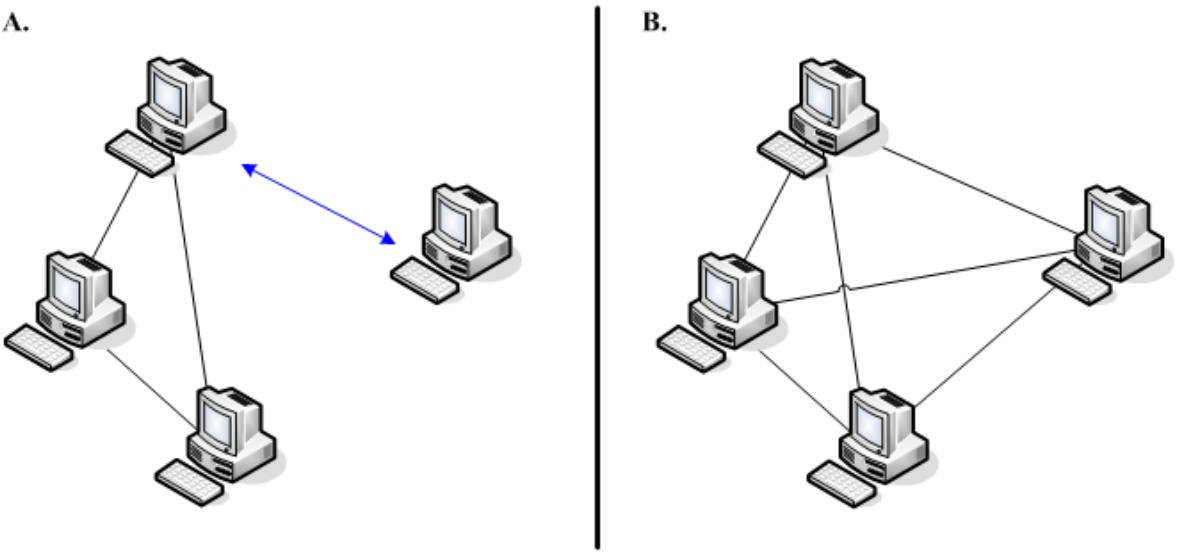

Figure 2.8: A new peer joins the network (pure peer-to-peer model)

Advantages	**Disadvantages**
- fully distributed - an work even in case of peers failure	- difficult to implement - difficult to find an active node - requires applying mechanisms for peers management

Table 2.9: Pure peer-to-peer model: Advantages and disadvantages

Napster

Napster [14], a peer-to-peer file-sharing system released in 1999 was the first distributed application which made it possible for people to search and share mp3 files. This system, which was very popular till 2001, was built on hybrid p2p model basis. A central server stored directory with information about active clients and files held by these users (metadata such as filename, size etc.).

A user who wanted to download some file, first had to gain information about other client which stored this file, by connecting to central Napster server. The server replied with information (metadata) about the client who had requested resource. The downloading process was performed without central server's participation.

Napster also provided other functions such as private chat or messages features. Is was not a secure protocol - all files and passwords were unencrypted.

Although this simple solution was very scalable the problems with the law and copyrights caused Napster to be closed.

Gnutella

Gnutella[15] is a typical pure peer-to-peer system. There are no any central points in the system and all users are equal rights. What is more, files are exchanged between users directly.

Gnutella is a protocol, not a concrete application or software. The first application based on Gnutella protocol was released by Nullsoft company at the beginning of 2000.

To connect to the Gnutella network a new peer has to know at least one currently active Gnutella node. Its address can be found on a web page or in another way. After establishing a connection to this node it sends a list of its neighbors. This new peer tries to connect to these peers from these lists creating his own list of Gnutella peers (non-active peers are removed from the list).

When a peer wants to download a file it has to find a peer which stores this resource, so it sends a message to all its neighbors. They forward this request to their neighbors etc. If some node has requested file it connects to the original sender (its IP is in the request message) directly and finally transmits the file.

Thanks to such decentralization it is impossible to close Gnutella network (as Napster), because it can exist even if there are only two peers connected.

Gnutella is very popular today and there are a few various clients released for most of the platforms. It is estimated that there are almost 1.8 million Gnutella clients.

BitTorrent

BitTorrent[16, 17] is another example of a file sharing system based on hybrid peer-to-peer model. In this system to download files peers have to upload other data simultaneously. Downloaded files are divided into 256KB parts and various parts can be downloaded from various peers.

There is only one centralized point which is used to find other peers. This point

is called a *tracker* and usually it is a web server hosting *.torrent* files which include metadata about active users. If a new peer wants to download a file it connects to the tracker of this files and receives a list of (randomly chosen) active peers. Then it can connect to these peers and start downloading.

In this hybrid system, the tracker does not take part in any downloading process, so if it fails any downloading process is not interrupted. The failure of the tracker only makes it impossible to increase the number of participating peers (no new peer can join the network if the tracker is down.).

BitTorrent also applies many other mechanisms to improve file sharing between nodes. These are for example chocking and unchocking policies or free-riding preventions which eliminate peers which do not upload any files.

2.6 Conclusion

The state of the art in the area of data synchronization shows that there exist number of various protocols designed for data exchanging and set reconciliation. The algorithms apply various ways of modified information marking and distinguishing - some by simple flagging modified records other by using more sophisticated mathematical or probabilistic methods.

The investigation of these protocols revealed some common problems of set reconciliation protocols. First of all, most protocols are designed for processing only between two devices, which can have a very negative impact on the scalability of these investigated protocols. Secondly, some protocols are reliable and provide fast reconciliation only on specific data sets. It means, that in various situations, e.g. when the number of differences is high, such algorithm cannot provide the service at the highest level of efficiency.

The survey of various of data synchronization protocols also presents that the costs connected with adapting such mechanism to the networked environment to enable data reconciliation between more than two nodes at the same time could depend on

the architecture and network model. Two distributed architectures - client-server and peer-to-peer model with the focus on that second one were analyzed in details.

In distributed architectures also very important are schemes of message exchanging which can minimize the network traffic. Such solution can be provided by applying some gossip protocol. The next chapter investigates a number of such mechanisms.

Chapter 3

Gossiping - the way of information dissemination

The success of data synchronization relies on the way of messages exchanging between devices which take part in the process of set reconciliation. In pair synchronization where only two machines are engaged in the process of synchronization only simple data exchanging protocol is required, but in a network environment, where the number of participating nodes is very high some sophisticated methods of informing other nodes must be applied.

There are a few mechanisms, which can be used to send data to all network members. Some of these mechanisms are network-topology aware, what means that physical links between computers are very important factor in message propagation. Multicast and broadcast are commonly used in distributed systems to propagate messages among connected computers, but all these have their specific use and limitations. Multicast (sending a message by one node to all other in the network) can be a very expensive solution (can require a lot of communication), especially for large, distributed systems, where the number of nodes is very high and where nodes join and leave intermittently. Another way to disseminate information over the large network are **gossip algorithms** (also called 'epidemic protocols').

3.1 The power of gossip protocols

Gossip protocols are used often and often in large distributed systems where the number of messages sent by each node should be limited to minimum.

Epidemic protocols work similarly to the nature: when somebody is infected by a virus, the disease spreads to the large number of people rapidly and without any frequent contact with the originally infected man.

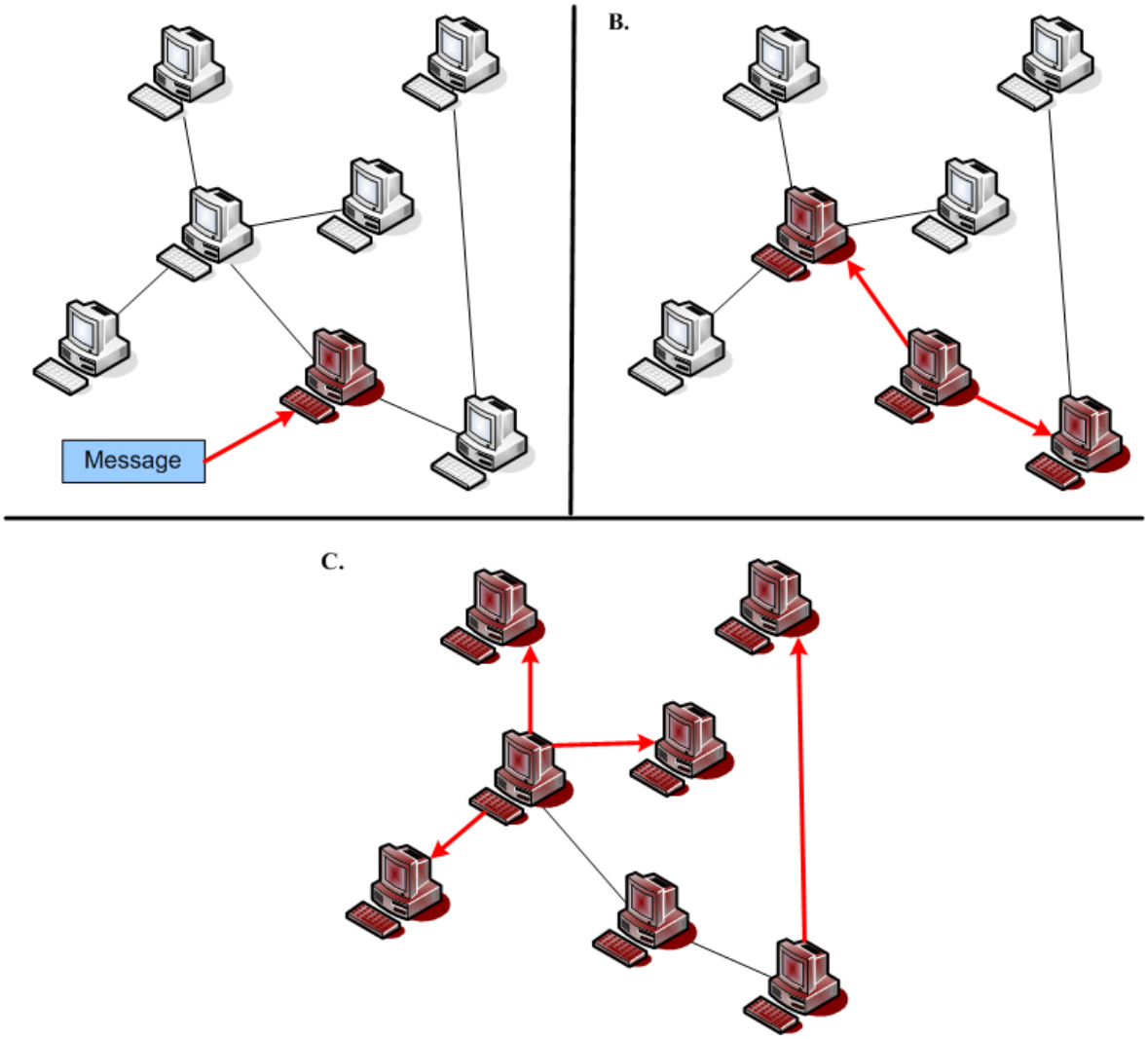

Figure 3.1: Gossiping in a network - example of message spread

In distributed systems gossip protocols spread the information through the network basing on the following scheme: when a node receives a gossip message (or initialize gossiping itself - it can send gossip message to itself) it buffers this information and

forwards it to the randomly chosen number of its neighbors (we consider as a neighbor of a node any other nodes directly connected to it). Each of its neighbors buffers the message and forwards this information to its randomly chosen nodes. The message dissemination process runs until all nodes receive the message. Figure 3.1 shows the general example of message dissemination using an epidemic protocol.

Another definition of gossip problem describes this in another way: each network node has a piece of information and the problem is to find an appropriate communication strategy that each network node can know the whole cumulative message [18]

Since gossip algorithms make it possible to spread a message through the network in a very effective way, they are still under development and research investigation. In the scientific literature there can be found a high number of documents devoted to the gossiping problem.

This part of my dissertation report presents and investigates some chosen epidemic protocols. The analysis and evaluation is split into two parts:

Gossip protocols for interconnection networks. These are the most common used epidemic protocols, which can be applied easily and with high efficiency,

Gossip protocols for unstructured networks. Design an optimal gossip protocol for unstructured, arbitrary network is an NP-complete problem, but some projects applying gossip schemes in unstructured network were designed,

3.2 Model of analysis

The analysis of various gossip schemes is based on a number of factors which can have impact on the use of chosen algorithm. This investigation concerns mainly on algorithms which can be applied in interconnection networks. The factors which are used in analyzing and comparing all these methods are following:

Assumptions and the process flow. Presentation of various assumptions required to apply the analyzed gossip method. These assumptions can vary from assum-

ing specific network structure for chosen protocol to various types of required membership management protocols,

Performance. In most cases it is impossible to measure the number of packets (bytes) or messages' exchanges to perform the whole process of gossiping over the network. The performance of epidemic protocols can be measured as a number of rounds required for message dissemination to all network nodes. This type of measurement is applied in this analysis,

Complexity. The complexity is not a mathematical factor, but it presents how complicated is the chosen protocol and how difficult is to implement it for specific network structure. This factor also describes other dependencies required to apply to run the analyzed epidemic protocol,

Application. This factor shows in which scenarios analyzed protocol can be used.

3.3 Algorithms for interconnection networks

3.3.1 Gossiping in the complete graph

The first algorithms analyzed in this section was designed for message dissemination in the network of the complete graph structure.

The complete graph is a network structure where each node is connected with each other. Figure 3.2 shows the examples of the complete graph of 4 and 5 nodes.

