CHALMERS

A Mobile Unit Synchronization Algorithm

A Partial Database Synchronization Scheme between a Centralized Server and Mobile Units

Master of Science Thesis in the Programme Network and Distributed Systems and Software Engineering and Technologies

ERIK HAMMARBERG
THOMAS GUSTAFSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Göteborg, Sweden, March 2011

A Mobile Unit Synchronization Algorithm
A Partial Database Synchronization Scheme between a Centralized Server and Mobile Units

Erik Hammarberg
Thomas Gustafsson

Examiner: Arne Dahlberg

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden March 2011

Abstract

This report covers the development of a new synchronization module for the Jetas Quality Systems AB. The developers at the company did not appreciate the existing module, based on Microsoft Synchronization Framework and thus this project was launched. The project first conducted a literature study to try to find a previously developed model that could be adapted to the Jetas system. This, unfortunately, was unsuccessful and the project continued with the design of a conceptual model for a synchronization module based on the Jetas systems characteristics.

This model was implemented and enhanced iteratively and functionality added according to the prioritizations done in the requirements specification. The synchronization algorithm itself is an optimistic timestamp based method. It utilizes the synchronization primitives that are already present in the Jetas database and used by other parts of the system as well. The selection of which data to synchronize, so as to only do incremental updates each synchronization, is done by keeping track of the timestamps of the last time a synchronization was performed by that PDA while each database row has a timestamp of when it was last changed. Conflict detection is performed by comparing the timestamps of uploaded and server version of the DataRow in question.

The module developed was tested in an office environment, but in this environment it showed a slight improvement in performance even while database indexes are optimized for the previous synchronization module and several optimizations can be done before this prototype is finalized for a production environment. In addition to this, the new synchronization module can easily be extended to implement new functionality, such as multiple users per PDA.

Table of content

1 Introduction

This report will cover the design of a new synchronization module for Jetas Quality System developed by Jetas Quality Systems AB. The Jetas Quality System is a web based service for scheduling and work order management for property maintenance. Administration of the service is web based via an ASP.NET environment and technicians in the field use handheld PDAs with barcode readers to receive and report work orders and details about components. It is also possible to send fault reports from the field back to the system.

The general architecture of the system is an optimistic client-server oriented architecture where the server provides work orders for the clients out in the field, which performs them and reports back. The most interesting part and the subject for this report is the synchronization between the handheld clients and the server, how to handle different conflicts that can occur and how to keep track of the data changes and to minimize the network traffic.

The synchronization used earlier in Jetas Quality System use Microsoft Sync Framework but since this is a closed framework it only provides limited possibilities to modify details in the synchronization module, which has proven to be burden for the developers at Jetas.

Both the server and the client are developed in C#. The server provides an administrative web service for desktop users, and provides services to the PDAs through Windows Communication Foundation –web services. Security is provided by SSL encryption of the web traffic and a ticket based authentication scheme.

This report will describe the way that the synchronization is performed and the considerations that were made to make this possible. Some of these considerations stem from the way that the Jetas Quality System is designed and, especially, how its database is designed and work. This information is not something that the Jetas Corporation wishes to make public and it is thus not included in this report. Similarly, the data model developed for transfer of data between server and PDA is conceptually explained but the details of this model are not disclosed.

1.1 Purpose and objective

The existing synchronizations module in the Jetas Quality Systems, which is based on Microsoft Synchronization Framework, has proven to be inadequate for the needs of the developers and customers at Jetas. The goal, therefore, of this project is to propose a new synchronization scheme for the Jetas Quality Systems.

To be able to test and evaluate the synchronization scheme developed in this project, a prototype should also be developed. This prototype should be developed to work with the existing System and, on the server side, also work alongside the previous synchronization system since one cannot assume that all customers can upgrade all their PDAs to a new version of the client software at the same time.

Finally, as the market for smartphones and other handhelds looks today, it is very possible that Jetas within a near future could decide to port their application to an additional platform beside Windows Mobile 6.x. A guideline for the project is thus that the server solution should not assume that the client necessarily is a Windows Mobile client and that the possibility of developing clients for another platform be left open.

1.2 Methodology

The project began with a planning phase where the main lines of the project were drawn up and the major phases and deadlines were planned and set. Discussions were made with Jetas about the time frame and preliminary requirements and delimitations. The outcome of this phase was the planning report.

After this the project continued with a preliminary literature study into related topics such as version handling, database replication and database synchronization and related research. This phase also included a study of the existing Jetas Quality System and the previous synchronization scheme. In parallel with this, the functionality and requirements of the application was explored via experiments with the old application, discussions with the developers at Jetas. Use cases for the application was developed and a requirement specification was produced from this and more discussions.

After the requirement specification was set, the project entered the design phase where the design of the algorithm as well as the design of the data structures and other data type that was needed to implement a solution, was to be set. In parallel with this work, several different small test programs were built to evaluate different ways of sending, receiving and packaging data in the target environment as well as to test whether certain ideas were possible or not.

The project then entered the implementation phase, where the design was slowly realized piece by piece in incremental steps. Each piece of the program was first built into a barely working version to prove that the design idea could work, which was tested to ensure that it performed what it was supposed to. When this was validated, things that did not work as intended was corrected and the piece was improved to perform better and more reliable. In this way, all the components that was marked as required in the specification was developed and then the entire project was tested and evaluated. After this evaluation, the entire project was treated the same way as each function and went through two major revisions to improve general performance and reliability and was enriched with further functionality from the optional parts of the requirement specification.

1.3 Delimitations

The authentication between client and server will use a ticket authentication already implemented to secure that only authorized users can use the handheld part of the system. Additionally, privacy and further security will be provided by encrypting the traffic between server and client with industry standard SSL.

The implementation language of Jetas Quality System is C# both on the server and the client and therefore the synchronization module has also been implemented in C# which makes changes in the future as easy as possible for the developers at Jetas Quality Systems AB.

Jetas Quality System uses Windows Mobile operating system for the client handhelds. Thus the client synchronization part will be developed to work within these PDAs memory and processor capacity.

1.4 Overview

This report consists of two main sections; the first gives a background(section 2) where an overview of synchronization is given, as well as two examples of algorithms suggested by other research articles (section 2.2 and 2.3) which did not fit the needs of this project. An explanation of the target system (section 2.4) and some technologies (section 2.5 – 2.7) used in the solutions in the project is also given. The second main section (section 3) describes the produced solution in three different chapters.

2 Background

2.1 Synchronization techniques

There are many techniques available today to use as synchronization primitives, a few are timestamp based, hash-value based, version and logical/global clock synchronizations. The different techniques are useful in different scenarios and different Database Managers support different techniques and primitives. The network and client types are also important factors when deciding which primitives to use.

There are also different major philosophies to choose from when creating a synchronization or data replication scheme. One has to consider whether the synchronization should be full or partial and also whether an optimistic or pessimistic approach to conflicts should be implemented. Data replication is when the same data is maintained on more than one computer called a replica. Using data replication is important in distributed systems to maintain high availability and reliability. It allows clients to access the data when some of the replicas are offline and also improves performance when the clients can be divided to different replicas, the one closest to their geographical position and by that decrease their latency and allow multiple accesses (Yasushi Saito March 2005).

2.1.1 Pessimistic and optimistic synchronization

Pessimistic replication is a synchronization technique which tries to maintain single-copy consistency. A widely used pessimistic technique is to have a primary replica that has access to all objects and handles all updates and when an update is done to the primary replica it writes all the changes to all secondary replicas. If the primary replica for some reason becomes unavailable the secondary replicas perform an election to select a new primary replica. This technique performs well in smaller networks with low latency and few failures like local-area networks. But using pessimistic replication in a big network like the Internet which is highly unstable with variable latency and availability will not work so well. There is also a need for people to work independently in for example software development and cooperative engineering, sometimes in different locations and disconnected from servers and the Internet, and therefore a need for optimistic replication.

Optimistic replication allows clients, like mobile devices, to work concurrently and assumes in the synchronization that conflicts will appear and relies on conflict solving to eventually agree on a consistent state instead of a blocking conflict avoidance scheme, like in pessimistic techniques, which blocks other users from writing, or even accessing a data block while another user has writing privileges. Some advantages the optimistic synchronization has over pessimistic are better availability and flexibility. The components in the network can work in an off-line state and then regardless of other changes commit its work. They can also work without knowledge of the whole network and over big networks like the Internet and updates can spread epidemically over the network to all replicas with help of techniques like epidemic replication. The only real drawback with optimistic replication is that it can lose consistency so it's important that the application can handle conflicts and inconsistency and ensure that conflicts are solved eventually and no errors occur while a potential conflict is present (Yasushi Saito March 2005).

2.1.2 Full and partial synchronization

Full and partial synchronization or slow and fast synchronization, as they are also called, are two different synchronization methods. The slow or full synchronization is when the whole replica is copied over to the client and is performed when the client synchronize with a new server or hasn't synchronized within a defined period of time and therefore has an "old" replica. This time period is often the lifetime of change tracking values for a database. Fast or partial

synchronization only synchronizes a subset of the replica depending on logs or change tracking values. The subset is updated and added objects/database rows as well as information on deleted data. Partial synchronization can only be done against the last server the client did a full synchronization with (Yasushi Saito March 2005).

