Master Thesis

30 June 2006

Technical University of Denmark

Offline Framework for Smart Clients

Abstract

This master thesis describes a generic and flexible Offline Framework that supports the development of smart clients, which must be functional in an intermittently-connected environment based on a Service-Oriented Architecture (SOA).

A new Hybrid Architecture that combines advantages of the Data-Centric Architecture on top of the Service-Oriented Architecture is presented. This Hybrid Architecture is combined with a Built-in Cache Framework solution, which together provides the foundation of the Offline Framework.

The designed Offline Framework is implemented and utilized in a case study in cooperation with PricewaterhouseCoopers, which results in a working offline capable smart client solution. This developed smart client illustrates that the generic and flexible Offline Framework successfully can provide offline functionality to intermittently-connected smart clients.

Resumé

Formålet med denne afhandling er at beskrive et generisk og fleksibelt Offline Framework, der kan benyttes til udvikling af Smart Clients, som skal kunne fungere i et løst forbundet miljø baseret på en service-orienteret arkitektur (SOA).

En ny hybrid arkitektur, som kombinerer fordele fra den data-centriske arkitektur med den service-orienterede arkitektur bliver præsenteret. Denne hybride arkitektur kombineres med en indbygget cache-løsning, som tilsammen danner grundlaget for Offline Framework'et.

Det udviklede Offline Framework implementeres og benyttes i et konkret eksempel foretaget i samarbejde med PricewaterhouseCoopers, som resulterer i en fungerende Smart Client, der understøtter muligheden for at kunne arbejde offline. Den udviklede Smart Client illustrerer, at det generiske og fleksible Offline Framework succesfuldt kan forsyne løst forbundne Smarte Clients med offline-funktionalitet.

Preface

This thesis is the result of a Master of Science project at the Department of Informatics and Mathematical Modelling at the Technical University of Denmark. The project has been supervised by Associate Professor Bjarne Poulsen and assisted by Assistant Professor Christian Probst, and has been carried out in cooperation with PricewaterhouseCoopers.

We would like to thank our supervisors Bjarne Poulsen and Christian Probst for the helpful guidance provided and making this project possible. We would also like to thank PricewaterhouseCoopers in Denmark for providing the best possible surroundings and our contacts Leif Christensen and Kim Bryndum for all their help and good advice during the project period. A special thank goes to Gitte Lind for proofreading this master thesis report.

We also highly appreciate the Subversion repository provided by Associate Professor Anders Fosgerau at the Depart of Communications, Optics and Materials. The Subversion repository provided has been a great help during the implementation process and writing of this master thesis report.

Finally, we would like to thank our friends Lars Drejer Olsen and Casper Jensen for proofreading this master thesis report and providing valuable advice and criticism.

Copenhagen, 30 June 2006

Alex Jankowski

Henrik Georgi

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Achieving robust offline/online support in occasionally connected applications can be a difficult and time-consuming task. This master thesis project focuses on analyzing how a framework for occasionally connected applications should be developed to provide the necessary offline/online support.

The purpose of this chapter is to:

- provide a brief introduction to the concept of smart clients
- explain the background and motivation of the project
- provide the overall objective of the project
- give an overview of the project plan
- describe the prerequisites needed to understand the report
- provide an overview of the thesis structure

After having read this chapter, the reader will have obtained the necessary knowledge and understanding of the context of the following chapters.

1.1 Smart Clients

Until today, client applications have mainly been divided into two categories, the thin client and the rich client. These two different types of approaches of designing client applications are now being challenged by a new concept called "smart clients". This new concept tries to combine the advantages from both worlds with the use of standardized Web Services, offline/online support and device adaptability.

To begin with, client applications were primarily based on the use of thin clients in the form of text-based terminals connected to a central server, in many cases a large mainframe computer. Later - with the development of the personal computer, the rich client became the dominating type of application, mainly because of the greater possibility of using the computers' local resources. As the internet evolved, the focus shifted back to thin clients, as many applications today are developed as web applications, which has made the internet browser one of the most used thin clients today.