Assumptions and process flow

The presented algorithm for the complete graph structured network assumes two things:

- Network topology - The (logical) topology of the network has to be implemented as a complete graph of n nodes. Each node has to be connected with each other, so in the practice it means that each node has to store information about all other nodes in the network,

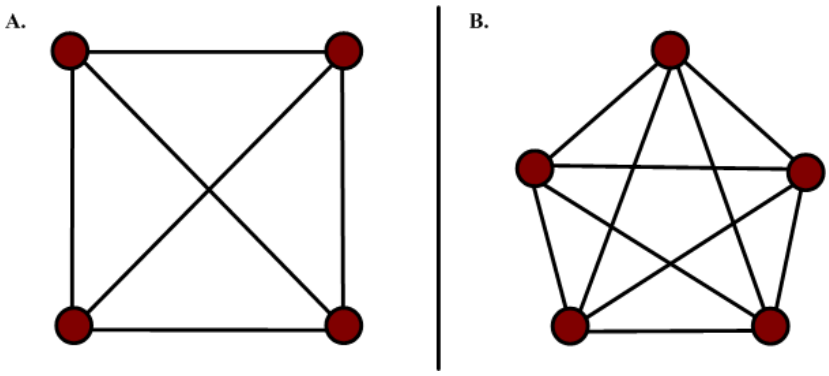

Figure 3.2: Interconnection networks: the complete graph of 4 (A) and 5 (B) nodes

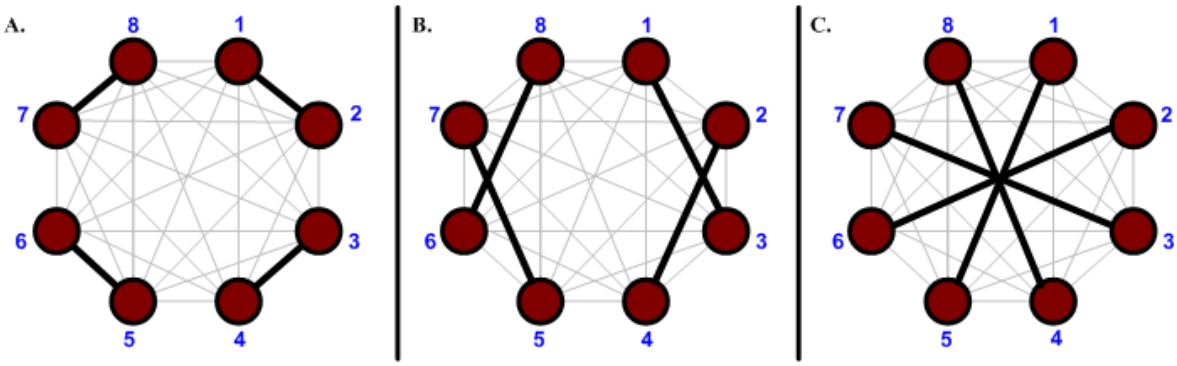

Figure 3.3: Message dissemination in the complete graph of 8 nodes (example use of the proposed algorithm)

- Two-way-gossiping - Presented algorithm is designed for two-way message flow. It means that each node during each round can act both as a message sender and as a message receiver

The two-way gossip algorithm in the complete graph consists of three steps [18]:

1. Send the information of the node $i + m$ to the node i for all $2 \leq$ i $\leq m$ +1
2. If $m + 1$ is even, than gossip in 1,2,...,$m + 1$. If not, than gossip in 1,2, ..., $m + 2$.
3. Send the information of the node $i + m$ for all $2 \leq i \leq m + 1$

where nodes are number from 1 to n and n = $2m$+ 1.

Figure 3.3 presents an example use of the algorithm for the complete graph of 8 nodes. It can be proved that for this graph a message can be spread to all nodes in three rounds.

Assuming that node 1 stores message A during the first round it communicates with node 2 and exchange information. So, after the first round both node 1 and 2 have message A. In the second round, node 1 communicates with node 3 and node 2 exchanges information with node 4, so after this round all nodes 1,2,3 and 4 have information A. Similarly, in the third round node 1 communicates with node 5, node 2 with 6, node 3 with 7 and node 4 with 8. Finally, after the third round all nodes have information A.

Performance

According to the proof described in [18] the algorithm presented above gossips the message to all nodes in the network of the complete graph structure in $\lceil log_2 \boldsymbol{n} \rceil + \mathbf{1}$ rounds.

Complexity

Two-way gossip algorithm for the complete graph is not so complex. The whole process is performed only in three steps and any complicated calculation are not required. What is more, It does not require applying any sophisticated membership management protocols, because all nodes have information about each other.

Application

This algorithm can be applied to disseminate message in networks where it is easy to organize nodes in the structure of the complete graph. Since all nodes store information about each other, applying this algorithm for large, distributed network can cost a lot or can be even impossible for the reason of network dynamism.

3.3.2 Epidemics in hypercubes

The next algorithm analyzed in this part this project is an epidemic protocol designed for hypercubes which is one of the most common network structures.

According to the definition presented in [18]: the hypercube H_m is a graph whose nodes are all binary strings of length m and whose edges connect those strings which differ in exactly one position.

Each hypercube H_m has 2^m connected nodes. An example of the hypercube H_3 is presented on 3.4.

Assumptions and process flow

The algorithm presented by David Krumme in [20] is much more sophisticated than that one designed for the complete graph and it is also mathematically complex. Because of the space limitation only the general view on this is presented and its characteristic according to the analysis model are described.

Similarly to the algorithm for the complete graph presented in the previous section this algorithm also assumes specific network topology which consists of 2^m nodes according to the considered hypercube's degree. Such structure can be more difficult to maintain in a dynamic network.

Figure 3.5 taken from [20] shows which pairs of nodes exchange information during each step of gossiping in 9-cube.

Performance

According to the proof presented in [20] the algorithm presented above gossips the message to all nodes in the network of the hypercube H_m in $1.88m$ rounds.

Complexity

The algorithm presented by Krumme in [20] is more complex than that one for the complete graph presented in the previous section. This algorithms is mathematically

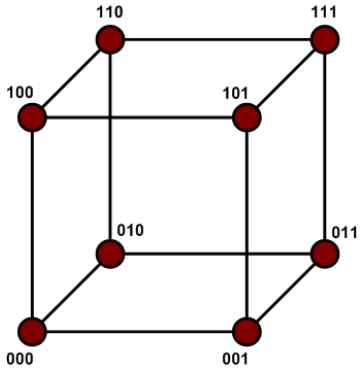

Figure 3.4: Interconnection networks: the hypercube H_3

difficult and requires to perform initial steps before message dissemination. What is more, preparing such network structure can be difficult for dynamic environment.

Application

This algorithm can be applied to disseminate message in networks which have structure of the hypercube or which can be reorganized in an easy way to create such structure. This algorithm has a significant potential to disseminate information through the whole network in short time.

3.3.3 Message dissemination in the cycle structure

Assumptions and process flow

The algorithm assumes that the network is organized in a cycle (ring) structure, so a cycle C_n has n nodes connected. Figure 3.6 presents the network of the cycle of 5 structure.

The detailed algorithms and proofs for both one-way and two-way modes are described and analyzed in [19].

Performance

According to the investigation and proofs presented in [18] gossiping in network of cycle structure of n nodes can be realized in $\lceil \frac{n}{2} \rceil$ rounds (two-way mode) and in $\frac{n}{2} + \lceil \sqrt{2n} \rceil - 1$

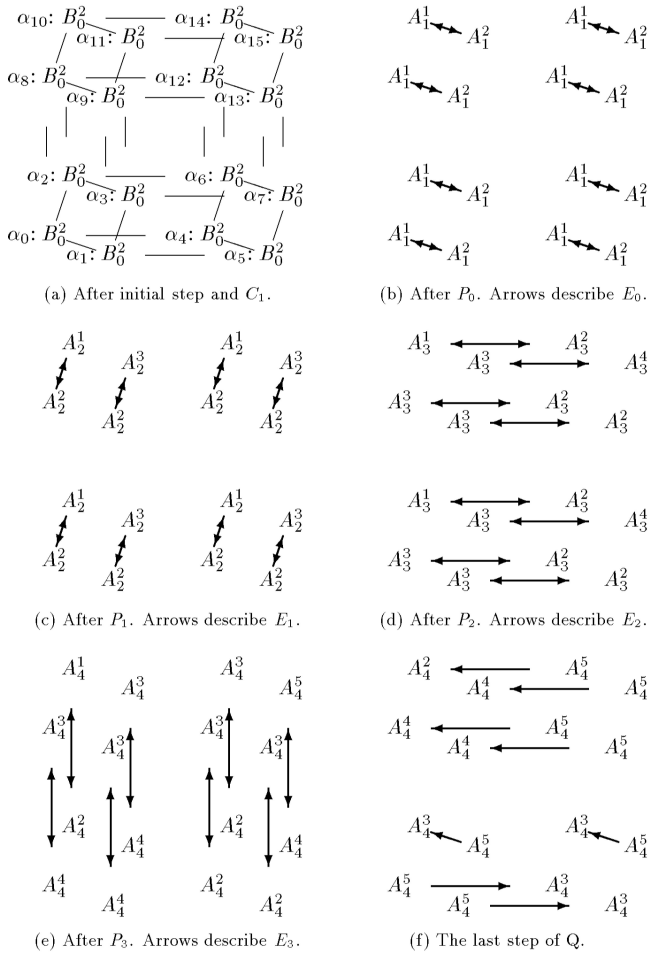

Figure 3.5: Interconnection networks: message dissemination in 9-cube (from [20])

(one-way mode).

Complexity

The epidemic protocol for message dissemination in the cycle structure is not so complicated and it is rather easy to apply in such network. Additional assumption have to be applied when the number of connected nodes is odd.

Application

This algorithm can be applied to disseminate message in networks which have structure of the cycle (ring). Building networks of such structure can be much simpler than of the structure of the complete graph of hypercube. What is more, maintaining such

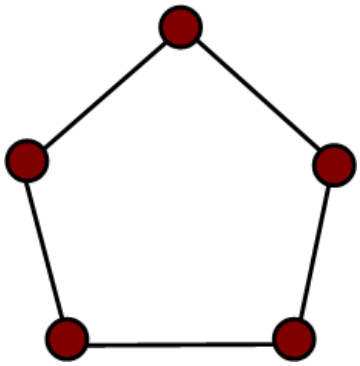

Figure 3.6: Interconnection networks: the cycle of 5 nodes

networks is cheaper even if nodes join and leave dynamically.

3.3.4 Other algorithms

There are also other similar algorithms designed for interconnection networks such as cube connected cycles, butterflies, grids etc. Detailed description, analysis and comparison of all these methods is presented in [18] and [19].

3.4 Algorithms for unstructured networks and related projects

The choice of nodes to which messages are forwarded is more difficult in unstructured networks because various nodes maintain (or not!) connections with some other nodes. A good scheme should ensure delivering these messages to all nodes in the system in minimal number of rounds. What is more, the same nodes should not receive the same message a few times.

There is a variety of gossip methods for unstructured networks. Assuming that each node maintains two lists: one with information about its neighbors (peer list) sorted according to the latency, so the closer nodes are at the top of this list and the second one with data sets it sorted (information list) according to the freshness of messages the, following methods, also described in [24], can be described:

Broadcast. This model is used when a peer sends all the messages included on the information list to all peers included in peer list. Such method is not so efficient, because most nodes receive the same information a few times,

Random. In this solution a peer selects only a few records from the information list and a few neighbors from the peer list. Such model generates much low traffic on the network than broadcasting,

Top. Another solution, similar to random model is to choose a few top peers (these are these which are the closest) and a few top messages from information list (these are these messages which are the newest)

There also exist other, more sophisticated and mathematically complex methods to choose the set of information which should be gossip and the set of peers which should receive these information during each gossiping round. These are, for example, Spatial gossip, investigated in details in [22] or Bin-Halving gossip scheme presented in [24].

Such algorithm should provide an information and mechanism to choose appropriate nodes from neighbor list and messages to ensure that in a minimal number of gossip rounds these message will be delivered to all active nodes connected to the network. Finding such solution can be difficult (see section 3.5).

3.4.1 Astrolabe

Astrolabe is a project initialized by researchers from the Cornel University (see [23]) designed for performing scalable data fusion and data mining functions to replicate information between data base application running on various machines connected in peer-to-peer manner. Information spreading in Astrolabe is based on gossip scheme and communication.

Astrolabe is designed to be used in large networks. Aggregation of information stored in a form of data base (each information as a separate row) is done by using simple SQL queries.

More details concerning this project are presented and investigated in [23].

Although Astrolabe and its description seem to be a good point to consider when thinking about distributed data synchronization architecture, no detailed algorithms and other protocols were found in the available literature.

3.4.2 PlanetP

PlanetP is an information sharing system based on unstructured peer-to-peer network model with use of gossip schemes to communicate and disseminate information is called PlanetP presented in [25] and [26].

Messages (rows) stored by each machine using PlanetP are described using eXtensible Markup Language (XML). The whole sets of information stored by one user are presented using Bloom filters [13]. To exchange information a gossip scheme is used as a protocol of communication between connected peers.

The gossip scheme applied in PlanetP requieres maintaining information about all nodes by each connected peer. It is a big disadvantage because it can limit the scalability of the whole system.

More details about PlanetP are presented in [25] and [26].

3.5 The gossip problem

Although gossiping seems to be powerful and works in many network structures such protocols designed for interconnection networks in many cases it requires fulfilling additional conditions: creating a specific network structure, maintaining connection between high number of participating nodes or even knowing each other by every connected node. What is obvious, such solutions cannot work in an environment where nodes join and leave intermittently, where the delivery of some parts of data sets is not reliable or where network partitions can appear.

For such scenario for which an architecture for data synchronization is going to be built an investigation of gossip protocols for unstructured, arbitrary networks was performed. There are a lot of various algorithms and research papers devoted to message

dissemination in peer-to-peer model networks, but the protocols presented there are not satisfying and do not address all the problems and requirements connected with the scenario of simultaneous data synchronization between more than two nodes.

What is more, finding an optimal solution for message spreading in dynamic network can be extremely difficult. [21] presents an information that:

> 'It is shown in [..] that broadcast and gossip are both NP-complete problems for arbitrary networks..'

and

> 'The broadcast and gossip problems are NP-complete in most of usual communication models considered in the literature.'