2.1.3 Synchronization Algorithm based on Message Digest

A synchronization algorithm based on message digest (SAMD) is a synchronization algorithm based on a central server for computation and synchronization that works against a database server and is connected to a wireless network to which the clients also connect. The clients in this algorithm are various mobile units such as PDAs, smart phones and handheld PCs. As the mobile devices has limited battery, computing power and memory capacity, it's expensive to process large amounts of data and without a stable network connection, it's preferable to do as much computations as possible on the server. The SAMD algorithm lets the mobile device download a limited amount of replicated data from the server which is needed for its user to perform work in an offline state. A problem for many server-client synchronization schemes can often be that the synchronization algorithm is database dependent in a way that it uses metadata, triggers or timestamps. That is not the case with SAMD as it only uses standard SQL queries in its communication. This means that there are no limitations to use different databases on the server and clients (Mi-Young Choi May 2010).

The message digest algorithm used is a hash function (H) that takes a message (M) and compromises it down to a fixed length hash value (h).

$$h = H(M)$$

The message M is a data row from a database table with random length that is transformed into a message digest. Any changes in the message will generate a different hash value. This is useful when detecting changes in the database against earlier synchronizations and also differences between the databases (Mi-Young Choi May 2010)

Both the client and server database have data tables, and for every data table they also have one *Message Digest Table* which is located on the server database. The client's data digest tables store the primary key value for the row on the data table it's linked to on the mobile device. It also has a message digest column and a flag that states if there have been any changes to the row. The server also has a *Message Digest Table* for the server data that has the same columns as the client digest table and an extra column, the deviceID of the client.

The synchronization is performed in three phases. In the first synchronization phase the client sends its data necessary for synchronization in one transmission to the server by using a batching process that the SQL query is capable of handling. On the server side the mobile data is transformed into a message digest values and compared with the values already in the digest table. If the message digest values are equal there is no change and no need to synchronize the row, but if they are not equal the new message digest value is inserted in the client digest table and the flag is raised to mark a change. In the second phase the same thing is done on the server and its digest tables with the difference that only the rows with the deviceID of the synchronizing mobile device is updated to save performance. In the third and last phase the server and client digest tables are joined together for the rows where either the client or server or both sides has a flag raised. If the server has the flag raised and not the client the server message digest value is copied into the clients digest table, the flag is reset and the data is sent to the client. If the client has the flag raised the procedure is done in the other direction with the difference that the server already has the data and just can insert it in its database and if both flags are raised there is a conflict which has to be solved with the chosen conflict resolution scheme. In the case either the client or the server has inserted a new row that row is copied to the other side's digest table and

the data is sent. If instead a row is deleted the message digest value is set to null and the other side will delete its row too (Mi-Young Choi May 2010).

The benefits of using SAMD is that there is very little processing done on the mobile devices, they have the data they need in order to work in a offline state and when synchronizing the client sends over all its data to the server which does all computations and then the client may get some data back. This also keeps the database on the client as small as possible which saves space and performance. Drawbacks are that the client always sends its entire data to the server in synchronization which can be troublesome with a bad network connection. There will also be a lot of extra tables on the server side.

2.1.4 Transaction Level Result-Set Propagation

The Transaction Level Result-Set Propagation (TLRSP) model consists of a fixed network with three different types of nodes; database servers, local servers and mobile support stations. Each database server maintains its own replica of the database. The mobile support stations all have wireless interfaces to be able to communicate with the mobile devices, which have their own replicas of the database and resides outside the fixed network (Zhiming Ding April 2001).

TLRSP is an algorithm for optimistic database replication between mobile devices and database servers and thus, allows the mobile devices to access and commit changes to its local replica of the database in an offline-state. When reconnected, committed changes on the mobile device are incrementally checked for conflicts on the server side and if none are detected, the data is inserted into the server's replica of the database. Then the database server, which received the updates, spreads them to the other database servers' replicas to maintained consistency (Zhiming Ding April 2001).

A mobile device can be in three different stages. The consistent state is occupied after synchronization is completed, when all conflicts have been resolved, and until there is new data committed to the database. In this state the mobile device database is an exact replica of the server database. In the Accumulating state, the local replica of the database has begun to change. Local transactions can be performed concurrently but are serialized when written to the database in order to keep track of the latest version of each object. All read sets, write sets and result sets are logged in the mobile device to be used in synchronization. A read set is all objects read, write set is all objects written to the database and result set consists of data objects from the writing and a Timestamp. The third state is Resolving state. When the mobile device reconnects to the database server a synchronization process will start by sending its locally committed data, found by the logs, to the database server. The database server solves all conflicts that may occur and incorporates the data into the database and then answers the mobile device with the updated data so it can return to consistent state (Zhiming Ding April 2001).

The synchronization process is divided in two phases, upload and download. In the upload phase the mobile device starts by creating an Upload Transaction Queue (UTQ). In UTQ the changed sets are included, which are found with help of the logs. The sets are included in the order they were written to the mobile database to ensure serializability and then the UTQ is transferred to the database server where it will be processed. The database server first creates a data access set, which is the read and write set from the UTQ combined. Next the objects in the UTQ and database are locked and verified against each other. If the data in the UTQ read set are identical to the data in the server database and the UTQ result set's Timestamp matches the Timestamp in the database that object's access set is moved to the commit set, if the verification fails it is instead moved to the cancel set. When each object in UTQ are either in the commit set or the cancel set, the latest version of each object in the commit set will be inserted into the database and then spread to the other database servers. The cancel set will be discarded and all database locks are released (Zhiming Ding April 2001).

In the upload phase the Timestamp from the latest synchronization, from the mobile device, will be compared to the Timestamps in the database server and all objects that have been changed will be sent to the mobile device as updates. Finally the mobile device goes back to consistent mode, as mentioned above (Zhiming Ding April 2001).

Benefits from using TLSRP are that in synchronization just a part of the database is transferred from the mobile device to the server which saves bandwidth and execution time on the server side. Using logs on the mobile device makes it easy to know which data has been changed and get it for synchronization. Drawbacks are that changes made on the mobile device can be cancelled on the server if the data already has been changed.

2.2 Supporting Frameworks

2.2.1 Windows Communication Foundation (WCF)

The Windows Communication Foundation is, in a .NET oriented environment, the preferred technology for building connected applications (Leroux Bustamante och Landry 2009). The WCF was introduced along with .NET Framework 3.0 in January 2007 and with the release of .NET Compact Framework 3.5, WCF-support was also available for Windows Mobile powered devices.

A WCF-service support many different types of services and scenarios; such as classic client-server applications, asynchronous and even disconnected calls using queuing services, web services based on SOAP protocols and many more. It also supports a rich set of features such as different types and layers of security and message encryption, streaming services, transport sessions, transactions, data contracts and the exchange of serializable types. Unfortunately, out of these features, only the exchange of data contracts and serializable types are available on the Compact Framework. (Leroux Bustamante och Landry 2009).

A conceptual view of a WCF-exchange will consist of a WCF-service and a WCF-client consuming the service. The service is hosted by a managed process; most common is probably in Internet Information Services (IIS) servers which create endpoints for the service. An endpoint consists of three main parts, an address, a binding and a contract. The address describes where request messages should be sent, the actual application server might be different from the IIS-server, the binding describes which protocols is supported by the receiving end and, finally, the contract describes which operations are available and what type of data they require and return. Additionally, the contract can be made to include definitions of all complex classes that can be used by the different operations.

To be able to consume WCF-operations, a WCF-client must access the service metadata, i.e. the contract, so that it can format the requesting messages correctly and parse the replies accordingly. To ease the development software that acts as WCF-clients, the developer can use built-in tools in Visual Studio to create a proxy for the WCF-service, which act as the interface to the server in the client application. The proxy-generation tools can parse the contract info provided by the endpoint and creates a proxy type that can be included in the client-project. The proxy-type contains the definitions of the service defined classes and structs, as well as the required logic to call the operations for the service and the definitions of how the serialization and deserialization of service data should be performed.

2.2.2 Language-Integrated Query

Language Integrated Query (LINQ) is a concept that has been developed, at least in part, to ease the use of non Object Oriented (OO) data stores; most common are probably XML and relational databases, in an OO software project. With LINQ, instead of adding specific XML or database-query features, there is a general-purpose query methodology that can be applied to all sorts of data stores, both native .NET Collections and Arrays as well as relational databases and XML.

With LINQ, the regular query operators that an experienced SQL-developer is used to, like the SELECT, WHERE and JOIN clauses, grouping and sorting as well as aggregate operators like SUM, MAX or AVERAGE, have been made available for use upon just about any type of in or out of memory location. Any developer is also free to create their own operators and operations to be used in LINQ-queries, or to overload the default operations.

LINQ uses deferred query evaluation, which means that a query is not performed until it is iterated over. This can be a great advantage since the resources needed to perform the query-execution is not allocated until it's actually needed, but a developer that does not take this into account can run into serious trouble. For example, if a previously defined query is iterated over several times, it is evaluated and computed from the original data store each time, and will thus yield different results if the data store has been altered. This is of course a great advantage in some situations, but if a query is used as input for another query, especially if the data source is a remote SQL-server or some huge data set, the careless developer can easily create poorly running programs. (Box och Hejlsberg 2007)

2.2.3 LINQ to SQL

As mentioned above, LINQ has support for connecting to a relational database and the main feature to do this is called LINQ to SQL. This feature however is only available in the regular .NET Framework. The .NET Compact Framework 3.5 has support for LINQ-queries to objects and XML, but not SQL.