Rich clients and thin clients have their advantages and disadvantages. The rich client is known for its rich user interface, responsive behaviour and high developer productivity because of the possibility of utilizing local resources in an efficient manner. The developer is not constrained by the limitations of a thin client and therefore has a greater freedom to create a more user-friendly application. The disadvantages are that rich clients typically have a heavy footprint and can be difficult to deploy. Especially, deploying changes to a rich client can be a time-consuming and costly task. One of the reasons for this is for example the

"DLL Hell" that rich clients on the Windows platform normally suffer from. The combination of dependency on special DLL files with the bad versioning capabilities of the DLL concept can cause many complex problems for these rich clients. Many of these issues have been solved with the introduction of the .NET Framework, but the footprint and deployment of these rich clients are still very dependent on how they are designed and implemented.

The thin clients are known for their broad reach, which makes it easy to access this type of application from almost any computer. Other advantages are that changes are made to a central server, which gives an easy change management and a simple deployment, because the changes come into effect on all clients instantly. The disadvantages are that the application at all times needs a network connection to function and in many ways is limited by the capabilities of the underlying thin client, which typically reduces the usability and user experience of the application. [1]

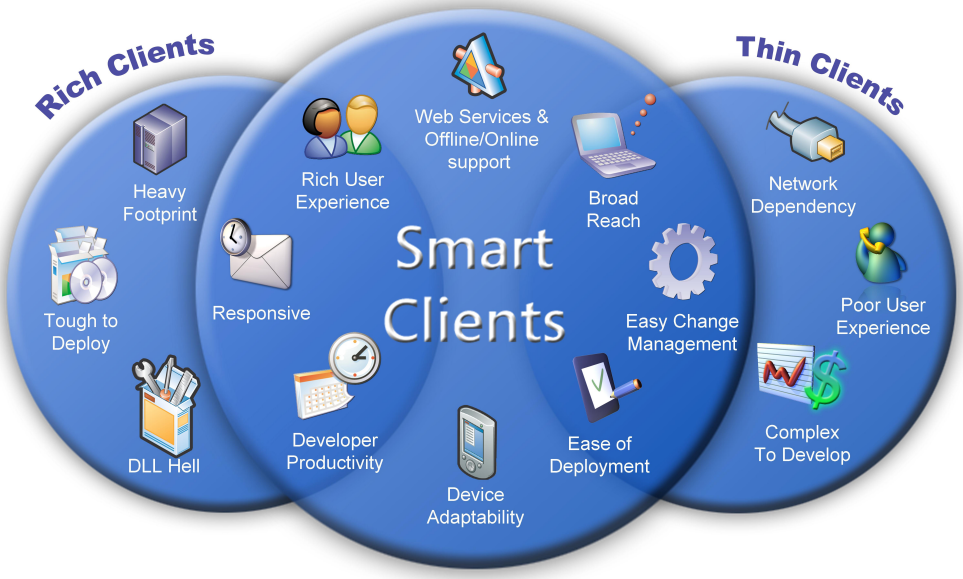


Figure 1: Relation between rich clients, thin clients and smart clients [5]

The concept of smart clients is to combine the best features of both client types with standardized Web Services, offline/online support and device adaptability, which is illustrated in Figure 1. Smart clients are by Microsoft defined as easily deployed and managed client applications that provide adaptive, responsive and rich interactive experience by utilizing local resources and intelligently connecting to distributed data sources. This means that a smart client provides the same rich interface as a rich client and also has the ability to manage content changes efficiently. Moreover, a smart client can provide these features while minimizing the resources that it uses, such as disk space and memory. [24]

This new concept of smart clients is the main focus area of this master thesis project. Smart clients have a wide range of functionalities that they must support as illustrated in Figure 1, but including all these topics would be out of the scope of this master thesis project. This project will instead primarily focus on how to achieve offline/online support in smart clients, as this functionality is seen as one of the most central functionalities in smart clients.