Basing on such observations of NP-completeness of gossip problem and the early stage of their development, analysis of other methods of message exchange based on already known solutions was began. After analyzing all the aspects and knowledge presented in chapters 2 and 3 an architecture for multi-party synchronization of data sets in a distributed environment was designed.

The next part of this report presents the detailed design of this architecture and justifies main decisions made during this process.

Chapter 4

Design

4.1 Introduction

This chapter describes and discusses some key decisions made during architecture design process of this project. Firstly, it describes the aim of the project and main requirements which should be addressed at the design level. Secondly, the main components and dependencies between them are considered and analyzed. Finally, proposed algorithms, additional options, considerations, and improvements are presented.

4.2 The aim of the project

The main aim of this project is to design and implement an architecture which provides a possibility of simultaneous set reconciliation in a network environment. It means that more than two devices such as personal or handheld computers can cooperate at the same time to synchronize their data sets. The proposed system is designed for distributed environment with the adoption of the scenarios and data synchronization algorithms described and analyzed in Chapter 2 and 3. Applying fast data reconciliation algorithm ensures efficient results in synchronization between network nodes.

4.3 Requirements

The analysis of data synchronization protocols and the investigation of gossip schemes as also advantages and disadvantages of common architecture of distributed systems provide a view of the main requirements which should be addressed by the architecture designed for networked, simultaneous set reconciliation process. The list below describes project specific requirements which should be addressed at the design stage.

1) **Distributed architecture.** The system should provide a service for various number of nodes connected in a network form. What is more, the cooperation between nodes and tasks processing should be distributed i.e. tasks should be processed by various participating nodes at the same time in various network places,

2) **Efficient set reconciliation between two nodes.** The method adopted to information exchange of data sets between participating nodes should depend on the number of differences not on the size of the whole data set stored by a node,

3) **Fast propagation of changes between all nodes.** The system should be based on small group synchronization, so an appropriate scheme of message exchanging between nodes should be adopted in order to disseminate information in the fastest possible way. What is more, the system should provide a clear structure of the network and each node should have only a partial view on the whole system.

4) **Clear structure.** To match the network topology and performance a clear nodes structure should be implemented in case of group synchronization process to connect nodes which are in neighborhood physically (number of hops, round-trip time or even the same class of IP address) and which have similar connection bandwidth,

5) **Clear format of data sets.** Data sets should have a simple structure. They should be stored in a simple data base system (for example MySQL) or in text files (Comma Separated Values file (CSV) or similar). Some additional fields such as

checksums or hashes can be added (depending on the algorithm applied) to make the synchronization process faster,

6) Simultaneous synchronization. The system should provide a mechanism which makes it possible to start synchronization process between a few nodes (or pairs) at the same time. It means that various pairs (groups) of nodes should perform set reconciliation concurrently.

7) Scalability. Scalability is a very important factor. The system should be designed and implemented in a way which makes it possible to maintain data and network consistency even if the number of participating nodes or resources grows rapidly. There should not be any limitation concerning number of participants and the system should work for both small and big groups.

8) High level of fault-tolerance. The designed system should be highly fault-tolerant which means that it should address most problems connected with any element failure (even hardware) to continue providing the service.

4.4 Basic design decisions

This part of the chapter proposes a solution which addresses the requirements and provides an easy way of data reconciliation in a distributed environment.

To find the best solution for networked data reconciliation preliminary investigations were done in three areas: existing data synchronization protocols, modern distributed systems' architectures and gossiping protocols.

The first part of this research, the analysis of existing data synchronization protocols reveals that most today's protocols for data set reconciliation are designed for pair synchronization. It means, that synchronization with various nodes can be more expensive, because can require storing additional data about other devices (similar solution is applied by SyncML initiative [4, 5]]). Since the fast-sync mode of Palm HotSync protocol[9] provides the best results in term of time and number of messages

required to process; adapting this to the networked scenario could provide potential solution for fast synchronization between high number of nodes.

The in-depth analysis of two distributed systems architectures: client-server and peer-to-peer model presented in Chapter 2 shows that in many cases peer-to-peer networking is desirable, because tasks processing can be divided between a lot of participating devices and the bandwidth can be used in a better way. On the other hand, in case of data synchronization such network model has to apply a special communication mechanism ensuring set reconciliation in short time. Such mechanism potentially could be realized by gossiping.

Although gossip protocols seem to be powerful applying such epidemic algorithms to dynamic environment where nodes can join and leave intermittently is not so simple. Moreover, NP-completeness of gossiping for arbitrary networks, stated in [21], discourages from adopting such schemes.

The investigation of gossip protocols and the difficulties connected with finding a fast solution for reliable two-way message dissemination in a dynamic, arbitrary network shows that applying gossip algorithm in the system realizing parallel data reconciliation process can be very difficult and finally does not have to provide efficient results. What is more, although gossip protocols for interconnection networks such these presented in [18] and [19] are very popular, building such specific network structures can be extremely expensive and infeasible to do in the dynamic environment. Moreover, although the solution for networked data synchronization presented in [11] works well (or the gossip scheme for the complete graph presented in Chapter 3), the scalability and network efficiency is extremely limited, because each node has to maintain the connection with each other, what could be unreachable in networks where the number of nodes is really high. It shows that such solutions should be omitted.

What is more, some level of system centralization, although expensive and not comfortable, could help to avoid storing information by any device about any device and finally decrease number of connections required to maintain in case of applying gossip protocol. Intellisync Data Sync [6, 7] - a commercial platform which is fully

centralized shows that such fully-centralized solution can work very effectively.

Basing on all these aspects and conclusions drawn from the previous parts of this work a new solution for networked set reconciliation has been worked out.

Two main design decisions which are fundamental for the whole architecture are:

Introducing semi-centralization. Basing on the previous investigations and analyzes semi-centralization seems to be the best solution combining advantages of centralized systems with possibilities of applying the fastest known algorithm dependent on the number of differences not on the whole data set (fast-sync). What is more, such solution does not require maintaining any specific network structure like in case of adopting any gossip protocol for interconnection network. In addition, semi-centralization makes it possible to control the whole communication;

Applying fast-sync method. According to the analysis performed in Chapter 2, fast-sync mode of Palm HotSync protocol[9] is the fastest known synchronization algorithm dependent only on the number of differences. What is more, this protocol can be adopted and implemented with ease. Finally, combining this with semi-centralization of the system makes it possible to service efficient synchronization based on pair communication.

These two basic design decisions make it possible to build an effective way of data synchronization between more than two nodes. A lot of details are explained in further sections.

4.5 General architecture overview

The solution worked out during the course of this project assumes that logically the system network consists of a number of connected nodes (M) which are split into groups depending on the physical distance, bandwidth or other factors. In each group there is one group coordinator (a group agent - A). All group agents are members of the system

coordinators group, so they are directly connected to it. Two members from this group will act as system coordinators (one as a secondary system coordinator (S) in case of the main super coordinator (C) failure). Such structure addresses requirements 1 and 4.

Figure 4.1 presents the high level view on the architecture of the proposed system.

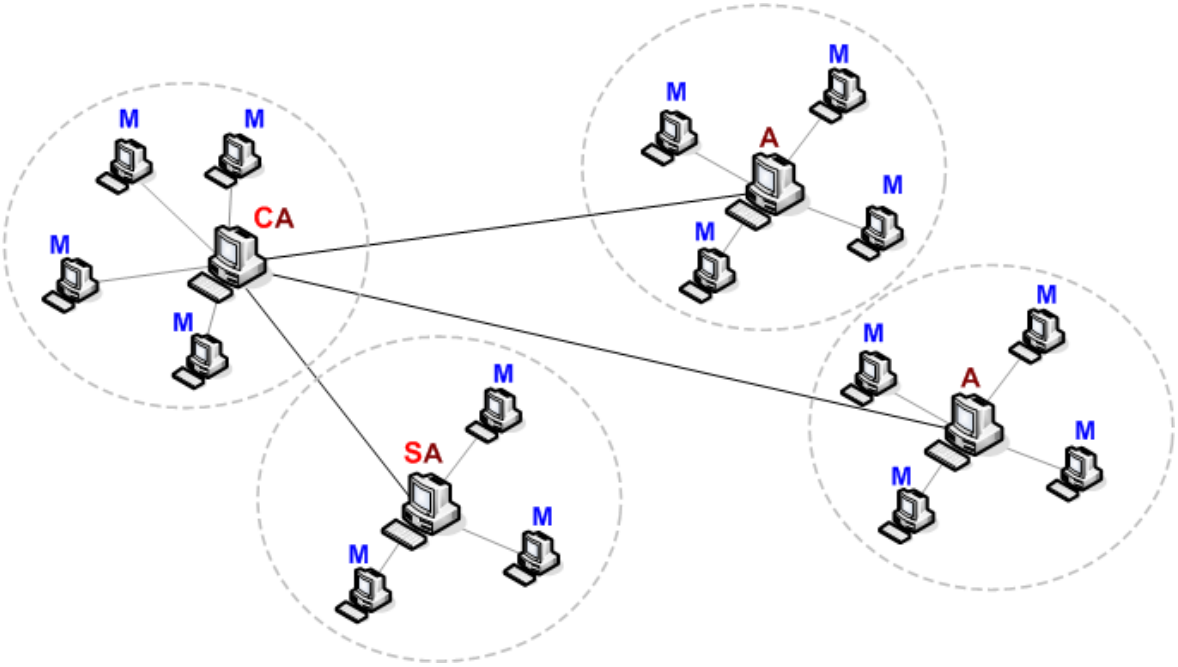

Figure 4.1: General overview of the network structure of the proposed solution. C - system coordinator, S - shadowing system coordinator, A - group agent, M - group member

4.5.1 Synchronization process flow

The process of set reconciliation consists of a few stages performed by various nodes:

1. One active nodes calls for the synchronization sending the request message to its group agent. This group agent forwards this message to the system coordinator asking for the permission to synchronize. If any synchronization was not already started the system coordinator starts a new set reconciliation and sends start message to all connected agents (groups); group agents call their group members for the synchronization,

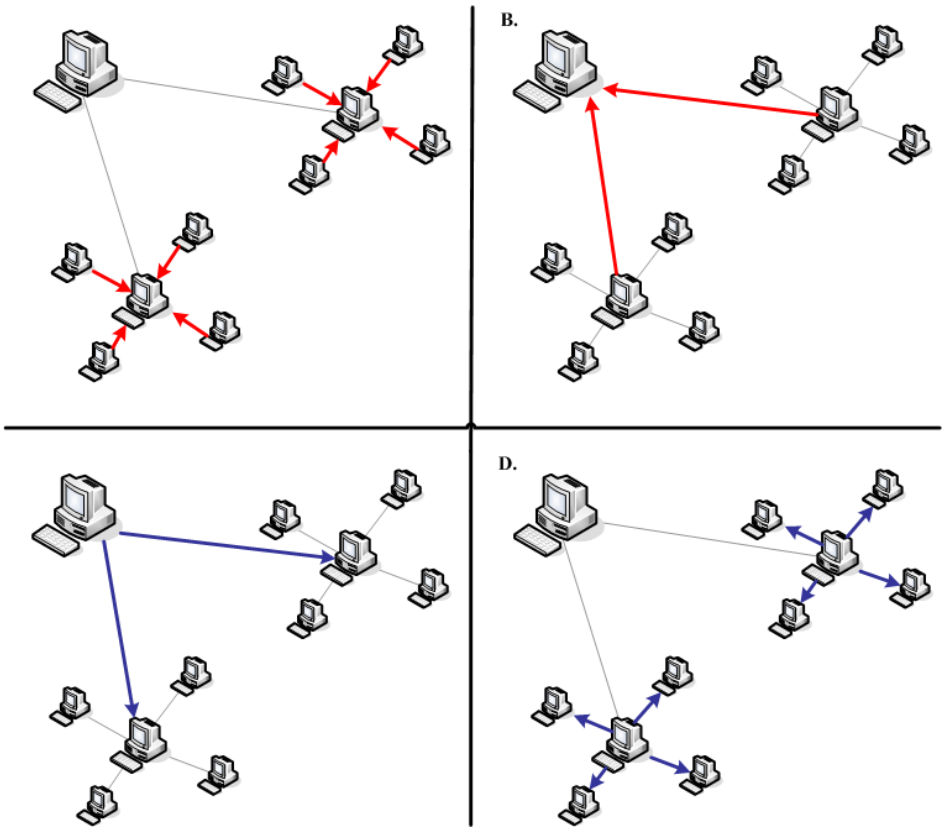

Figure 4.2: Four main steps of synchronization process

2. At the beginning, each group member prepares its own set of differences choosing only modified and newly inserted records (since the last synchronization) and sends its prepared data set differences to its group agent (see Figure 4.2A),

3. A group agent, after receiving set of differences from all its group members, calculates the cumulative set of differences of the whole group and sends it to the system coordinator (Figure 4.2B),

4. After receiving difference sets from all group agents (groups) the system coordinator calculates the cumulative set of differences for the whole system filtering out all the duplicates and sends it back to all group agents (Figure 4.2C),

5. Group agents, after receiving data from the system coordinator calculate common set of data for their group (there is a possibility that they can receive some data they already have) and then send it back to all members of their group, global synchronization process terminates (Figure 4.2D).

This process cycle can be repeated all the time if there is an appropriate request message from one of the users. Such solution addresses requirements 2,3,6 and 7.

4.5.2 Node and its processes

Since the proposed system is designed as a multi-agent client-server network, there are three types of nodes (roles) in the system: a system coordinator, a group agent, a regular peer (group member).

The relationships between these types of peers are following:

- Each regular node can be elected to act as a group agent (in case of current group agent failure),
- Each group agent can be elected to act as a system coordinator (in case of current system coordinator failure),
- Any regular node cannot be elected to act as a system coordinator directly (firstly it has to be elected to be a group agent)
- System coordinator can act as a normal node at the same time (it stores the up-to-date data set).