When using LINQ to SQL, a developer first creates entity classes that represent the tables in the database in an entity model, this subsystem is called the *Entity Framework*. In this model, all relations that are present in the database can be included as well, and as an extra feature, a one-to-many relation can, in the entity model, be realized at both ends; as a reference in one end and as a collection of entity objects in the other. The model is then linked to an actual database server upon construction and the model will provide the interface to communicate with the database and LINQ-queries involving the entity objects are passed to the framework and parsed into SQL which is then executed by the database server. Changes that are made to the entity objects returned by queries and entity objects created by the program are tracked during execution and are written back to the database upon a call to the save-method of the model.

This way of communicating with the database provides numerous advantages to the regular way of creating SQL-adapters and then providing the SQL-expressions as strings to be sent via the adapters. For the developer, it will be more convenient since the Intellisense will be active and help the programmer by correcting a large amount of possible errors while writing queries. Moreover, working with entity objects gives the advantages of a strongly typed Object Oriented language, like type checking and code safety, while still having the speed and stability of a relational database. (Kulkarni, o.a. 2007)

Some of the drawbacks with working with LINQ to SQL instead of sending pure SQL to the server are a slight loss of functionality; for example getting access to Change Tracking information can be very hard without major revising of the automatically generated code for the entity model and entity objects. As mentioned above, a careless developer can also quite easily create slow running programs by accidentally making multiple calls to the database when traversing the relations realized by the entity framework. A final concern is that the way LINQ to SQL translates complex queries into SQL is not always perfect and can create bottlenecks, which a developer needs to watch out for.

2.3 Target system

The current release of the Jetas System is the Jetas 5.x and is the fifth generation of the Jetas Quality system product line. The main idea behind the system and the Jetas Method is that all objects on which service will performed is contained in the system and all service tasks, be it interval based routine checkups or fault complaints by users, is administered via the Jetas system. All service on site is also verified by barcodes placed on the objects on which service will be performed. (Jetas Quality Systems u.d.)

The core of the Jetas system is the property model, which is generated for each customer when they implement the Jetas Method and then updated as their properties are updated. The property model is built up from objects of different types. Some of these types are built in, like building and property, whereas others are specified by the customer to fit their needs and their administrative model. This creates a model, which is very logical for the customer, but since the properties of different objects are dynamic and defined by type, the database model to contain the objects becomes quite complex.

The system is administered via a web interface where both the object models can be created and work orders and tasks can be created and administered. After a work order is created, it is assigned to a user, either via the web interface or via the User's PDA and then the work order, along with all the information about the property model objects is transferred to the User's PDA during the next synchronization. The work order is then performed by the User and verification is done by the User scanning one or several barcodes with the PDA, which can only be found at the correct objects. All changes to the objects related to the work orders is then uploaded to the server and updated in the database.

2.3.1 Server system

The server system for the Jetas 5.x System is centred around a Microsoft .NET 4.0 Web Service system and a MSSQL 2008 Server for database access. The system can roughly be split in three parts; the web system which handles the administrative web system, the mobile service part which handles synchronization, authorization and some other services for roaming PDAs and the common support part which performs supportive functions for the two access systems. The scope of this project will mostly focus on the mobile service part of the project.

2.3.2 Client system, Jetas Mobile 5.x

The PDA program, the Jetas Mobile 5.x is a Mobile application built on Microsoft .NET Compact Framework 3.5. The Jetas Mobile also relies upon an installation of Microsoft SQL Server Compact Edition 3.5 on each PDA to store and query the subset of the Server database that is transferred to the PDA in a synchronization session.

The Jetas Mobile 5.x enables the User of the system to view which work orders are available based on the Users credentials and assign them to the active User. Then it also allows a User to download all work orders which are assigned to the active User to the PDA.

Once the work orders, along with the different objects that provide all necessary information about the objects that the work order concerns and the tasks that is to be performed, the user can choose to start a work order from the list that is provided. Once started, the application provides the user with instructions and requests the user to scan barcodes and provide other input, such as comments, the status of a particular object or the value of a meter and, once finished, the user is allowed to report the work order as completed. The user can also choose to pause a work order for later completion. In addition to performing work orders, the user can also create fault complaint work orders directly in the PDA. This can be either based on a work order that the user was working on, or stand alone, possibly based on scanning the barcode of a faulty device that the user happened to pass by.

2.3.3 Interoperability

The mobile service part of the Jetas 5.x server system is built around two WCF (Windows Communication Foundation) services, which provide a number of different operations. The two WCF-services are the old synchronization service, which will be handled in more detail later, and the general-purpose mobile service. This service provides several operations for different tasks needed to ensure the correct operation of the Jetas Mobile 5.x system and the synchronization. It provides operations to set the mobile unit's clock correctly and to ping the server to verify connectivity and to determine the PDA's IP-address from the Server's point of view. The service also provides operations for authentication that enable the other service operations to authorize the PDA and User for each operation. The authentication works as a Kerberos-like system (http://www.kerberos.org/software/tutorial.html, 2011-01-25), where the PDA holds a long term key for the User which is currently registered to use it and the PDA requests a temporary Authentication Ticket from the server when it's about to perform an operation which requires the User to be authenticated.

2.3.4 Old synchronization scheme

The previous synchronization scheme, that the goal of this project was to replace, was based on Microsoft Synchronization Framework. With this scheme, much of the logic controlling the synchronization process was delegated to the framework and adaptations, local optimizations and other deviations from default behaviour was either quite unwieldy or straight out impossible to create. The synchronization framework service was wrapped by a WCF-service that provided the connection interface between PDA and server.

The previous framework consisted of four WCF-Operations which is the way a WCF-service exposes it's logic and can be seen as Remote Call Methods. The first exchange info between server and PDA to ensure further correct behaviour and the second provide the Schema from which the Database on the client is created. The client application thus, does not contain information about the database upon which it relies on for correct operation; this is provided by the server in the synchronization. This Schema that the server provides is a subset of the Schema that defines the server-database and is generated from the definition of which fields are included in the sync.

This way of defining the database, which of course is not without its benefits, has some drawbacks. The first obvious drawback with this scheme is that the constraints of the database is not distributed to the PDA with the Schema and has to be coded and added on the PDA after the synchronization is complete, if safe manipulation of the data shall be possible on the PDA. Another drawback with this system is that the PDA cannot have any fields or tables in the database without a corresponding field or table on the Server. If the developer wishes to store values or settings on the PDA, he/she has to rely on other regular files or such. The final drawback that was recognized by this scheme was that it made switching to an alternate DBMS (Data Base Management System) in the PDA quite difficult. The other two available operations are for sending and receiving updated data, and they send and receive .NET – DataSet-objects which are serialized to XML via the WCF-service.

To track changes between synchronizations so that the PDA only needs to download updated information and to detect conflicts when a PDA uploads data that has been changed on the server as well, MSSQL's built-in ChangeTracking functionality is used. When using this functionality, a retention period is set, which is how long the server will store the changes. If a user wait for longer than this period between synchronization, the server cannot determine which changes has occurred and thus, all relevant data must be sent again to the PDA and, more importantly, changes that is uploaded from the PDA cannot be properly checked for conflicts and might have to be discarded. (Microsoft Corporation u.d.)

3 Product

As mentioned above, data synchronization and database synchronization is a wide and diverse field of research and development, both in business and the academic world and many different algorithms are in use and have been suggested for different purposes. This project began with a research phase with the goal to try to find a previously developed algorithm that could be adapted and implemented in the Jetas Product, but with regards to that goal, the research phase must be seen as a failure.

Within the academic field of research, database synchronization involving mobile, roaming, or partially connected agents, the names vary, is a wide field with many contributions from different research groups. Many of these suggestions are interesting from a theoretic point of view but they most often allow that changes be made to any database involved, at anytime, anywhere. This is of course very good in applications where this is needed, but in the context of Jetas Quality Systems, it is not needed.

3.1 The synchronization prerequisites

In the Jetas Server Application Database, there are approximately 130 different tables fulfilling different purposes, and the Mobile Client has a subset of 20 of the tables in its mobile database. Except for one case, which will be explained below, the Mobile Application only makes changes to certain fields in two of these tables, and this is done in very specific ways.

The two tables that can be changed on the PDA are the WorkOrder-table and the WorkOrderContent-table. On the WorkOrder-table, only two fields can be changed, Status and CheckInDate, and this is done when a WorkOrder is finished, and when it's finished, it is only kept on the PDA until uploaded to the server in the next successful synchronization, and then deleted. The WorkOrderContent-table has a several fields that can be set by the PDA, but when a row is set by the PDA, a Timestamp-field is set at the same time, which is NULL before. This means that no advanced algorithm is needed to keep track of what has changed on the PDA, and there are very limited conflict-scenarios that can actually occur in this synchronization environment.

As mentioned above, there is an exception to the above-mentioned limitations. The Jetas Mobile application also gives the users of a PDA the possibility of creating Fault Complaints in the PDA when they discover some faulty equipment in the field. When such a Fault Complaint is created, rows are created in the two tables mentioned above, but also in several others. When these special WorkOrders are created they are marked in a certain way in the database, and since they are created on the PDA and uploaded to the server, no conflicts can occur with these rows since they are new.

3.1.1 The Work Order

What the user of the Jetas Mobile Application wishes to do is to perform WorkOrders, and so, the most important table to keep track of before, during and in between synchronizations is the WorkOrder table. It can even be said that most tables in the PDA-database are present to enable the WorkOrders to be performed and, for most of them, what is needed during synchronization is directly decided by which WorkOrders that are to be downloaded.