1.2 Background and Motivation

Today, many companies make use of Enterprise Resource Planning (ERP) systems to manage all kinds of business processes, e.g. finance, human resources, sales, marketing, reporting and registration of time. These ERP systems therefore normally play a very central role, which typically makes them the mostly used and important IT system of the companies.

PricewaterhouseCoopers in Denmark (PwC) uses an ERP system called Maconomy to handle the majority of their business processes. In many cases, optimization of the processes when users interact with ERP systems can be both cost saving and ease the everyday life for the users, for which reason this optimization is highly prioritized by the management of PwC.

The background of this master thesis project is that PwC has a desire to provide offline functionality to their Maconomy ERP system by developing a smart client. In the following two subsections, a brief description of PwC will be given and the motivation of the project will be described in the case scenario of this project.

1.2.1 About PricewaterhouseCoopers

PwC is the world's largest knowledge firm with expertise in assurance, advisory and tax services and has over 130,000 employees located at 771 offices in 148 countries. In Denmark, PwC has 1,265 employees distributed on 17 offices throughout the country.

PwC is divided into the three main departments called Assurance, Advisory and Tax. This master thesis project is made in co-operation with Advisory, who advises its clients on solutions to complex business challenges. Advisory has comprehensive industry insight which is supported by the strong competences within financial, analytic and technological advising possessed by the department's 150 advisers.

Advisory is subdivided into ten competence centres ranging from enterprise risk management over financial effectiveness to sustainable business solutions.

The competence centre Financial Effectiveness supports its clients by optimizing their financial management, business processes and system selection by analyzing their current situation and needs to find a suitable solution and helping the clients with the final implementation and project management.

A few years ago, PwC in Denmark wished to replace their old customized business applications with a new ERP system. For this purpose an internal project was created, which

was managed and carried out by consultants from Financial Effectiveness. The ERP system was successfully implemented and the competence centre now has the responsibility for making ongoing improvements by optimizing their own financial management and business processes of the chosen ERP system. This master thesis project is carried out in cooperation with the Financial Effectiveness competence centre.

1.2.2 Case Scenario

The employees of PwC often work at locations where it is not possible to get a connection to the company's network. The Maconomy ERP system can be accessed either by a rich client in the form of a normal Windows application or by a thin client, in relation to which the employees logon to a web portal using an internet browser, but both solutions require a connection to the company's network. The employees at PwC can therefore not perform time registrations, when connectivity is unavailable.

The registration of time is a central task for the employees, which currently is a time-consuming and troublesome task and impossible when the employees are working out of office. Many employees therefore have to postpone the task, which only makes it harder for them to remember when and how long they have worked on the different jobs. Therefore, the employees and the management of PwC show great interest in simplifying this process to minimize the time spent on the time registration process and at the same time maximize the data quality.

The motivation of this master thesis project is to develop a generic framework that can provide offline functionality to smart clients. This framework should then be utilized to develop a smart client solution for the described case scenario.

1.3 Project Objective

The objective of this master thesis project is to analyze and design a framework that can be used to develop smart clients that must provide offline/online support on existing systems. It is central that this framework makes the offline and online state as transparent as possible for the developer, which will ease the process of developing smart clients significantly and result in more advanced and user-friendly client applications.

This master thesis has the following objectives:

- **Analyze the main design architectures of occasionally connected applications**
  An overview of different architectures should be provided to make the reader familiar with some of the primary design challenges related to occasionally connected applications.

- **Provide an analysis of how offline functionality can be achieved in the chosen design architecture**
  An overview of existing approaches for providing offline functionality should be analysed. This analysis should provide a solution strategy of providing the needed offline functionality.

- **Analyze the challenges in connection with the chosen solution strategy**
  The challenges according to the chosen solution strategy should be analysed in more detail.

- **Provide a generic and flexible framework that can be used in multiple smart client solutions**
  The research conducted should result in a generic and flexible framework, which should ease the development of smart client solutions needing offline capability.