Each node can play a few roles at the same time, so it has to run a set of processes. These are:

- system coordinator process (if elected),
- group agent process (if elected),
- system member process,
- logging process,
- synchronization process.

Each of these sub-processes includes a few actions which are performed to reach the main system's goals. Every sub-process (except logging) can communicate with connected nodes (with nodes which are known as neighbors) to perform required tasks. The dependencies between various types of nodes as also the possibility of changing their roles dynamically addresses requirements 7 and partially 8.

System coordinator process. This process runs only when the peer was elected to act as a system coordinator or secondary system coordinator. It is the main process in the whole system and it is responsible for keeping consistency of the logical network structure.

It implements an algorithm for peers membership and redirects new peers to chosen group agents to create groups with similar number of members.

Group agent process. Group agent process runs only when a node was elected to act as a group agent. It is responsible for internal (in group) synchronization process - it shows where regular peers should send it their set differences). What is more, this process is also responsible for data exchanging with the system coordinator and propagating cumulative set differences of the whole system (received from the system coordinator) to its group members.

System member process. The system member process runs on every node and provides the main functions such as communication with group coordinator, message exchanging with the group agent.

Logging process. Logging process is optional and it is responsible for saving chosen information about the state of the system in constant time intervals. Logs can be used for peer recovery.

Synchronization process. Synchronization process is directly connected with the group coordinator process and the system member process. It implements data set reconciliation algorithms and makes it possible to exchange information with other system members.

To guarantee these processes' efficiency, a node can store various types of data. These are:

Group coordinators list. This type of data is stored only by the system coordinator. It includes the whole up-to-date list of all group coordinator in the system,

Group members list. The up-to-date list with IP addresses of connected nodes. It also contains fields with information about the time of the last interaction between nodes, current synchronization process id etc. This list is stored by all group coordinators. Regular nodes store information only about its group coordinator,

Synchronization processes list. This list contains information about current data set reconciliation processes,

Data sets. This is a data base with information which are the subject of synchronization process (text file),

Log files. Each nodes stores up-to-date logs which include records with information about previous, finished synchronization processes.

4.5.3 Synchronization algorithm

As presented in section 4.5.1 the global synchronization process consists of three stages. Because of various logical network structures and data stored on by peers during each of these stages, two synchronization algorithms must be applied.

Fast-sync

The fast-sync algorithm is used as a main data exchange method. Both regular nodes and group agents mark their data sets using appropriate flags (modified or not modified since the last synchronization process). Thanks to it the fast-sync algorithm can be applied. It means that only information about new or modified data records are sent to other nodes.

This algorithm is very simple, but it is also very effective and fast. This method is, in comparison to other, not data-size dependent, but the time and communication required to perform data synchronization depends only on the number of differences of data sets (records added or modified).

Fast-sync algorithm will be used in most cases, in other scenarios slow-sync will be applied. Using fast-sync mode fully addresses requirement 2.

Slow-sync

When a new node joins the system it has to receive the up-to-date data set from its group agent (not only set of differences). It means that the group agent has to use slow-sync algorithm to send all his data to the new joining peer.

Slow-sync performance depends on the size of data set, so it is not an effective method, but in the proposed system it will be used only in the scenario described above.

4.5.4 Message format

Since the proposed system is designed for use in mobile devices environment the structure of any message sent should be as simple as possible to limit the number of data required to send during synchronization process.

Data sets should be stored as a text files. The structure of any row will depend on the specific application requirement, so it can include various types of field. The only required message format is the flag which will be used to perform fast-sync synchronization process (inserted, modified, not-modified, deleted).

What is more, to distinguish various information, an unique hash code is used. This hash code is generated when the information is created and it is not changed in case of any modification. This presented format of a message fully addresses the requirement 5.

4.5.5 Joining mechanism

The system coordinator acts according to the following algorithm:

1. The system coordinator is active. A new device wants to join the network and sends a message to the system coordinator (it could be redirected to the system coordinator by any other nodes already connected to the network),

2. The system coordinator receives the message from the new node and checks the current status of the network (number of groups, number of members of groups and number of group agents),

3. - If the difference of number of members in various groups is very high the new user is redirected to the group agent of the smallest group in similar distance (similar number of hops). The chosen group coordinator is informed about the new user as well,
   - If the difference of number of members in various groups is small the new user is redirected to the randomly chosen group. The chosen group coordinator is informed about this new user as well,

4. From time to time the system coordinator asks groups coordinators about status of their groups (number of active nodess)

4.6 Features and improvements

4.6.1 Handling latecomers

The synchronization request requires each group agent to prepare the cumulative set of differences of its group. This set of information is prepared by summing received set of differences from each group member and further filtering out all duplicates. The whole cumulative set of differences of the group can be prepared after receiving set of differences from all group members.

Preparing set of differences by each group member and sending this to the group agent takes time. What is more, there could be a situation when a new node joins the group and starts preparing its set of differences later than other group members. Finally, it can lead to the infinitive postponing the group agent work (and the whole system as well) in a case when hundreds of new nodes join the group with some delay.

To cope with such problem the following solution is applied: Latecomers cannot send their set of differences to the group agent if a synchronization process is has already started, but they can receive sets of information sent by group members by the group agents. What is more, based on this received set of differences of the whole system each latecomer can calculate its own up-to-date set of differences and in the next phase send only part of stored information.

4.6.2 Solution for unknown nodes

When a new node which has not synchronized earlier with any other nodes in the system comes it can request for the synchronization process. During this process it sends all its data records as its set of differences and finally it receives all differences of all other nodes in the system or its group. The problem is that it still does not have any data records which were stored by other users before this synchronization with flag 'not-modified' - these records were not exchanged. Such behavior can lead to the data inconsistency.

To solve this problem a new unknown node after connecting to the group receives the whole group agent's data set (not only differences). After this it can calculate differences between received information and its originally stored data set marking all these records as a new or modified.

4.6.3 Shadowing system coordinator

The semi-centralization of the architecture proposed in this dissertation project provides some possibilities of system failure. The termination of system coordinator could

cause a lot of problems with the communication between agents and finally could eliminate any possibility of synchronization of data sets in the whole system.

The solution for system coordinator failure is the secondary system coordinator which is shadowing the primary system coordinator all the time. It means that secondary system coordinator receives the same set of messages and system updates. What is more, it stores information about all the group agents.

In case of primary system coordinator failure secondary system coordinator starts acting as a system coordinator. Because he has knowledge about other group agents it can designate a new secondary system coordinator. This solution partially meets the requirement 8.

4.6.4 Time stamping

The feature of the proposed architecture is that no node has to be active all the time to have up-to-date consistent information. Each device known by the system, after some time of disconnection should have a possibility to join again, synchronize and finally receive all records modified and inserted by other nodes during the time of the disconnection.

If some node modified some records in the result of synchronization process these records are sent to other devices in the system and are stored after this reconciliation by these devices as not modified, because all flags are cleared so they are not exchanged during next synchronization cycles. Such algorithm can lead to inconsistency of stored information, in case if some node disconnects from the system and misses some cycles of the set reconciliation. After rejoining the system it cannot receive previously modified records because they currently have flag 'not modified'.

To omit such problem each synchronization and each data record is time stamped, so after rejoining the network a node receives all records with the time stamp higher than its last synchronization process in this system. Additional feature of this solution is this, that if there are two or more information with the same hash code (it means that there are two or more versions of the same data record) a group member can

decide which one to choose.

4.6.5 System distinguishing feature

Various nodes can participate in synchronization processes in other similar systems. These systems can have different structure and participants and finally can store different data sets. Taking part in synchronization processes in various systems can lead to data inconsistency because some records maintained by various systems can be represented by exactly the same hash code integers.

To distinguish such records in case of synchronization in two or more various systems additional information is introduced - system identifier which is generated in a way finally this ID key is unique for each synchronization system.

4.6.6 Network partitions

Another problem which is very important in distributed systems for the sake of data consistency are network partitions which are quite popular in mobile, wireless networks where connected nodes can be out of the range from time to time when changing their positions. Network partitions can lead to the temporary disconnection of some nodes in the system and creating two separate subnetworks, so appropriate mechanisms to provide continuous service for both disconnected groups and final merging both partitions has to be applied. The network partitions problem is not so easy to solve in semi-centralized systems, so detailed analysis has to be performed.

There is no problem with partitioning when partitions are created by group agent failure, because similar mechanism such as the one when system coordinator fails can be applied. It means that one of regular group members can shadow group agent and store information about other nodes. In case of group agent failure it can communicate with the system coordinator and nodes in its group and start servicing.

The situation when in one partition are only regular group member is more complicated. Group members do not know each other, so they cannot elect any new group

agents to provide the service of this group. Such situation can happen if both a group agent and the group member shadowing it fails. In this case each node can try to communicate with the system coordinator (group members know its IP address, because they had to communicate to the system coordinator to join the network). Thanks to this, each node can be redirected to any other working group. Such solution is not an ideal method for handling network partitions, but in semi-centralized environment such problem is difficult to avoid.

4.7 Conclusion

The proposed system is designed as a multi-agent based client server distributed system with the main coordinator, which takes control (coordinates) of the global synchronization process and takes control of the logical structure of the network - it redirects joining device to the appropriate subgroup. This central element does not have any impact on the connection and data transmission speed. What is more, the failure of the central point does not have any impact on the process of synchronization - the secondary system coordinator can take its place and service.

All the nodes are split into a few smaller groups. In every group one chosen node acts as a group agent and all these group agents are connected directly with the system coordinator.

The architecture designed and analyzed in this chapter addresses all the requirements (1-8) stated at the beginning of this part of the dissertation report and presents an innovative solution for simultaneous data synchronization process between high number of various types of devices with application of the simplest existing algorithms.

To check the performance of this architecture it was implemented and examined by number of test in various settings. The next two chapters describe details concerning this implementation and provide the results of some experiments performed using this implemented tool.

Chapter 5

Implementation

This chapter provides short information about implementation of the architecture for multi-party synchronization presented in Chapter 4.

For the purpose of experiments which will be performed to show proposed architecture's performance (the total time and number of messages required to perform the whole synchronization process) some options were simplified or omitted and possibility of manual settings was introduced - especially in case of designating the system coordinator and group agents.

5.1 Technology used

The choice of Java language used to implement the proposed architecture is rather obvious. Java provides a lot of facilities and tools which support application programming designed for networked, distributed environment. Built-in mechanisms and added libraries make it possible to implement such application in a very fast way. What is more, object-oriented language which is platform-independent ensures that the final solution can be tested both on usual personal computer and on mobile devices such as PDAs or mobile phones. Finally, the popularity of Java and availability of various extensions and platforms is also a very important factor.

5.2 General overview

The architecture presented and analyzed in Chapter 4 was implemented in Java language with a lot of simplification providing a possibility of manual settings to create an environment for testing the performance (according to the time and number of messages needed be sent to perform the whole process of system set reconciliation in various settings).

The actual version of the implementation consists of the following files (classes):

EmemSync.class This is the main class starting the whole application. Depending on the parameters it can run only client process or agent and coordinator process as well,

CoordinatorProcess.class The class responsible for running the main system coordinator process,

AgentHandler.class The class (thread) responsible for handling all agents communication processes,

AgentDescriptor.class The class creating objects maintaining each group agent's details,

ClientProcess.class The class responsible for running the client process,

ClientThread.class The class running client thread responsible for communication between a client and its group agent,

AgentProcess.class The class responsible for running the group agent process,

AgentClientProcess.class Object of this class starts an agent client process responsible for maintaining communication with the system coordinator,

AgentClientHandler.class The thread responsible for the cooperation between an agent and system coordinator,