Figure 1 shows a state-chart of the most common states that a WorkOrder can be in and the most common transitions between these states. The states that a WorkOrder can occupy in the PDA are the *Pending Assignment*, *Checked Out*, and *Complete in PDA*. When a WorkOrder is *Created*, it can either be assigned to a Group (Group Assignment) or directly to a User (Assignment). A WorkOrder that is *Pending Assignment* will be downloaded to all user's PDAs which are a member of that group, so that they can choose to assign the work order to themselves. If the user chooses

to do this, the server will be notified during the next synchronization and the Server will assign the WorkOrder to the user and WorkOrder will be in the *Assigned to User*-state.

A WorkOrder, which is in the *Assigned to User*-state, will be moved to the *Checked Out*-state and then downloaded to the PDA when the user, which it is assigned to, performs synchronization. Thus, the *Assigned to User* is a state that will only exist on the server. When a WorkOrder is downloaded in the Check Out state, all the additional information that is linked to it, like the Property Objects it is linked to and instructions on what to do with them, is downloaded as well. This information is not downloaded when a WorkOrder is in the *Pending Assignment*-state.

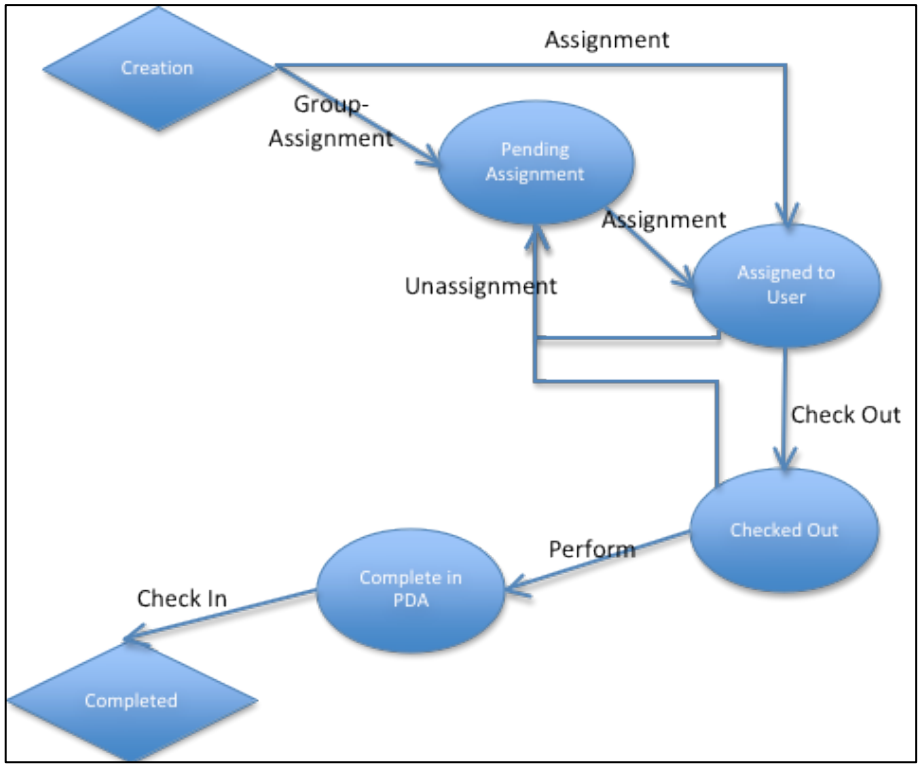

Figure 1: The States of a Work Order

When the WorkOrder is in the *Checked Out*-state in the PDA, the user is free to perform the WorkOrder and when this is done the WorkOrder changes state to *Completed* but only in the PDA, which is why it is named *Completed in PDA* in the figure. Upon synchronization, the info that was changed when the WorkOrder was performed, which is in the WorkOrder-table and the WorkOrderContent-table, is uploaded and written to the server and the WorkOrder and related information is deleted from the PDA. Additionally, a user can choose to pause the WorkOrder in the performing phase. In this case, the WorkOrder itself is unchanged, but the WorkOrderContent-rows, which is where the actual work that has been performed is stored, will be updated. When a WorkOrder is in Check Out state, but one or more of its WorkOrderContent-rows have been updated, the WorkOrder is said to be *Paused*, which can be seen as an intermediate state between *Checked Out* and *Completed in PDA*.

Apart from the above-mentioned transitions, where the flow can be seen as moving towards completion, or forward, a WorkOrder can be reassigned to either a group or another user, as well as unassigned. The WorkOrder can also be set to Void, which is the equivalent of deleting the WorkOrder (a proper DELETE is never performed on a WorkOrder on the Server) at any time during the transitions. The reassignment is the main risk for conflicts (technically, setting the WorkOrder to Void is a conflict, but it is quite easy to resolve) and this will be discussed in the Server View chapters.

3.1.2 Data model

The Data Model that was developed for this application is tightly coupled with the synchronization algorithm and explaining one without the other can be problematic. To grasp the concept in the algorithm, an understanding of the Data Model is a requirement and thus this section will have to come first.

Data categories

The way that the WorkOrder state transitions work and how the different relations of the database tables interleave have led to a classification of data that should be sent to the PDA into 3 different categories, where a database table belongs to one of the three (with a few exceptions where tables belong to more than one).

The first data category is the *Metadata*. This category contains data that is not needed for a specific WorkOrder but is needed for all WorkOrders or for the proper working of the application. These include Users, Groups and their relations, as well as different decorating properties and settings for other tables. This category also has certain objects that are needed to create Fault Complaints in the PDA but which reside in the tables that belong to the *WorkOrder-related* category.

The second category is the *WorkOrder-related* category. This category contains the WorkOrderContent mentioned before, as well as the objects on which the WorkOrder should be performed and their properties. In short, everything that the WorkOrder, and some relation-realising tables, have references to, except for the tables that are explicitly *Metadata*, can be considered to be *WorkOrder-related* data. This category of data is the main bulk of the data that is sent between the server and the PDA.

The third and final category is simply the WorkOrders themselves.

Database entities

To represent and transport the actual data from the server application to the client application, the rows and tables have to be represented in some way. There are of course many different ways this could be implemented, and there are advantages and disadvantages to all of them. The previous synchronization scheme sent .NET DataSet objects. These objects are generated by the SQL-Adapters when querying the database and can contain several DataTables, which in turn contain DataRows that hold the actual field data. The advantage of sending DataSets is that they are created by the adapter so no further manipulation would be needed, and the client could easily write their content to its database without much, or any, manipulation on the client. The drawback with this scheme, however, was twofold. Firstly, constructing these objects was not as easy with LINQ-to-SQL, which was desired for the advantages it provided in code readability; and secondly, using a .NET built in structure would impede platform independence.

The solution that was implemented was to simply create classes that reflected the Database tables of the PDA-database and mark these classes and their properties with WCF-attributes. This lets the WCF-backend create a serialization scheme for the classes, and also methods that let clients read this schema and create the same structure of classes in their end. The base structure of these classes was created with the Visual Studio Tool that creates an entity model for usage with LINQ-to-SQL from a database file, using the PDA-database as a model. The classes were then refactored, removing the code and attributes that implemented Entity Framework and adding constructors and WCF-attributes.

One of the major drawbacks with this scheme is that these objects have to be created a second time for each row or object that is returned by the LINQ-to-SQL queries which creates both additional computing and additional memory usage. An obvious improvement would be to expand the Entity model of the existing Jetas System with these, scaled down, versions of the database entity-classes and have the Entity Framework backend perform the construction only once and then send the entity-classes to the PDA. This was not done because the entire Server application was developed so that it would not affect the rest of the application, and starting to experiment with the entity framework could have great consequences to the rest of the application, but it is a road that should be considered if this prototype is to be extended and implemented for use in production.

Carrier bundles

To organize the downloads that will be done to the PDA, the above mentioned Database entities need to be bundled together according to the rules of the synchronization algorithm and placed in containers that can be ordered and batched in order to achieve a good compromise between the size of the information exchanged (the larger size, the higher risk of errors occurring) and the communication overhead of setting up connections multiple times.

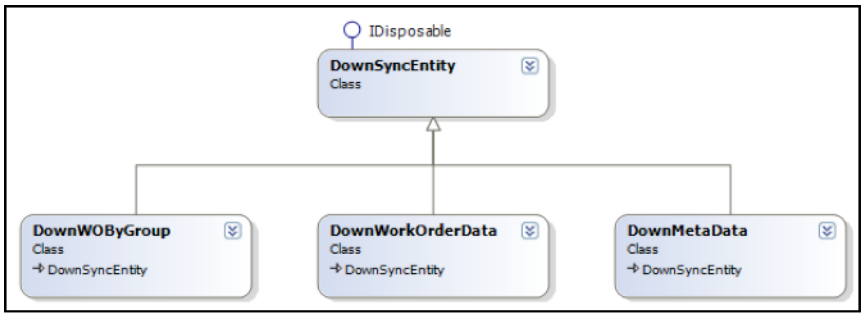

Figure 2: Class diagram of Bundles for Download

The three bundles created for the downloading part, shown in Figure 2, (from server to the PDA) is almost the same three as the three different data categories specified earlier. The *DownWOByGroup* class is made up of only WorkOrders and is to be made up of the WorkOrders assigned to a group that the user is member of. The *DownWorkOrderData* class will contain a WorkOrder-object which is assigned to the active User, as well as the *WorkOrder-related* data that is linked to that WorkOrder, the *DownMetaData* will contain the *Metadata*.

All three classes will be of the common type *DownSyncEntity* so that they can be put in the same queue and methods and such can be of that return type.