- **Utilize the developed framework in a case study**
  The developed framework is to be used in a case study, which is carried out in cooperation with PwC. The purpose of the case study is to develop a prototype of a smart client that will improve and ease the time registration in PwC's current ERP system.

The above objectives of this master thesis project will be used throughout the report.

1.4 Timetable and Process

This master thesis project began on 14 November 2005 and ended on 30 June 2006. Figure 2 shows a timetable for the entire project indicating milestones of the project by dark rectangular marks.

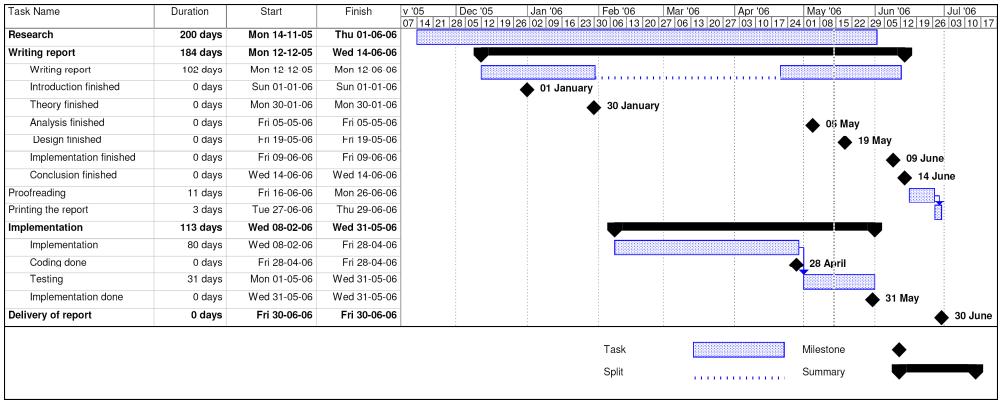


Figure 2: Timetable for the master thesis project

The timetable has formed the basis of the project planning, but should be seen as an ideal of how the project should be carried out. However, this project is highly influenced by the fact that it is an iterative process, which will be reflected in this master thesis.

1.5 Prerequisites

To read and fully understand this master thesis, there are a number of prerequisites listed below that the reader is assumed to have general knowledge of.

- Microsoft .NET Framework 2.0 platform
- Web Services and Extensible Markup Language (XML)
- Distributed systems and multithreaded applications
- A general understanding of database systems
- Basic understanding of synchronization and replication
- Understanding of the C# language

Because C# has many similarities to Java, knowledge of Java will in many cases be sufficient to understand the C# code shown in this report.

1.6 Thesis Structure

This master thesis is structured as follows:

- Chapter 1 presents an introduction to the topic of this master thesis and the scope of the problem that is addressed.

- Chapter 2 analyzes the main design architectures of occasionally connected applications and provides a solution strategy for the problem addressed.

- Chapter 3 analyzes the challenges of the chosen solution strategy. The solutions found will be utilized in the implementation of the framework.

- Chapter 4 analyzes relevant technologies and how these technologies can help solve the problem addressed.

- Chapter 5 describes the design and implementation of the framework developed.

- Chapter 6 describes how the developed framework has been utilized to solve a case study, which has been carried out in cooperation with PwC.

- Chapter 7 summarizes the main challenges in this master thesis that requires further work and analysis.

- Chapter 8 concludes the work of this master thesis by summarizing the research conducted and the results obtained.

2 Analysis of Occasionally Connected Architectures

This chapter will initially describe the problem domain of this master thesis by providing the specifications and possible risks associated with the development of a framework, which must handle the online/offline scenario.

An analysis of the main design architectures will be provided to give an overview of the advantages and disadvantages associated with these design architectures. This analysis will form the basis of evolving a design architecture that can be used to fulfil the specifications given by the problem domain.