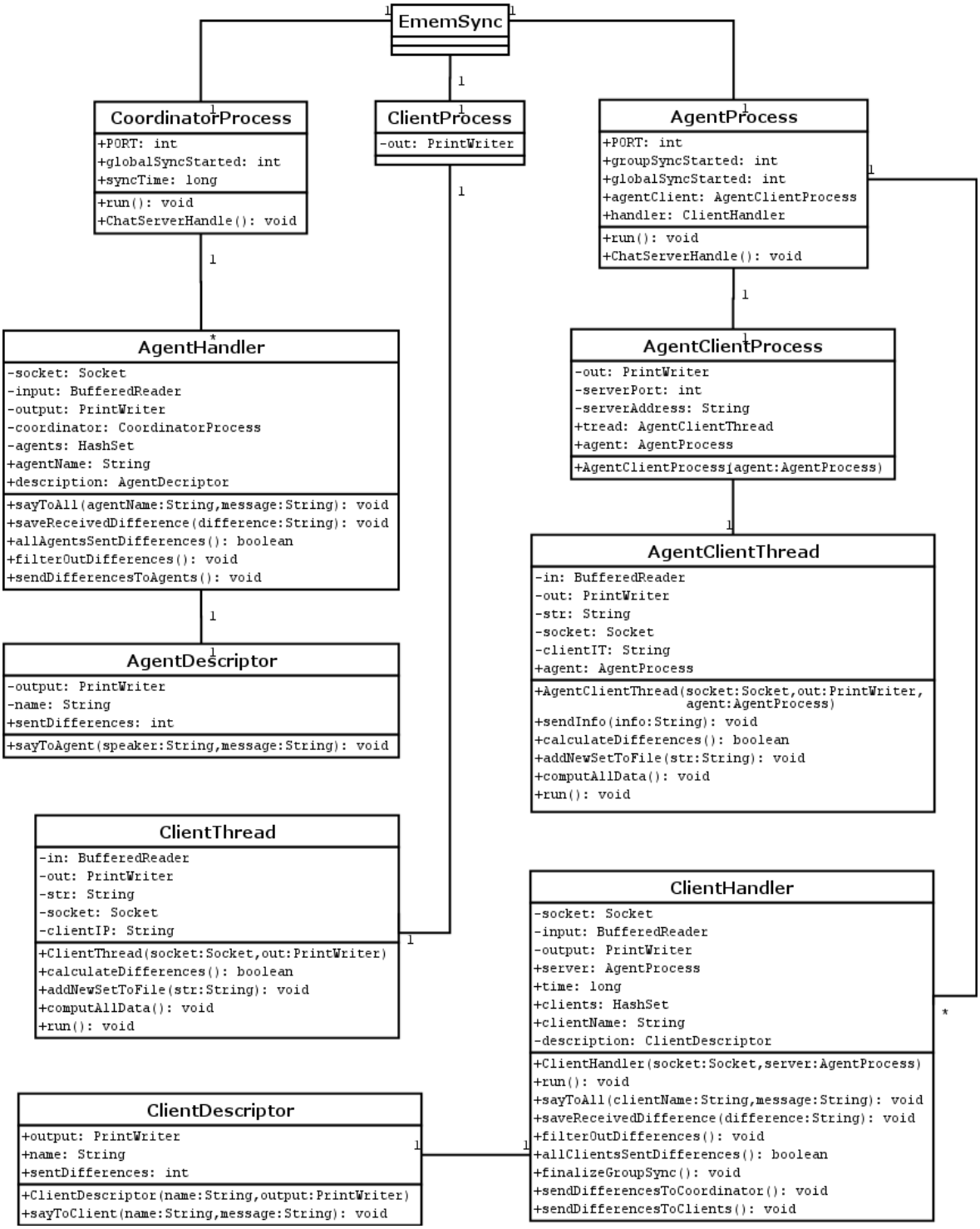

Figure 5.1: Implemented tool - UML class diagram

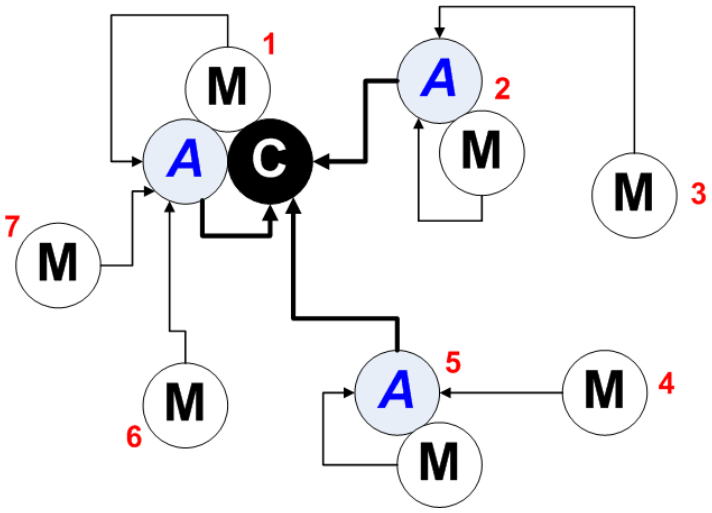

Figure 5.2: Implementation: Example scenario of 7 nodes connected in 3 groups. C - system coordinator, A - group agent, M - regular group member

ClientHandler.class The class (thread) responsible for handling all clients connected to the group agent communication processes,

ClientDescriptor.class The class creating objects maintaining each client's details,

The UML class diagram (Figure 5.1) shows the dependencies between all these Java classes presented above.

5.3 Node and its processes

The tool implemented during the course of this project provides one application which enables to run all types of nodes described in the design chapter. Each node can act as a regular group member, group agent and/or system coordinator.

According to the main assumptions each group agent can also be visible as a regular group member and can take part in each synchronization process. What is more, the system coordinator can run group agent process at the same time. Because of using Java Sockets technology in the implementation and to separate coordinator and agent threads both these processes listen to their clients on separate TCP ports.

Figure 5.2 shows an example scenario of network consisted of 7 nodes connected in three groups. Device 1 runs three processes: system coordinator, group agent and

Example of data file used in the implementation
information_information_information_000001::10000001::M information_information_information_000002::10000002::M information_information_information_000003::10000021::0 information_information_information_000004::10000031::N information_information_information_000005::10000041::0

Table 5.1: Example of data file format used in the implementation

regular group member. Other group agents are started on devices 2 and 5.

5.3.1 Data format

According to the requirements described in Chapter 4 the implementation of the proposed architecture for multi-party synchronization of data sets in a distributed environment assumes that all data sets are stored in a form of data base - it means that each information is stored in as a separate data record.

In the implementation provided in this project all data records are stored in a plain text file. Each record includes the following fields:

- information text
- a unique hash code of the record (integer)
- flag (M - for modified records, N - for new records, 0 - for not changed records)

All these fields are separated by double colon ('::'). Table 5.1 shows an example data file.

5.3.2 Message types

The following message types are used during the whole cycle of system synchronization of data sets process:

GET NAME - the message sent by a group agent to a new group member as a request for its name; similarly the system coordinator requests each newly connecting group agent about its name,

NAME name - the message sent as a reply for the request GET NAME; this kind of message is sent by the new group member to its group agent and by new group agent to the system coordinator,

GLOBALSYNC - the message request for starting the synchronization in the whole system, this kind of message is initially sent by group member to its group agent,

SYNCHRONIZE - when a group agent receives such message it sends to the system coordinator this kind of message,

GLOBALSTART - the message sent by the system coordinator as a reply for the `SYNCHRONIZE` request. This kind of message is send only when there is more than one group agent in the system (there are at least two groups of nodes) and the system synchronization process is not already started,

GROUPSTART - the message sent by the system coordinator as a reply for the `SYNCHRONIZE` request in case when there is only one group agent in the system - so only group synchronization is requested,

START - the message sent by group agents to all group members with the request for starting group synchronization process and preparing client's set of differences,

DIFFERENCES - the message sent by group members to their group agents during the process of group synchronization, this message includes also the whole set of differences prepared by group member,

GROUPDIFF - the message sent by group agents to the system coordinator after receiving group members' sets of differences and calculative the cumulative set of differences for the whole group - this message includes this cumulative set of data,

FINALDIFF - the message sent by the system coordinator to all group agents after preparing the cumulative set of differences for the whole system, this message includes this data set,

FINISHED - the message sent by group agents to the system coordinator after propagating the cumulative set of differences of the whole system to all group members,

END - the message sent by group member to the group agent or by the group agent to the system coordinator before leaving the system.

5.4 Set filtering

Message filtering in an operation of removing duplicate data records received from synchronizing users. During one synchronization cycle filtering is performed at both system coordinator and group agents levels.

Accurate and fast preparing the whole cumulative set of differences for group or the whole system has a big impact on the time of the whole synchronization process.

In the implementation worked out for this project filtering is performed by function `filterOutDifferences();` in `AgentHandler.class` and `ClientHandler.class`. Table 5.2 presents the sample code of this filtering function.

Thanks to the assumption that each data record includes additional meta data with a hash code unique for each row a `HashMap` object can be used. Each data record is added to the `HashMap` with the key of this mentioned hash code, so if there are a few records (duplicates) with the same hash code they are overwritten. The final `HashMap` is a set of unique data records.

5.5 Synchronization

After starting the system using the implemented tool each group member can trigger off the system for a group synchronization process.

```
void filterOutDifferences(){
     BufferedReader in;
     BufferedWriter out;
     Map differences = new HashMap();
     try{
        FileInputStream in_ = new FileInputStream("agent.dat");
        in = new BufferedReader(new InputStreamReader(in_,"Cp1250"));
        FileOutputStream out_ = new FileOutputStream("diffset.dat");
        out = new BufferedWriter(new OutputStreamWriter(out_,"Cp1250"));
        while(true){
             String s = in.readLine();
             if (s==null) break;
             String[] tmp = s.split("::");
             if(differences.containsKey(tmp[1])) continue;
             differences.put(tmp[1],s);
        }
        Collection values = differences.values();
        Iterator iterator = values.iterator();
        String s;
        while(iterator.hasNext()) {
             s = (String) iterator.next();
             out.write(s);
             out.newLine();
        }
        in.close();
        out.close();
     }catch (Exception e){
        System.out.println(e);
     }
}
```

Table 5.2: Set filtering function

As mentioned in the previous section there are two modes of synchronization: **global** synchronization and **group** synchronization. The first mode is run where there are at least two groups of nodes in the system i.e. when there are at least two active group agents, otherwise the group synchronization mode is performed.

To start the synchronization process one group member sends a message request `GLOBALSYNC` to its group agent. This agent, after checking if another set reconciliation process is not already started, sends to the system coordinator message `SYNCHRONIZE`. If there are more than 1 group agents (groups) in the system the system coordinator initializes the global system synchronization sending message `GLOBALSTART` to all group agents. Otherwise, only group synchronization in which the system coordinator does not take part is started (by sending message `GROUPSYNC`). When receiving

such message, group agents send to their group members message `START` initializing the group synchronization. Each group member prepares its set of differences (method `calculateDifferences();` in `ClientThread.class`) and forwards these differences as one message to its group agent. The group agents save these data sets to the file using method `saveReceivedDifferences` (`ClientHandler.class`) and after receiving differences from all group members they filter it using method `filterOutDifferences();` Such prepared data sets are forwarded to the system coordinator by using method `sendDifferencesToCoordinator();` or sent back to all group members if only group synchronization mode was started earlier.

The system coordinator works similarly to group agents. Each group differences set is saved to the file by method `saveDifferenceToFile();` (in `AgentHandler.class`) and after receiving information from all group agents filtering is performed (method `filterOutDifferences();` in `AgentHandler.class`). Finally, the whole cumulative set of differences for the system is sent back to all group agents by method `sendDifferencesToAgents();` (`AgentHandler.class`) using message `FINALDIFF`. All group agents forward this message to all group members and send back to the system coordinator acknowledgment message (`FINISHED`). The synchronization cycle terminates.

Figure 5.3 (UML activity diagram) presents the whole implemented synchronization algorithms.

5.6 Conclusion

The tool implemented and described in this chapter adapts only the main functions and a general scheme of the architecture for multi-party synchronization of data sets presented and analyzed in Chapter 4. All these simplifications done are connected with the adaptation of this implemented tool to perform number of tests examining the performance of the architecture, so manual settings and designating the system coordinator and individual group agents is introduced. Additional mechanism which

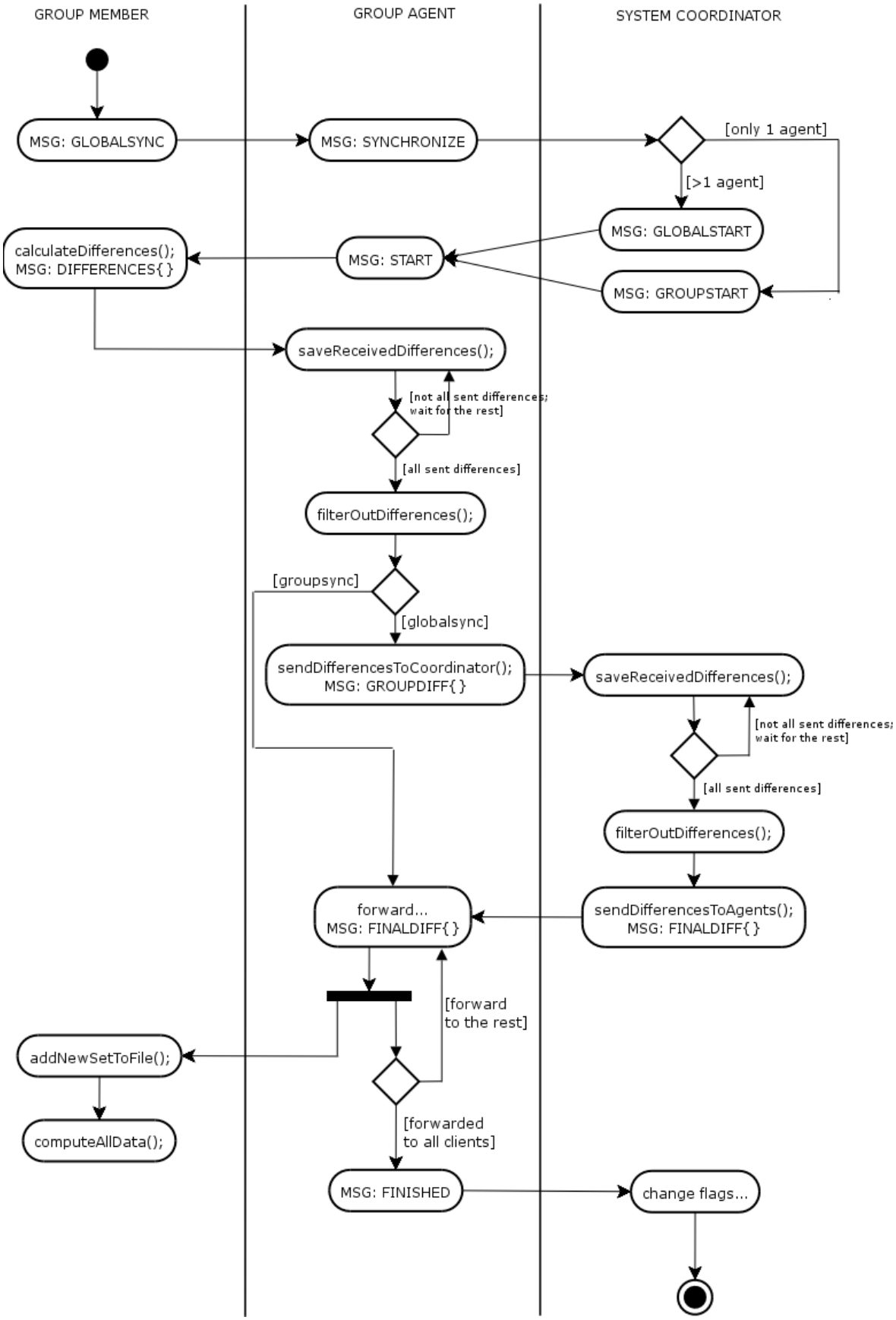

Figure 5.3: Implemented synchronization process - UML activity diagram

calculates the total time of data reconciliation process and number of messages sent are introduced as well.