Upload bundles

This chapter has been very much about download so far, and of course, a data model scheme is not complete without both upload and download. The distinctions between different types of data for the download described earlier, however, does not apply to the upload (from PDA to server) of data. As described earlier, the way that the PDA can change the data stored on it is very limited, and thus, only those tables that can be changed on the PDA needs to be represented in the Carrier bundles for the upload.

Figure 3 shows the four different bundles that are used to wrap data for upload, and the four bundles represent the four ways that the PDA can change data in its database, described earlier. The UpWoPaused and UpWoComplete are very similar in content; the UpWoPaused contains WorkOrderContent that has been changed for a WorkOrder but not the WorkOrder (since it has not been changed) and represents a WorkOrder that has been paused during the performance-phase, while the UpWoComplete also contains the WorkOrderObject, which now has been set to Completed.

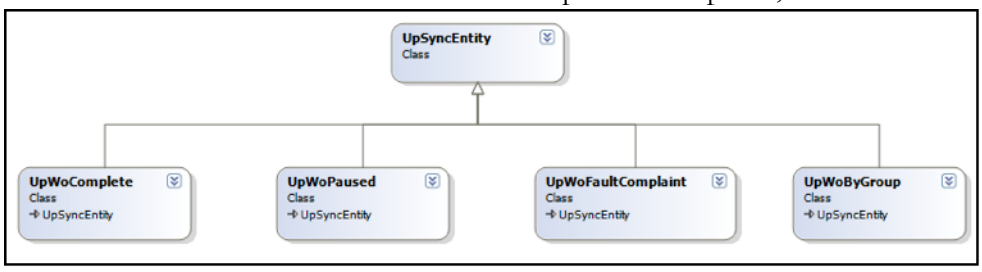

Figure 3: Class diagram of Bundles for Upload

The UpWoByGroup represents the WorkOrders that the user wishes to assign to him/herself, and is thus not an actual change in the PDA-database, but an instruction for the server to perform this change. Thus the UpWoByGroup simply contains a list of all the WorkOrders that the user wishes to assign to him/herself. Finally, the UpWoFaultComplaint contains all the

objects representing the rows added to all five tables involved when creating a FaultComplaint. As with the download bundles, the upload bundles are of a common type UpSyncEntity, for practical purposes.

3.2 The Client View of algorithm

To describe the actual algorithm that was developed, the easiest way to start is probably to start by describing the client's view of the entire sync progress, as this is the view that will give the best overview of the flow of the algorithm. This is because it's, in essence, the PDA that controls most of the synchronization process, the server mostly responds to requests.

The synchronization from the PDA's point of view includes two phases, first the upload phase and then the download phase. See Figure 4 for a full synchronization overview.

3.2.1 Upload phase

When the PDA initiates synchronization, it starts with the upload phase by constructing all the entities and puts them in their bundles. The bundle then attaches information on which type of upload data it contains so the server later will know how to handle the bundle, and puts them in a send queue.

As mentioned earlier there are four types of upload bundles and therefore four ways to find them. UpWoComplete and UpWoFaultComplaint are both found by the status of the Workorder table which indicates either *Complete in PDA* or *Created*. When a workorder is paused, the application does not change the WorkOrder, so to find them it has to look at the WorkOrderContent-table instead. Here a timestamp-field is set if the content has been changed, and, as mentioned earlier, a WorkOrder that has had data changed but is not set to Complete is considered as paused. Thus if a WorkOrderContent has had the Timestamp set since last time a synchronization was performed, it is due to be uploaded as a paused WorkOrder. The last bundle, UpWoByGroup, is a list of WorkOrder identifiers and is only sent to the server

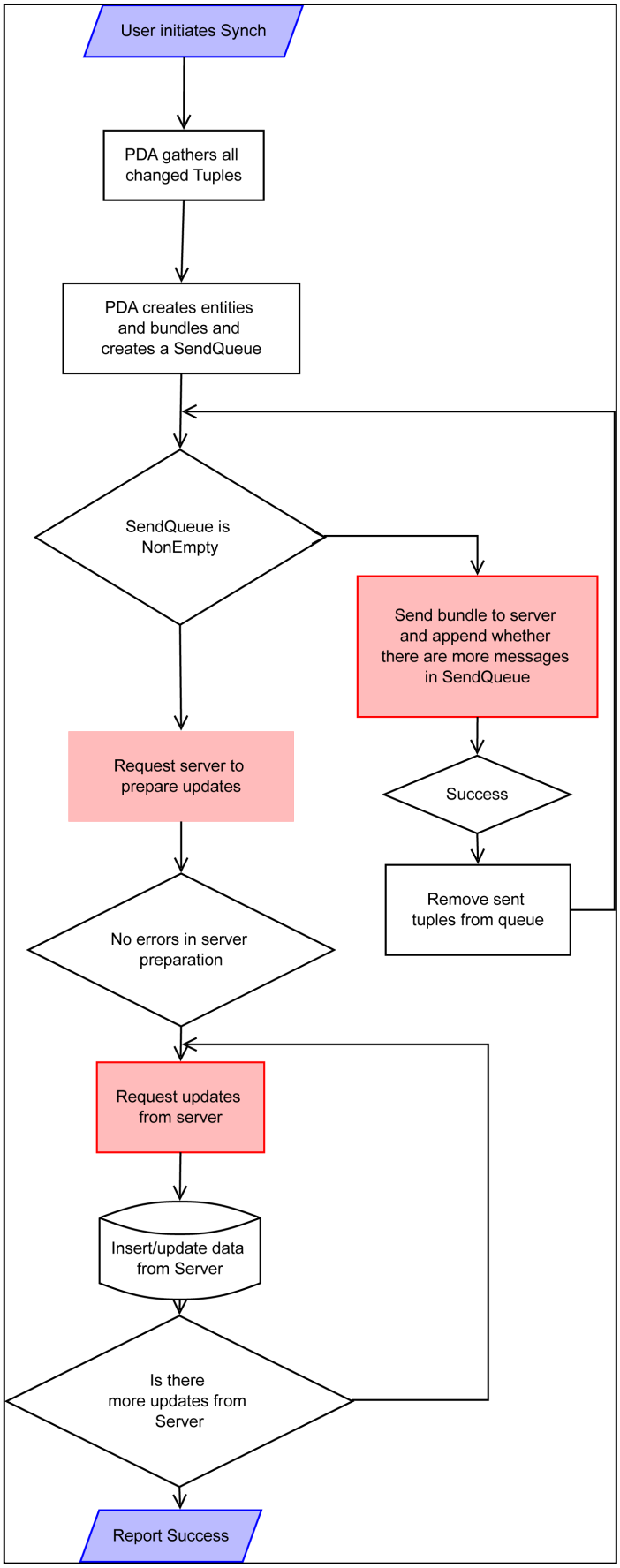

Figure 4: Client view of synchronization

in the upload phase if the user has chosen to assign himself Workorders from the Pending Assignment list.

When all information to be sent is gathered, the bundles are extracted from the queue, decorated with an authentication ticket and synchronization information and sent to the server. Early on, this project assumed that WorkOrder wise would be a good batching size, which proved to be wrong, so a more suitable batching scheme should be introduced here before the prototype is used in a production environment.

The last part of the upload phase is to remove all completed Workorders and all FaultComplaints from the PDA database; they are now in the server database and are no longer needed on the PDA. The WorkOrders and FaultComplaints are located in the same way as before and then deleted. When a WorkOrder is removed, its content and history data is automatically removed as well due to database constraints, but the other objects connected must be handled separately because they can still be needed by other WorkOrders. A clean up routine is called that removes the *WorkOrder-related* objects that is not connected to a WorkOrder in the PDA-database nor is one of the *Metadata* objects with dual purposes.

3.2.2 Download phase

The download phase begins with the client calling the prepare operation on the server, to have the server calculate which data should be sent to the PDA. To enable the server to perform these calculations, this call includes information about the last performed synchronization, if any, of the active user and an authentication ticket. The mobile device in return gets a bundle containing: if there is new data to download, new synchronization information and two lists with WorkOrder identifiers (Assignment and Group Assignment), which will be used in the end of the download phase to ensure that only the right WorkOrders remain in the PDA after the synchronization is completed.

If there is new data to download from the server, the request data operation is called on the server to receive a data-carrying package. The downloaded packages can contain one of three types of bundles; DownMetaData, DownWOByGroup or DownWorkOrderData. If a DownMetaBundle is to be downloaded it will be sent in the first package as it contains data needed to insert the other bundle types, such as information about the users and groups. The *Metadata*-bundle also contains a special flush flag that indicates if the *Metadata* is outdated or not. If the *Metadata* is outdated, all Meta-tables will be erased of all data, except the data about the active user, before the new data is inserted. If the *Metadata* isn't outdated new data will be inserted or updated into the database and data deleted from the server will also be removed on the PDA. The DownWOByGroup bundle contains WorkOrders assigned to the users groups that the user can chose to assign to him/herself and the DownWorkOrderData contains WorkOrders assigned to the user and all additional data needed for performing them, this is, for example, data about the objects that the WorkOrder affects and the contents and history of the WorkOrder. The batching used in the download phase sends the *Metadata* in one package, as well as the WorkOrders assigned to the users groups. DownWorkOrderData bundles are also sent all in one package, but this isn't optimal for most of the customers and should be changed before the prototype is included in a release.

In the first version, the new synchronization module used regular sql-queries to insert and update tables in the database. All data was first run in an update query to the affected table and if it were successful the data row was already in the table and now updated, if the update weren't successful the data didn't exist an insert query was used. This method resulted in lots of database communication which is known to be a bottleneck. So in the second version of the new synchronization module the update and insert queries were changed to use DataTables and DataAdapters instead. This resulted in two commands to the database for each table and package instead of two commands for each data row, which increases performance significantly.

After all packages have been downloaded and inserted/updated into the database there is no more communication with the server. To finish up the synchronization old Workorders are deleted from the database, as well as unnecessary data from Workorders that has been reassigned. All WorkOrders should be included in the two lists with WorkOrder identifiers (Assignment and Group Assignment), from the bundle received from the server in the beginning of the synchronization, to not be deleted. A WorkOrder assigned to the user who is deleted will also have its content and history deleted due to database constraints. If the lists are both empty the user have no WorkOrders assigned to himself or his groups and therefore all WorkOrders will be deleted for the active user. If the Group Assignment list isn't empty all WorkOrders assigned to the groups will have content and history data deleted to save space on the PDA.

In the last part of the download phase the synchronization information received from the server are now properly saved. Each active user has a Timestamp of the last synchronization performed stored in the User table in the database to allow multiple users on the PDA to remember their last synchronization between sessions. Then the Timestamp together with the version value for the *Metadata* is stored in a settings file for easy and fast access. All users share the same *Metadata* value. To finish up the download phase the special cleanup database routine is once again called to remove unconnected objects from the database after the different deletions done in download phase.

3.3 The Server View of algorithm

The view of the algorithm from the server is somewhat more fragmented than the PDA's view of the algorithm. This is because the server only responds to the requests from the PDA, which is responsible for the flow of the synchronization. The reason the algorithm was created this way was in partly because the PDA will always be the part that initializes synchronization and the WCF-Framework does not support sessions when communicating with Compact Framework-powered units. A session-scheme could of course have been built on top of the WCF in the algorithm, but it was decided that the more fragmented approach would be sufficient to fulfil the requirements for this project.

The server is divided into three parts, that is the WCF-service that provides the Synchronization service provides three operations, which are available for the PDAs to call. These three operations will be discussed in detail below.

3.3.1 Version handling

One of the key components in any synchronization scheme is how it keeps track of versions, or changes, to the data. In this project, different methods are used at different stages, but the most commonly used way is a method that we have chosen to call optimistic TimeStamp checking. The main usage of this scheme is to, during the prepare-phase of the synchronization, determine which objects (or rows) are new (or updated) and which already have been sent to the PDA in a previous synchronization. The first thing that one can note with this task is that its purpose is solely a performance booster. To send less data is faster than to send more data. If the scheme thus misses an update, such that the PDA already have the object, and it's added anyway, it will cause performance to drop slightly but the behaviour of the program will not suffer. If, on the other hand, the scheme would fail in the other way, and would think that the PDA already had a object which it has not, the synchronization could fail when trying to insert rows to the PDA-database whose foreign key constraints were not met, or worse, the application could possibly fail at a later stage in a less controlled way. This then, would be an error, and must be avoided. Thus there is a bound on which type of misses that the TimeStamp checks can be allowed, and not allowed, even though a scheme with no misses would be optimum.

Another reason to use Timestamps is that the application already uses these to keep track of changes internally. All database tables has a trigger which fire each time a table is updated and writes the time of the change to a Timestamp-field called LastChange. This makes it easy to find conflicts if the PDA uploads the entire row object, since the PDA does not change this field, if the uploaded does not match the one on the server, a change has occurred on the server since the object was downloaded to the PDA.

To enable a scenario which uses TimeStamps to avoid all misses, one would either have to store, and possibly exchange, one timestamp per object (database row) or, in the instance with only one, common, TimeStamp, one would have to lock all concerned tables from updating during the retrieval of all the data from all the tables. Neither of these two scenarios would be very effective; the first because there can be several hundred objects and handling all those TimeStamps would be very unwieldy, and the second because the retrieval from all the 20 database tables is a time consuming operation and locking over such a big (and buzy) section of the database would be both a huge bottleneck for the database, as well as a huge risk of causing deadlocks with other operations.

The goal then is to have just a single TimeStamp to keep track of. Since we do not wish to use locking while reading the tables we will have to allow for one type of misses but make sure to avoid the others. Thus the program stores the UTC-time, called LastSync, exactly before it begins to retrieve the data that is to be sent to the PDA. If a change then occurs to a row after the TimeStamp is stored but before the row is read, then the updated version will be sent to the

PDA. When the PDA synchronizes again, the server will believe that the PDA has an older version of said data row, and send the same row that the PDA already holds to the PDA. More formally, if data row A has a LastChange > LastSync (the TimeStamp stored), then it may be different from the one in the PDA. If, on the other hand, A's LastChange <= LastSync, then the PDA has this version of the data row. Thus this scheme fulfils the above requirements of allowing the right type of misses, while avoiding the other and this is the scheme that is used to limit the number of objects to download to the PDA to the most newly changed.

A valid question is why the program does not use the built in change tracking that is used by the old synchronization scheme. The answer to this is that this information is not included in the LINQ-to-SQL Entity model and after discussions with the developers at Jetas, it was decided against changing in the Entity model, so that this product could be plugged in alongside the existing products. To access the Change tracking now, regular SQL must be used, which defeats the purpose of switching to LINQ. An advantage of not using the Change Tracking is also that if a user does not synchronize inside the period which changes are stored, not all the information is flushed.

3.3.2 The Upload operation

In the upload operation, the server receives the data that a PDA has updated, or changed, since the last synchronization and the objective of this operation is to commit these changes to the server database. To be able to do this, the server must first check so that no conflicting operations have occurred on the server since the last synchronization with the PDA.

Upon receiving an upload package from a PDA, the first task is to authenticate the user for the database that it wishes to write to. The Authentication is of the same type as described previously in section 2.3 about the existing product. The PDA, via a different service operation, provides an authentication ticket, which is checked so that it is still valid. In this ticket, user credentials, which company and database to connect to among other things, are appended. The process of authentication is also used to construct the LINQ-to-SQL, Entity Framework model object, which is used for all communication with the server database. This is the same authentication procedure that is performed in all of the different service operations in the Synchronization system.

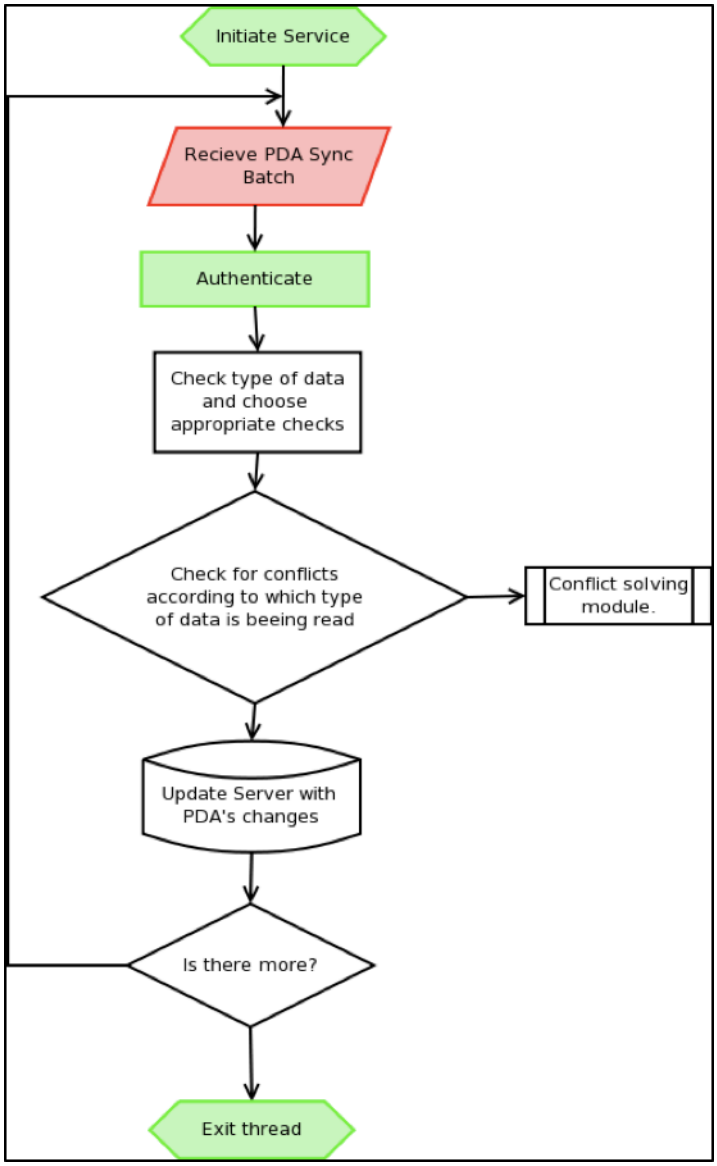

Figure 5: Upload scheme

After authentication is successful, the application proceeds to determine which type of data that the upload package contains to determine which checks and actions to perform next.

If the package received was of type UpWoByGroup, what is received is basically just a list of primary keys of the WorkOrders that the PDA-User wishes to assign to him/her-self. The possible conflicts that could disrupt the operation in this case is if the WorkOrder already has been assigned to another user (and/or completed), set to void or that the WorkOrder has been assigned to another group. To find the WorkOrders that might be involved in a conflict, the LastChange-value of the WorkOrders from the database is compared to the LastSync-variable. If the LastSync is greater than the LastChange-value, there has been no conflict and if not, then there might have been a conflict and further investigation is needed. All conflict cases but the last can be found by a change of status to the WorkOrder, and the last is easily found by checking if the user is member of the group that the WorkOrder is assigned to. As this scheme might give false positives when comparing with the LastSync-variable, the conflict analysis should have a default action if no conflict was found to commit the changes.