Synchronization process presented in Chapter 4 was split into a few steps and implemented in the described tool (see Figure 5.3). To see how all these steps of set reconciliation process are served all results of each step are saved in separate text files. Such solution, in some specific situation, can have a negative impact on the performance, especially when a lot of data sets have to be written and then read instead of processing in the memory ('on the fly').

To show all the benefits of the proposed architecture for networked set reconciliation a lot of methods should be developed and added to the existing version of the implementation. The 'core' was implemented in Java using Java Sockets according to the design presented in the previous chapter work very well and can be a good starting point for the future development.

Chapter 6

Evaluation

This chapter presents results of experiments performed with the tool described in Chapter 5. A set of various parameters and scenarios was established to achieve these results. This outcome is presented on various graphs. Detailed analysis of achieved results is provided.

6.1 Experimental setup

Because of the difficulties with simulating a natural distributed environment for which proposed architecture was designed the experiments were performed on two laptop machines connected via network interface cards. Thanks to the implementation in Java various number of thread could be established to provide different settings and scenarios for these experiments.

Parameters of the machines used in simulation were following:

Acer TravelMate	**Dell Latitude D400**
- Intel Pentium M 740 2 MB L2 cache, 1,73 GHz, 533 Mhz FSB - 512 MB RAM DDR - NVIDIA GeForce Go 6200 128MB - 60 GB HDD - MS Windows XP SP2	- Intel Pentium M 1,3 GHz - 256 MB RAM - 40 GB HDD - MS Windows XP SP2t

Table 6.1: Experimental setup

6.2 Testing metodology

To show the performance and usefulness of the architecture for data reconciliation worked out during the course of this project a number of tests in various scenarios were carried out.

In the set reconciliation process there are a lot of factors which are important in case of efficiency meaning. Because another architecture designed for simultaneous, networked set reconciliation was not found even in the literature theory there were no possibility to compare achieved results to another solution. In connection with this, the performance of the architecture proposed in this work was compared on the basis of applying three various synchronization algorithms in the this architecture:

fast-sync mode In this mode group all group members sent to the group agents only modified differences. Agents and when preparing the set of differences for their groups filter out all the duplicates an such set of differences is sent to the system coordinator,

filtered slow-sync In filtered slow-sync mode group members send their whole data set (not only differences), but group agents still filter out all the duplicates when preparing the cumulative set for the group,

pure slow-sync Pure slow-sync mode assumes that all group members sent their whole data sets and neither group agents nor the system coordinator do not filter duplicates.

The experiments set up in various scenarios (the number of running processes and the data sets they store varies) provide two types of measurements important for the sake of its potential use in the network environment:

Number of messages These measurements deliver information about number of messages required for the communication of the participating nodes during the whole process of set reconciliation (including message requests etc.),

Time Tests measuring time provide information about time required to perform the whole process of synchronization (including transferring the data through the network, filtering out the duplicates, etc.) in various settings.

All the experiments were performed on two machines. On each machines various number of separate processes (it means that each process stored separate, independent data set file) depending on the scenario prepared for the specific test was run.

6.3 Topologies simulated

To show the results presenting various aspects of the architecture performance a couple of experiments in different scenarios was processed.

First of all, the experiment of the synchronization process was perform for the scenario where there is only one group consisted of 5 nodes. All these nodes stored data sets with exactly the same number of new records and exactly the same number of common data rows, increasing od decreasing depending on the specific measurement point.

Secondly, the experiment measuring total synchronization time depending on the number of nodes connected nodes in one group was done. In these tests each node stored exactly the same number of common rows and also newly added information.

The next test was based on the number of group in which the constant number of connected nodes was split. This experiment measured both factors - number of messages sent during the whole synchronization process and the total time required to process.

Detailed results of all these experiments and their interpretation are presented in the next section.

6.4 Analysis of the results

First three experiments performed measured the total time required for the whole set reconciliation process.

6.4.1 Time measurements

1 group, 5 nodes, increasing number of differences, decreasing number of common rows

In this experiment the system consisted of one group of 5 nodes was simulated. In this case, where there was only one group the system coordinator did not have to take part in the set reconciliation process - it just agreed with message request sending to this only one group agent an information about group synchronization.

Each node stored a data set of 5000 records. For each measurement point the number of new records increased and number of common records decreased.

Figure 6.1 shows the results of this experiment for the number of differences more than 500 in the whole data set stored by each group member taking part in the synchronization process.

The results confirm the choice of fast-sync mode for the synchronization in contrary to the both modes of slow-sync. What is more, it is shown that semi-centralization of the system provides the feature which minimizes the time required for the synchronization process in case of slow-sync mode use - the difference between pure slow-sync and the filtered slow-sync gained thanks to this feature is obvious.

Moreover, the distance between fast-sync and slow-sync line shows that even for the 'worst' point when the number of differences is really high in the relation to the whole set the fast-sync mode is more efficient.

Figure 6.2 shows similar measurements in scenario where the number of differences was less than 500 records.

These results show that for such settings when the number of differences is low the time required for performing synchronization process in fast-sync mode is symbolic and

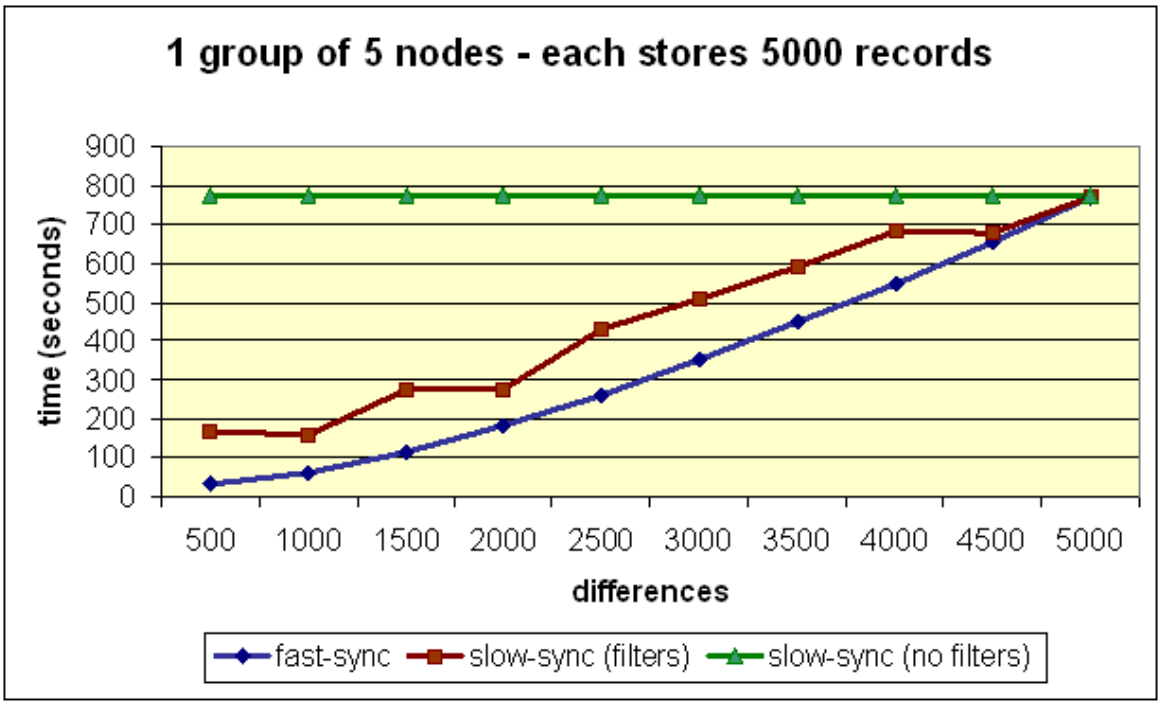

Figure 6.1: Overall synchronization time in the system consisted of 1 group of 5 nodes with 5000 records each - experiment A

represents only the cost of transferring data through the network.

What is more the filtered slow-sync line is very similar and the distance to the fast-sync line is almost constant. It means that the difference of the time required to complete synchronization in slow-sync mode is consumed by the process of filtering out duplicates and not changed records. There is a high probability that these results could be much better after improving filtering functions.

1 group, increasing number of nodes, 1000 differences

The next test including a couple of measurements was performed to check how the time of system synchronization varies depending on the number of participation nodes.

The number of nodes connected to one group agents varied from 2 to 10 - each with the data set of 5000 records and with 1000 new records included.

The results of this experiments are presented by Figure 6.3. It is obvious that the slow-sync mode is much slower than fast-sync, but observing this graph there could be a conclusion drawn that the differences between these two modes are bigger the more

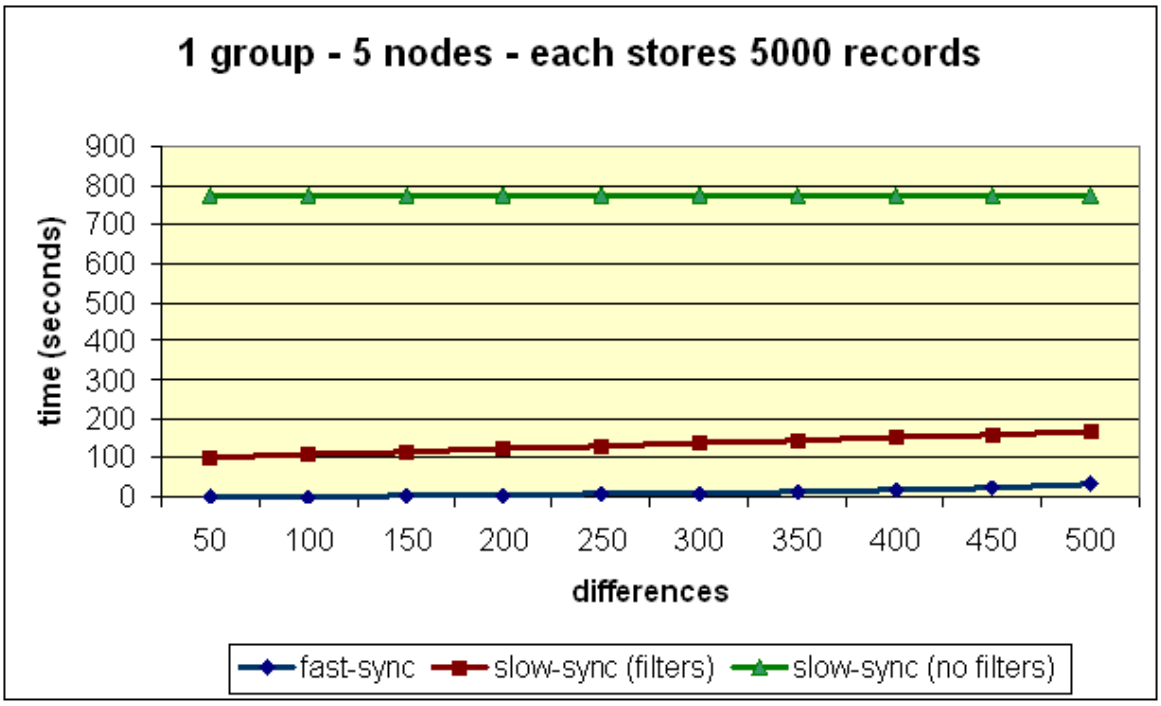

Figure 6.2: Overall synchronization time in the system consisted of 1 group of 5 nodes with 5000 records each - experiment B

nodes are connected to the system. It means that use of the slow-sync mode should be avoided especially when the number of differences is high. Finally, the role of filtering and processing data by group agent is more distinct here, where the difference in time between both tested synchronization mode is so high.

10 nodes, increasing number of groups, 50 differences

The aim of experiments performed using the implemented tool was to show various views on the architecture designed in Chapter 4. The following test examined the behaviour of the synchronization system depending on the number of groups (started group agents) in which the constant number of nodes where split.

The measurement provide intresting results - it is better to create less bigger groups than more smaller groups. What is more, for the first case, when connected nodes are organized in bigger groups the total time of the synchronization can be even two time shorten then in other scenarios. What is more, the worst result was achieved in the system with couple of group with the same size. Detailed results are presented on

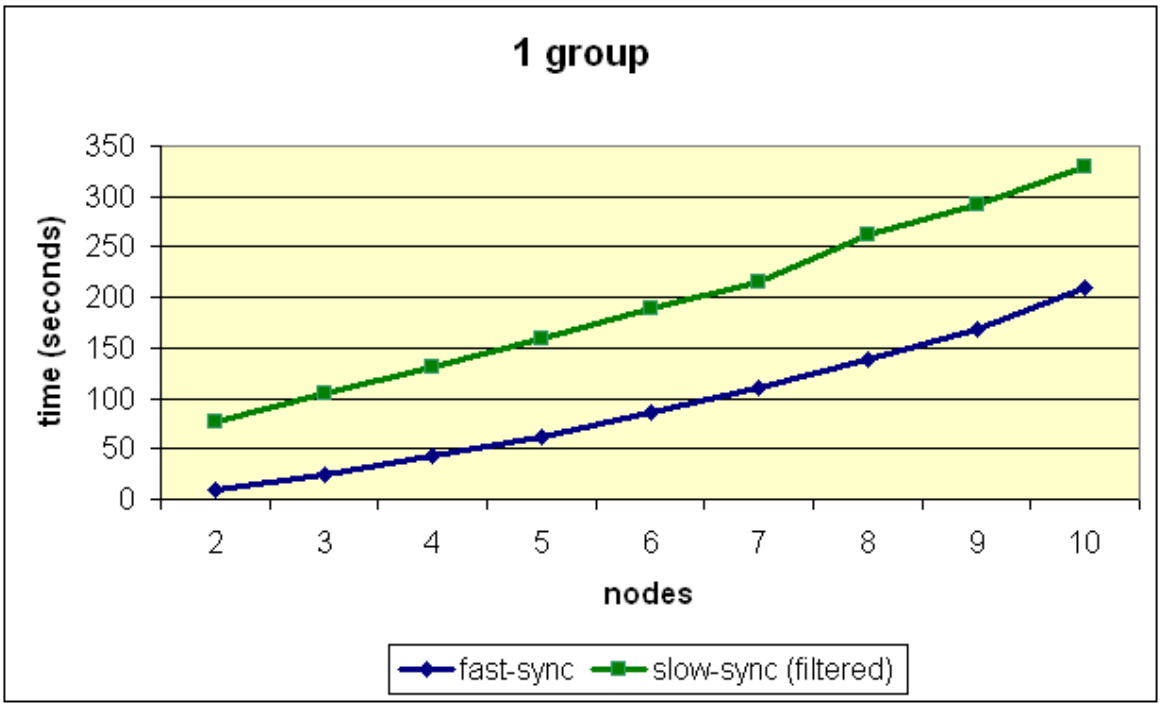

Figure 6.3: Overall synchronization time depending on the number of nodes in one group (2-10) with 5000 records and 1000 differences each - experiment

Figure 6.4.