If the package received is of the type UpWoFaultComplaint the flow for the server is quite straight forward. The server checks that no WorkOrder (and the other table entries) exists with the same primary key as the objects uploaded. If this succeeds, the objects in the package are written to the database. If a conflict occurs, the highest probability is that the PDA has already uploaded the same Fault Complaint earlier, but failed to remove it, in which case, the information can be safely ignored. The other possibility for a conflict, that the GUID-value, which is the generated primary key in all of the tables, was generated to a duplicate of one already in the database, is so improbable that it is ignored.

The handling of UpWoPaused and UpWoComplete is very similar since the content of the packages is very similar. If the package received is an UpWoComplete-package, first the WorkOrder is retrieved from the server database and the application compares the uploaded WorkOrderobject's LastChange-timestamp to the one gotten from the Database. If the check succeeds, the WorkOrder is changed and the application moves on to the WorkOrderContent described below, but if it fails, the application performs a series of tests to determine what has changed so it can determine if it can perform the change suggested by the PDA or not. Since the PDA has uploaded the WorkOrder-object in its entirety, it can be compared thoroughly by the application.

For the UpWoPaused, the checking process starts a little differently. The application retrieve the WorkOrder to perform checks, but here it's up to each customer how the application proceeds. One option is to always write the WorkOrderContent, because work was performed and should be logged, while others might argue that if the work was not performed by the now assigned technician, it is of no interest. Either way, a decision is made at this point.

If the decision is to continue, the application proceeds to retrieve all the WorkOrderContent-objects that were in the received package. The first check after that is to check each server side object for whether it's Timestamp-field is NULL. If it is, then this object has not been written to, and it's safe to write to it. The Timestamp field is written to by the PDA when the user performs the task that the WorkOrderContent represents. The actions if the Timetamp-field is set can be defined for each customer by checking which user is assigned and so on, but the suggestion in this project is to compare the Timestamps and store the newest one, since that represents when a technician was at the site last. Since the deviation allowed is just 1 minute, it is very unlikely that if Timestamp A > Timestamp B then B was at the site later than A. In this scenario, the technicians should by all reasonable possibilities have met each other there.

Once all the data objects in the package is either written or discarded, the operation returns and the server is ready to receive the next batch. Thus there is no special order of functionality, server

side, that keeps track of whether a PDA has more to upload or not, the server handles what it gets and the PDA makes sure that it's updates is delivered before it requests new information.

3.3.3 The Prepare operation

The prepare operation is, for the server, the most demanding operation in the synchronization service. The goal of the prepare operation is to determine which data should be sent to a the PDA, extract this information and build a send queue that will serve the Request operation later on, which is the operation that actually sends the data to the server. There are certain rules regarding the queue that has to be followed throughout the execution of this operation and that has affected the design of the Data bundles and the operation greatly. Firstly, a Data object (i.e. a data row, thus an entity object) must be added to the queue before or at the same time as any other object that is dependent on it, and secondly, a certain Data object must only be added to the queue once in each synchronization session. Additionally, a Data object should only be added to the queue if it is not in PDA already or if it has been updated to a new version on the server.

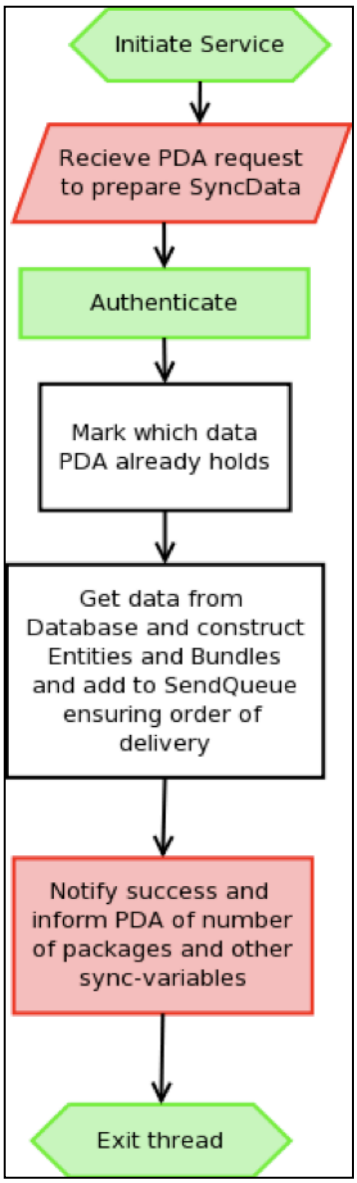

Figure 6: Prepare scheme

First of all, this means that all *Metadata*, that is added in the DownMetaData-bundle, must be added first, since this contains data that all objects can have references to. Since some of the data that can be added to the Meta-bundle also could be added in the DownWOData-bundle, info on which of these objects were added must be kept to ensure they are not added again later on. The Meta-data phase is in general quite straight forward, with one exception. Most of the computation in the Meta phase is simply to formulate a series of queries, one for each table in the PDA-database (with a few exceptions where relations make it possible to easily write queries that retrieve several LINQ-entity-objects at once) and then simply construct the Entity-objects and add them to the Meta-bundle.

Unfortunately, some of the tables in the Meta-category have a trait that the tables involved in the other Categories lack, namely that in these tables, data can be deleted. This calls for some special care to be taken with these tables, since the regular check done by the LINQ-queries to the database would not find a deleted object, since it has been deleted, and thus, the PDA would not be informed about the deletion and would keep that object indefinitely. Since the built in functionality of Change Tracking, described briefly in the section about the old synchronization scheme previously, is already turned on for these tables in the database, the choice was made to use that possibility instead of creating a custom tombstone model. Regrettably, the Change tracking only stores the primary keys and the operation (Insert, Update or Delete) that was performed, so there is no way to know if a deleted row was on the PDA or not. The way that the deletion tracking works is that it sends all primary keys of objects that was deleted in each table since the last synchronization to the PDA, which then makes sure that none of the sent keys are left in it's database. Since the Change tracking have a limit on how old changes it stores, if the last time the PDA synchronized is longer than this period the PDA is told to delete all it's previously stored Meta data and all Meta data is retrieved with the queries mentioned above, regardless of when it was last updated. The Delete-tracking phase also gets the last-version variable used by the Change-tracking and adds it to the

return-bundle for the prepare operation so that the PDA can store it for the next synchronization.

After the Metadata has completed and the Meta-object has been added to the send queue, its time to fill the DownWOByGroup-object. This object, like the Metaobject, is created in just one copy, since it is not expected to contain any huge amounts of data. To fill the DownWOByGroup-object, the program defines a query that retrieves the WorkOrders that are assigned to a group which belongs to the user, but not assigned to a user, and which has been changed later than the LastSync-value that the PDA provided. The Entities to send to the PDA are constructed and added to the DownWOByGroup-object which is added to the send queue.

The final task, as well as the most complex task, is to create and fill the DownWOData-objects. One of these objects are created for each of the WorkOrders that is to be downloaded to the PDA, and all the data that is needed for that WorkOrder is added to the object, unless, as mentioned before, that data is already added to the queue or can be determined to already reside at the PDA. In the prototype, these objects were first sent one by one, but tests proved that these batches was much too small to provide good performance. Thus a wrapper for the DownWOData-objects was created to be put in the queue, which held a predefined number of DownWOData-objects.

To create and fill the objects the following workflow is used. First of all, the previously mentioned block list, which contains the keys of already added objects, is extended to include all the WorkOrder-related objects that can be determined is already at the PDA. This is done by querying for all the WorkOrders that are older than the LastSync-variable and joining these with a helper table that contains the keys of the WorkOrder-related objects, and the resulting keys are added to the block list.

After this, a helper query is created which joins the WorkOrders to sync (those newer than LastSync) with the WorkOrder-related helpertable. This query is then queried by one query per object type (with two exceptions, where the database structure allows one query to get 2 object types without making the query to complex) where each object query grabs the object of the type it is of from the result of the first query and then joins these keys over the actual server table. To keep track of which WorkOrder the resulting objects belong to, the result of these queries are grouped by which WorkOrder they belong to which, of course, might cause quite a lot of duplicates to be returned.

Each of these query-results is then iterated over and in the first query-result, which always generates results, the actual objects and the WorkOrder Entity-objects are created. Then the key of the data-object is checked against the block list, and if it is not present there, it is added to the DownWOData-object, and the key is added to the block list.

To have this amount of loops and repeated use of the same query, performed over and over, does not seem to be very efficient. Attempts were made to make the querying process more effective by pasting all queries into one but were abandoned quickly. Mostly, these attempts were abandoned because it would defeat a large part of the purpose with this project; the new synchronization scheme should be transparent in how it progressed and be easy to debug, extend and modify. One massive query that would perform more or less everything would almost definitely be severely complex and very hard to understand for someone else than the programmer who wrote it. A second issue is that the way LINQ-to-SQL translates LINQ-queries into SQL-queries is sometimes not as optimal as could be wished. Thus, the more complex a query in LINQ is, the higher risk of LINQ-to-SQL of interpreting the query into something less than optimal in runtime.

3.3.4 Request operation

The third and final operation in the synchronization service is the request operation, which is the actual download operation where data is sent to the PDA. The operation begins, like the other, by authenticating the user and PDA and then the actual work can begin. Upon each call to the request operation, the application picks an object from the queue to create a package to send to the PDA. Information about whether any object after this one exists in the queue is sent so that the PDA knows whether to request for more packets or not. Then the package is simply sent to the PDA.

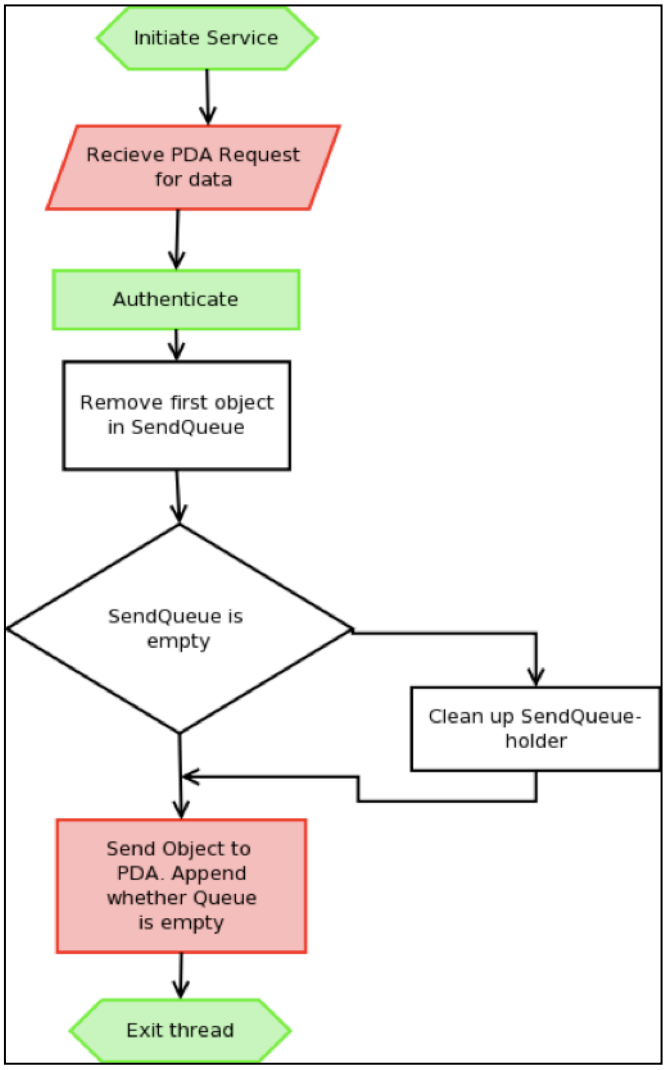

Figure 7: Request scheme

If a more elaborate batching scheme should be developed, it could either be implemented here, by sending several objects from the queue at once, or in the prepare operation by bundling data into bigger pieces before adding them to the send queue.

4 Conclusion

This report describes in part database synchronization as a concept and then more in detail the development of a replacement synchronization scheme for the Jetas Quality Systems application. The synchronization algorithm that was developed in this project is very much adapted exactly for this context and the assumptions and limitations that this algorithm relies on make it quite unsuited for implementations in any other context. But how did the newly developed algorithm perform compared to the previous implemented algorithm?

Functionally, it performs all the functions of the previous synchronization, with a few additions. The functionality tests that have been performed shows that the functionality of synchronizing the correct information in several different tested scenarios work satisfactory, although exhaustive testing has not been performed. Additionally the new algorithm can easily implement customer adapted batching schemes that depend on each customer's network conditions. The new algorithm also has support for several parallel users of the same PDA without discarding the data when the user switch, as was done before. The conflict handling when PDAs upload their data is also quite transparent in the new algorithm, and can be extended and adapted in the code by simple case- and if-statements.

Performance wise, this project has not been able to conduct any production like tests. All performance tests that was performed was done with an office machine acting as web- and application server, whereas the database server was still at the remote site where the servers reside. This means that any figures are very uncertain, since the performance and bottleneck could be very different when the programs are ran on the proper application server, with close proximity to the database server. In the limited performance tests that was performed, the new algorithm, when the initial batching scheme was replaced with a scheme similar to the old application, which sent 30 WorkOrders (with data) each batch, the new algorithm performed slightly better with regards to the time it took for the user to synchronize. Since this is not verified in a production like environment, it is not counted as a complete success yet, but regarding that the database is optimized for the old synchronization scheme and that there is room for several improvements in the new synchronization scheme, the authors still regard it as a small victory.

Of course, there are some issues remaining in the product, and some of them will be handled below, but overall, the project's goal was to produce a prototype for a synchronization module, and a working prototype has indeed been presented.

4.1 Problems

4.1.1 Testserver

As mentioned above, all tests in this project was performed on the local office machines that the development was done on. This also means that all testing was done with the built-in webserver in Visual Studio 2010 or the built-in IIS-version that is shipped with Windows XP. None of these webservers support the SSL-functionality needed to test the SSL-wrapping functionality that was implemented in the product. This is rather unfortunate, but as the function is implemented mimicking the previous application's SSL-functionality, we are confident that it should work with only minor tweaking. It uses the WCF frameworks standard SSL-functions, which should work if the application works without in non SSL-mode.

4.1.2 Complex environment

The severe complexity of the system, and especially the duality of some tables to be both part of the *Metadata* and *WorkOrders-related* data categories forced the project to several large revisions of data model and revise large parts of the solutions along the way. It's hard to see how this could have been solved in another way, but the idea of incremental development has the drawback of sometimes running into the problem that in order to get further, one has to go back and change a previously working component since it does not allow for or provide the functions or data needed by the added functionality.

4.2 Recommendations for further development

To change the mobile device database from Microsoft SQL Compact Edition (SQL CE) to SQLite will simplify the development of Jetas Quality System for other platforms than Windows Mobile in the future since SQLite is platform independent and open source. SQLite is said to have better performance than SQL CE but the drawback is that foreign key constraints does not seem to be very safe and large parts of the client would possibly have to be updated if a change should be implemented (SQLite development team u.d.).

The batching scheme that is used in both the upload and the download operations could quite easily be improved to increase performance in the synchronization. How much data the different packages should contain to be optimal could be made to depend on the customers' circumstances. For example a customer on a poor network connection may want more packages with less data in each compared to a customer on a high speed network.

An improvement that would enhance the performance in several stages at once would be to implement the new Data Entity-classes that were created to send to the PDA as LINQ-to-SQL-Entities. This would both remove the need to construct the Data-entities from the LINQ-to-SQL-Entities, saving memory and execution time as well as decrease the size of the objects returned in the queries in application, and possibly the data retrieved from the database server.

Another improvement that could be made to the application by changing in the LINQ-to-SQL Model Framework is to make either the new classes mentioned above or the ones used today include the Change tracking information that is only available by pure SQL in the application today. This would take away the risk of misses described above and the Change tracking variables could be used in conflict detection in a higher rate than now.

5 References

Box, Don, and Anders Hejlsberg. *LINQ: .NET Language-Integrated Query* . 2 2007. http://msdn.microsoft.com/library/bb308959.aspx (accessed 02 15, 2011).

Jetas Quality Systems. *www.jetas.se*. http://www.jetas.se/en (accessed 02 11, 2011).

Kulkarni, Dinesh, Luca Bolognese, Matt Warren, Anders Hejlsberg, and Kit George. *LINQ to SQL: .NET Language-Integrated Query for Relational Data* . 3 2007. http://msdn.microsoft.com/library/bb425822.aspx (accessed 02 14, 2011).

Leroux Bustamante, Michele, and Nickolas Landry. "WCF Guidance for Mobile Applications ." *Codeplex.com*. 12 05 2009. http://wcfguidanceformobile.codeplex.com/ (accessed 02 14, 2011).

Microsoft Corporation. *Configuring and Managing Change Tracking* . http://msdn.microsoft.com/en-us/library/bb964713.aspx (accessed 02 14, 2011).

Mi-Young Choi, Eun-Ae Cho, Dae-Ha Park, Chang-Joo Moon, Doo-Kwon Baik. "A Database Synchronization Algorithm for Mobile Devices." *IEEE Transactions on Consumer Electronics, Vol 56, No 2*. May 2010. 392-398.

SQLite development team. *SQLite documentation*. http://www.sqlite.org/docs.html (accessed 2011 □□□ 10-03).

Yasushi Saito, Marc Shapiro. "Optimistic Replication." *ACM Computing Surveys, Vol 37, No 1*. March 2005. 42-81.

Zhiming Ding, Xiaofeng Meng, Shan Wang. "A novel conflict detection and resolution strategy based on TLRSP in replicated mobile database systems." *Seventh International Conference on Database Systems for Advanced Applications*. Hong Kong , China, April 2001. 234-240.

Appendix A – List of abbreviations

This is a list of all abbreviations that occurs in the report.

ASP	Active Server Pages
DBMS	DataBase Management System
GUID	Globally Unique Identifier
IIS	Internet Information Services
LINQ	Language INtegrated Query
MSSQL	MicroSoft Structured Query Language
OO	Object Oriented
PDA	Personal Digital Assistant
SAMD	Synchronization Algorithm based on Message Digest
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SQL CE	Structured Query Language Compact Edition
SSL	Secure Sockets Layer
TLRSP	Transaction Level Result-Set Propagation
UTC	Coordinated Universal Time
UTQ	Upload Transaction Queue
WCF	Windows Communication Foundation
XML	eXtensible Markup Language