These results encouraged to perform similar measurements testing the number of messages required to be sent during the whole set reconciliation process.

6.4.2 Messages

The second part of experiments done using the implemented tool was performed to check various scenarios' influence on the number of messages sent during the whole synchronization process. This factor (number of messages) is very important in contemporary distributed systems.

The experiments measuring number of messages sent were performed in exactly the same scenarios that previous time examines were done. Thanks to such solution, various depenencied can be found.

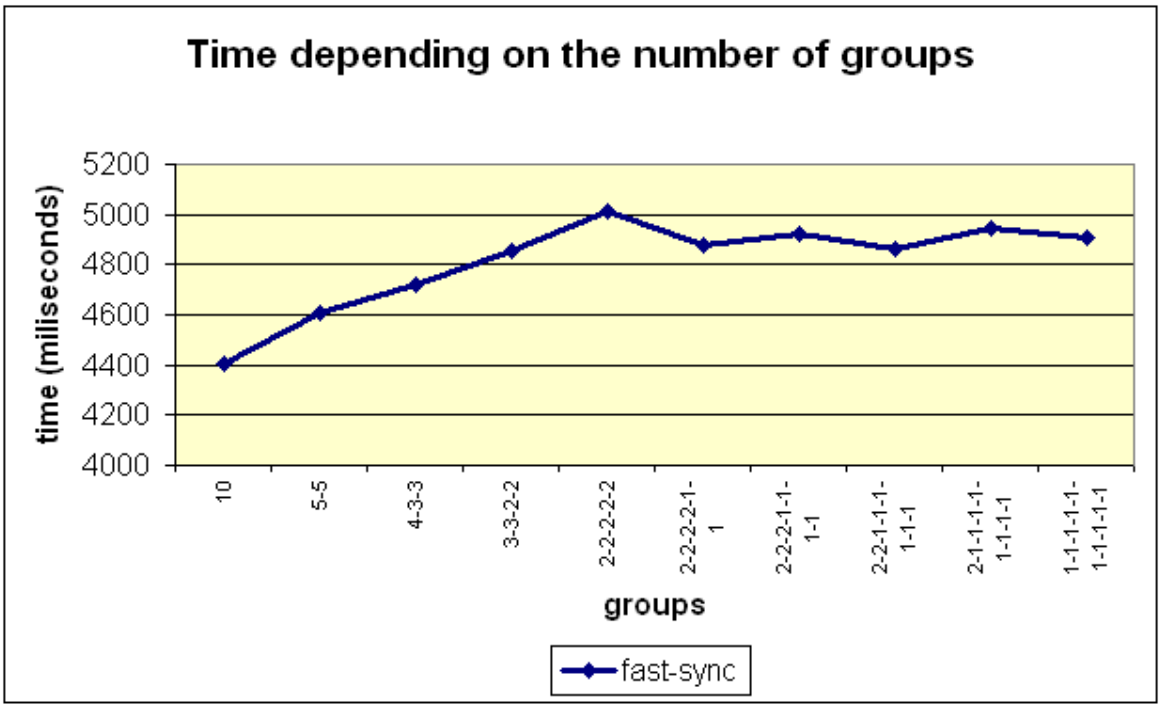

Figure 6.4: Overall synchronization time depending on the number of groups. 10 - one group of 10 nodes, 5-5 - two groups for 5 nodes each, 4-3-3 - groups of 4,3 and 3 nodes etc.

1 group, increasing number of nodes, 1000 differences

In the first test the number of messages sent was checked for the system with only one group agent with the increasing number of connected nodes (from 2 to 10). In this specific case, where there was only one group agent, the system coordinator did not take part in the set reconciliation process.

Figure 6.5 shows exact results of the test performed in the scenario described above.

The results of this test seem to be very promising, because the cost of communication between regular group member and its group agent is only about 3 messages per node. The minimal cost of communication between group agents and the system coordinator (including a message request forwarded to the system coordinator from group member and final acknowledgement - the system coordinator does not take part in the synchronization process but it agrees with the request and receives final acknowledgement) is only 7 messages.

There is no difference between fast-sync, filtered slow-sync and slow-sync modes,

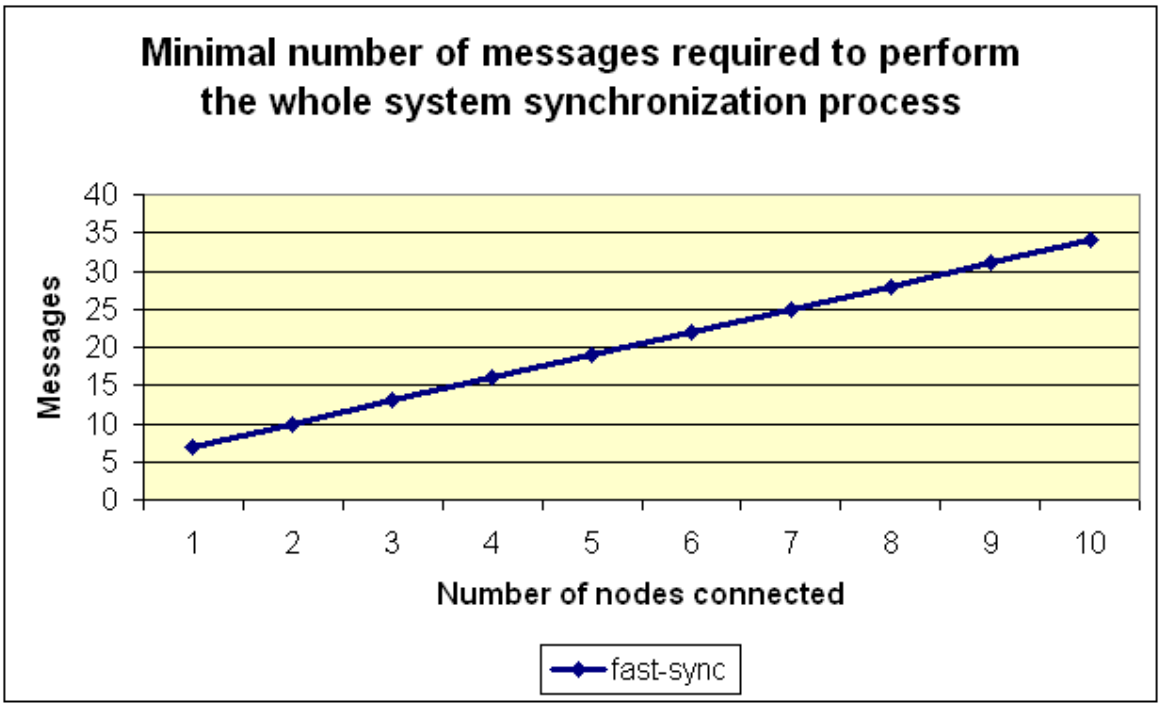

Figure 6.5: Minimal number of messages required to perform the whole synchronization process depending on the number of nodes connected to the system

because in this case the number of messages sent in all modes is exactly the same - the whole data sets are sent as one message each.

10 nodes, increasing number of groups, 50 differences

The last experiment shows the performance results depending on the number of groups which are created by 10 nodes connected to the system. Similar test was done checking the synchronization time for this scenario.

The resuls of both experiments are also similar. It means that the smallest number of messages to perform set reconciliation is generated when the nodes connected to the system are split in small number of bigger groups.

Figure 6.6 shows the line of results generated during this experiment.

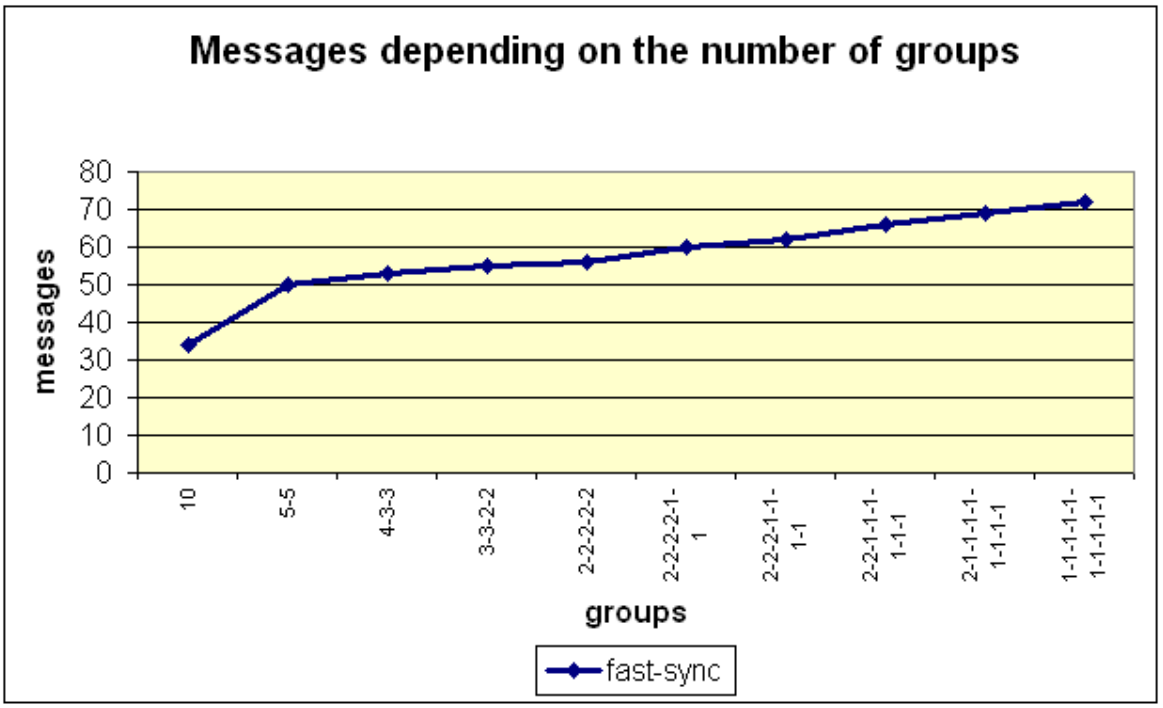

Figure 6.6: Number of messages sent to perform the whole synchronization process depending on the number of groups which are created by 10 nodes connected. 10 - one group of 10 nodes, 5-5 - two groups for 5 nodes each, 4-3-3 - groups of 4,3 and 3 nodes etc.

6.5 Conclusion

The implemented tool was used to perform a number of various tests to examine the performance of the achitecture designed and presented in Chapter 4 of this dissertation report.

Since it is very difficult to simulate a wide distributed environement for which this architecture is dedicated, the tests were performed on desktop PCs by running a number of threads on each machine. Two most important (for distributed architecture) factors were examined and analyzed by number of tests and measurements.

First of all, the total synchronization time in various scenarios and settings was measured and studied. These tests showed that that decision of semi-centralization was appropriate, because thanks to this centralized set filtering could be applied, what is a big feature of this architecure - fast-sync mode can be used as a main synchronization protocol. What is more, time measurements presented a way how groups should be

created. The best results were achieved for the scenario in which connected nodes were split in small number of bigger groups.

The second factor with high importance which was examined was the number of messages required to be generated to perform the whole system set reconciliation. These tests proved that the number of messages sent by participating users in communication processes is minimal for this architecture.

All these experiments performed and evaluated showed that multi-agent semi-centralized architecture presented in Chapter 4 has a significant potential to be fast and efficient for synchronization of data sets in a real distributed environment.

Chapter 7

Conclusions

7.1 General conclusions

Today there are more and more various solutions for data reconciliation. Most of these solutions adapts algorithms (or their modifications or combinations) presented in this dissertation project. Since most algorithms analyzed in Chapter 2 are very fast in pair synchronization of data sets their performance in a networked scenario can fall rapidly. Almost every algorithm is designed for synchronization between only two, the same, devices. This assumption very often requires to store additional information when adopted to networked environment (e.g. to store information about other devices to synchronize always in the fastest mode).

One of the ways of message dissemination in a network analyzed during the course of this project were gossip protocols. Although gossiping seems to be a powerful way of information spreading between number of machines adoption of such algorithms turned out to be be extremely difficult and not effective. Algorithms designed for interconnection networks such as the complete graph, hypercube, cycle etc. require to maintain stable connections between nodes what could be impossible to realize in the scenario where nodes join and leave the network intermittently. Similarly, finding an optimal solution for unstructured network was not so easy in spite of number of researches found in the literature.

The architecture of synchronization of data sets in a distributed environment developed during the course of this project shows a significant potential to be fast and efficient for simultaneous set reconciliation between more than two nodes connected in a dynamic network where nodes can join and leave intermittently. What is more, although the cost of semi-centralization of the system seems to be high, the proposed system provides a possibility of parallel synchronization of high number of participating devices what could be recognized as a kind of innovation in this area. Finally, the aforesaid semi-centralization ensures that so simple algorithm such as fast-sync mode of Palm HotSync can work very effectively.

7.2 Other possible scenarios

As presented in the previous parts of this dissertation report also other possibilities, algorithms and network structures than this used in building the architecture presented in Chapter 4 were considered.

There is still a potential possibility to build a networked data synchronization system based on simultaneous, continuous set reconciliation using peer-to-peer network model where tasks processing is not so centralized. Further research, especially in the area of simultaneous peer contribution and message dissemination could be performed.

What is more, gossip protocols seem to be a solution for message spreading in such networks, but these algorithms are still a subject of many research, so in the future some nearly optimal methods, which could be applied in such scenario, can appear.

Combining peer-to-peer network model with a two-way gossip protocol which ensures message dissemination between all participating peers in a short time could have a very positive impact on the performance of such system. What is more, problems connected with system centralization and organization could disappear, what finally could make synchronization in such system much faster.

But the lack of control on message flow in such scenario can also block some parts of the system and delay the whole process. All the aspects of such solution (also various

types of synchronization protocols) should be analyzed and compared, to find the best way of building such system working in peer-to-peer manner.

7.3 Objectives fulfilled

The following objectives were stated at the beginning of this work:

Review and analysis of existing data synchronization methods. In Chapter 2 a set of chosen data reconciliation protocols was presented. Their advantages and disadvantages concerning complexity, efficiency, usability and suitability for the network environment were analyzed, compared and considered. What is more, two industrial projects were also presented. Finally, an overall balance sheet for all these methods was sketched.

Investigation various types of distributed systems. Two most common distributed technologies were analyzed in Chapter 2 as well. Their similarities and differences were characterized and their suitability for potential distributed data synchronization environment was evaluated.

Analysis and comparison of various gossip protocols. Chapter 3 provides a lot of information about gossip protocols and their potential use in various network systems. A detailed analysis based on the proposed model including various factors such as algorithm complexity or suitability for dynamic network was formulated and presented. What is more, two projects built on basis of gossip algorithms were described in short.

Design of the multi-party data sets synchronization system. Architecture for multi-party synchronization of data sets in a distributed environment was presented and described in Chapter 4. Based on the previous analysis of synchronization algorithms, distributed systems and gossip protocols a few decisions were made to provide a possibility of the fast data synchronization in a network environment where nodes can join and leave the system intermittently.

Implementation of the proposed system. Architecture described in Chapter 4 was finally adopted in practice and implemented in Java language.

Number of experiments with the implemented tool. An evaluation of the implemented tool described in Chapter 4 and 5 was realized by the number of experiments which are described and analyzed in Chapter 5. In this chapter a legible presentation of experiments' results is shown.

Consideration on another possible scenarios. Section 7.2 of this Chapter presents short considerations on other possible scenarios for data synchronization in peer-to-peer network model. Difficulties connected with applying known algorithms in this scenario are considered and analyzed briefly.

Determining goals for the future work. Section 7.4 of this Chapter determines goals for the future work.

7.4 Future work

Although the detailed investigation of various data synchronization algorithms and gossip schemes was done and a lot of problems connected with the proposed architecture were analyzed and finally addressed a few potential goals for the future work were outlined:

Handling deleted records. The proposed and tested architecture handles very well with the information which were modified and inserted to the system since the last synchronization process. The problem with information which were deleted on same devices is not so simple - there should be applied some mechanisms which can provide a possibility to decide whether delete some data records from all machines or just treat some messages as a deleted by mistake. This problems should be analyzed in details and a solution addressing this problem should be designed and applied to ensure comprehensive service of data synchronization,

Decentralization. Although problems connected with centralization of the proposed architecture are handled and solved (e.g. by secondary system coordinator use) some degree of decentralization is still desired. The possible way of this work could be further analysis of gossip or similar schemes and applying such solution to disseminate message without any central server participation,

Applying other algorithms. The choice of algorithms applied in the proposed architecture for distributed data synchronization was mainly based on the analysis presented in the Chapter 2. Applying other algorithms or combination of them could show some other results of the architecture performance and reliability,

Further testing. The tests performed with the implementation of the proposed architecture show significant potential to be fast and efficient for synchronization of data sets in a distributed environment. Since all the tests were performed only on two machines with various settings and different number of processes ran on each machine the further testing in a real distributed environment such as Internet could show other features of the architecture developed in this dissertation project,

Security. Security is one of the most important factor of our existence and cooperation in networked environment. The proposed architecture currently does not provide any mechanisms ensuring sent data sets safety. All messages and data sets are sent to other users in a plain text form. The future work should focus on applying various encrypting mechanisms and/or authentication schemes. Some possible solutions are described in Appendix A.

Bibliography

[1] Starobinski D., Trachtenberg A., Argawal S., *Efficient PDA Synchronization,* Department of Electrical and Computer Engineering at Boston University.

[2] Argawal S., Starobinski D., Trachtenberg A., *On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices,* Department of Electrical and Computer Engineering at Boston University.

[3] Trachtenberg A., Starobinski D., Argawal S., *Fast PDA Synchronization Using Characteristic Polynomial Interpolation,* Department of Electrical and Computer Engineering at Boston University.

[4] SyncML Groups *Building an industry-wide mobile data synchronization protocols, SyncML White Paper* http://www.syncml.org.

[5] Byung-Yun Lee, Tae-Wan Kim, Dae-Woong Kim, Hoon Choi, *Data Synchronization Protocol in Mobile Computing Environment Using SyncML,* Electronics and Telecommunications Research Institute, Deparment of Computer Engineering - Chungnam National University, Korea.

[6] Intellisync Corporation, *Intellisync Data Sync Product Functionality Paper,* http://www.intellisync.com.

[7] Intellisync Corporation, *Intellisync Mobile Suite 6 Technical Datasheet,* http://www.intellisync.com.

[8] Milojicic D. S., Kalogeraki V., Lukose R., Nagaraja1 K., Pruyne J., Richard B., Rollins S. ,Xu A., *Peer-to-Peer computing,* HP Laboratories Palo Alto, March 2002.

[9] *Palm OS 68K SDK Documentation,* http://www.palmos.com/dev/support/docs/.

[10] Schilit B. N., Sengupta U., *Device Ensembles,* Computer, vol. 37, no. 12, pp. 56-64, December 2004.

[11] Schutt T., Schnitke F., Reinefeld A., *Efficient Synchronization of Replicated Data in Ditributed Systems,* Zuse Institute Berlin

[12] Tridgell A., *Efficient Algorithms for Sorting and Synchronization,* PhD thesis, The Australian National University, April 2000.

[13] Brodery A., Mitzenmacherz M., *Network Applications of Bloom Filters: A Survey*, Division of Engineering and Applied Sciences, Harvard University.

[14] *Napster home page,* http://www.napster.com/.

[15] *Gnutella protocol development website,* http://rfc-gnutella.sf.net/.

[16] Izal M., Urvoy-Keller G., Biersack E.W., Felber P.A., Al Hamra A., Garces-Erice L., *Dissecting BitTorrent: five months in a torrent's lifetime,* Institute Eurecom, France.

[17] *BitTorrent home page,* http://www.bittorrent.com/.

[18] Hromkovic J., Klasing R., Monien B., Peine R., *Dissemination of information in interconnection networks (broadcasting and gossping),* Department of Mathematics and Computer Science, University of Paderborn.

[19] Hromkovic J., Jeschke C.D., Monien B., *Optimal algorithms for dissemination of information in some interconnection networks,* Department of Mathematics and Computer Science, University of Paderborn.

[20] Krumme W. D., *Fast gossiping for the Hypercube,* Department of Computer Science, Tufts University, Medford.

[21] Cohen J., *Broadcasting, Multicasting and Gossiping in Trees under the All-Port Line Model,* Universite Paris Sud, France, 1998.

[22] Kempe D., Kleinberg J., Demers A., *Spatial Gossip and resource location protocols,* Cornell University, New York.

[23] Birman K. P., van Renesse R., *Scalable Data Fusion Using Astrolabe,* Cornell University and Reliable Network Solutions Inc., New York.

[24] Gupta A., Mohan A., Sundararaj A., *Fast resource dissemination in a P2P network* , Northwestern University, 2003.

[25] Zhenjie Z., Feng Y., Xiaoyan Y., *Report on unstructured network in peer-to-peer system* .

[26] Cuenca-Acuna F. M., Martin R. P., Nguyen. T. D., *PlanetP: Using Gossiping and Random Replication to Support Reliable Peer-to-Peer Content Search and Retrieval*, Department of Computer Science, Rutgers University.

Appendix A

Short security considerations

Introduction

When considering multi-party synchronization of data sets the security issues connected with data accessing and transferring from one point to the second one etc. should be considered and analyzed.

What is more, when applying any algorithm to a network infrastructure we should remember about various vulnerabilities which can be met in such environment. There is no problem when a network is a separated one and when all hosts have equal rights to access each other. But if there are other hosts which should not have access to some data on other hosts and should not have right to take part in data synchronization we should consider applying various methods to improve security of our data.

Since there are number of vulnerabilities in networks there should be improved at least two the most important mechanisms to make the whole system safer. First of all, applying some authentication/identification methods to improve safety of our data and to have control who can have access and who can take part in data synchronization should be considered. The second important thing is how this data should be transmitted between hosts which take part in data synchronization process. So some cryptography methods should be applied to make data safer.

In this short consideration various methods of authentication and cryptography are

briefly analyzed to show what solutions could be implemented in the project. At the end other problems which can affect security of the system are mentioned.

Authentication

A very important thing is to know who can access the data and who can take part in information exchanging process. There are various methods which can be used to identify users.

First of all it should be considered how the users who can take part in synchronization process are identified. There can be used simple methods such as login/password schemes - passwords which are provided earlier for these nodes. These passwords should also be encrypted and stored in a secure place - not just in a plain text. So here various message digesting algorithms such as MD4, MD5 or SHA-1 can be implemented. These methods generate a unique hash string on every text. In this form passwords can be stored on nodes. Users can be obligated to present login/password every time they want to synchronize.

The second problem is how the users should be authorized in a secure way. So, passwords can be checked a few times during synchronization (to eliminate a case when somebody guessed the password or password has changed).

In such case applying some authentication system - for example Kerberos can be an effective solution which will ensure that only entitled users can take part in the data exchanging process. But such a solution could work only in a system with centralized servers (Kerberos requires a server which has to be online to everyone almost always). There are not any good, well-tested and secure authentication schemes for peer-to-peer infrastructure.

Cryptography

Regardless of the structure in which data sets are stored by each network node implementation of some chosen encryption methods could be necessary - especially in the scenario where a lot of information sets are transmitted between a few hosts located in an unsecured network.

There are many various algorithms which could be used to encrypt this data - of course the choice depends on the type of this data (how long is it, how important it is etc). So we can choose between secret-key and public-key algorithms or message digesting. A short analysis of chosen encryption methods is presented below. Their differences, advantages and disadvantages are described in a short.

1. Cryptography with secret-key (for example: DES, IDEA, RC4, RC5, AES)
2. Cryptography with public-key (for example: RSA, Diffier-Hellman)
3. Message digesting - hashing (MD4, MD5, SHA-1)

Cryptography algorithms can also be split into two groups: **synchronous algorithms** where encryption and decryption processes require the same key and **asynchronous algorithms** - where two different keys are used, one in encryption operations and the second one is used to decrypt messages. In this case, where data will be transmitted between hosts to synchronize, hashing algorithms may be not appropriate, because these are one-way methods - it means that hash message cannot be decrypted to the plain text. These methods can be used to compare various messages, files etc., but not to send data to other host, because this data can be useless.

DES. Data Encryption Standard - this method uses 56-bit long key and input and output block are 64-bit long. The key length is not so good solution, because there are algorithms which uses longer keys (more secure), but this algorithm is a very complex one (users special encryption block and a special mechanism (number of XORs and loops)). This method is also complicated in implementation, but is very reliable.

3DES. Tripe Data Encryption Standard - it is a kind of mutation of standard DES which applies DES three times (with different key, but K1 can be equal K3). Thanks to it the key is 112-bit long and input/output is 64-bit long. This method is much better that the previous one, but the problems are still the same.

IDEA. International Data Encryption Standard - in this method key is 128-bit long (so it is very good!) and input and output are 64-bit long. The mechanism is much simpler than in DES and this method is much faster - especially in software use.

RSA. This algorithm is used for example in banking. Its difficulties are based on big number which are difficult to factorize

The algorithms mentioned above are the most popular ones. Some of them are used nowadays. For example RSA is used in banking to provide secure money transactions. We should also remember that encryption algorithms which use public-key are not so sufficient when we want to transmit a lot of data (they are so complicated so the whole process can be much slower). So, we should consider applying algorithms which use secret keys - such as DES (3DES) or IDEA - these will be probably the best as a solution for this problem. Although the implementation these algorithms can take a lot of time the effects can be good.

Other problems connected with security

There are also other problems which we can experience in network environment. The two most popular are DoS attacks and buffer overflows. So, first of all we should create this software for synchronization to eliminate DoS attack, which can appear when hundreds or thousands of users will try to ask to synchronize at the same time - in this way the whole mechanism can be blocked a no one can synchronize. This problem can be solved by using some methods to eliminate high number of participants in this process - for example by introducing some limits. The second problem and the most

common in today's networks is buffer overflow. This problem is usually caused by bad design or implementation of the software, so security issues should be considered at the software designing level. During implementation process a lot of tests with various settings of data should be performed to eliminate buffer overflow problem if it appears.

Conclusion

There are many methods which can be applied to improve security of the computer system, but it is obvious that it is almost impossible to eliminate the risk at all - it just can be restricted.

In the environment where a lot of various data sets are transmitted between more than one computer (in a network) security issues should be considered at the design level. Various methods should be applied to improve the security and restrict non-entitled nodes to take part in the process.

As discussed above applying some methods of authentication, authorization or message encryption can be very effective and can limit the risk. What is more, problems like buffer overflow can be eliminated at the software design level.