There are different approaches to the handling of the online/offline scenario, which will be analyzed in more detail in section 2.3. These approaches are based on caching of Web Services, in relation to which results from research conducted in this area will be utilized to give an overview of the possible solutions.

The analysis made in this chapter will result in a solution strategy that influences the rest of the master thesis and is utilized in solving the described problem.

The purpose of this chapter is to:

- provide a description of the problem domain
- analyze the relevant design architectures
- analyze existing theory concerning caching of Web Services
- provide a solution strategy based on the analysis made

After having read this chapter the reader will have obtained the necessary understanding of the problem that needs to be solved and been presented with a solution strategy, which the rest of the master thesis will be based on.

2.1 Problem Domain

This section provides a description of the problem domain forming the basis of this master thesis project. Currently, PwC has a need for a framework that can support the development of smart client solutions for their ERP system. PwC has the following specifications of the framework:

- **Generic and flexible**
  The framework must be as generic and flexible as possible to make different systems offline capable.

- **Simplify development process**
  One of the main objectives of the framework is to ease the development process of new offline capable smart clients.

- **Communicate through Web Services**
  It is a requirement that the framework provided must communicate over Web Services, as this is the only allowed way to make changes in PwC's current ERP system. Changes may not be made directly to the database because all support and warranty provided by Maconomy will be lost.

- **Secure communication with authentication**
  It should be possible to provide secure communication and authentication, because business critical data is to be transferred.

- **High performance**
  The framework should provide fast access to the local data and efficient synchronization of data. The performance of the framework is very essential because good performance will lead to better usability for the end user.

The requirements for the framework needed by PwC have now been specified, but there is a risk that it is not possible to fulfil all these requirements in the framework. One risk is making a framework that should be generic will loose its flexibility. It is therefore important that a balance is found between the specified requirements, especially when making the framework both generic and flexible.

2.2 Design Architectures

There are two main architectures for designing smart client solutions; the Data-Centric Architecture and the Service-Oriented Architecture. These two different approaches primarily differ in the way data is accessed on the client and how data is exchanged between the client and the server. The problem domain described in section 2.1 specifies that communication must be performed through the use of Web Services, which restricts the use of the Data-Centric Architecture. The Data-Centric Architecture has some interesting properties, which is the reason for including this architecture in the analysis. The purpose is to give the reader an overview of the architectures used, when designing smart clients.

2.2.1 Data-Centric Architecture

The Data-Centric Architecture is characterized by having a local database instance on the client side with a replication mechanism to manage the changes made to the local data by the client application. These local changes are then propagated back to the central database server when the client gets online.

The client initially creates a subscription to the data it needs from the server so this data can be copied to the client before it goes offline. The subscribed data is typically only a subset of the data on the server, because it, in most cases, would not be a scalable solution to have the entire server's data locally stored. It is important for the server to know the client subscriptions as it needs this information to handle the client's queries, updates, inserts and deletions correctly. With this information correctly configured the Data-Centric Architecture helps ease the implementation of smart clients, because much of the offline functionality needed by smart clients is provided by the database system.

The main advantage of the Data-Centric Architecture is that all change-tracking code is contained inside the relational database. Generally, this includes code for conflict detection at both the column and row level of the database, data validation code and constraints. This implies that it is not necessary to implement change-tracking or conflict detection and resolution functionality in the smart client, since this is provided by the relational database. The developer although needs to be aware of the merge-replication scheme so that the application can be optimized for data conflicts and data update.

The Data-Centric Architecture also has the advantage that the smart clients do not have to implement a caching mechanism as this already is automatically provided by the local relational database. The local relational database also supports advanced data querying functionality, which implies that complex retrieval of local data can be performed easily. This simplifies the implementation phase of smart clients as the developers can concentrate on the business processes instead of implementing their own caching and data querying functionality, which makes the implementation of smart clients more time-saving and with lower development costs.

Synchronization of data is also handled by the local relational database, as it has all the necessary information to perform this action, which can be scheduled in jobs or be forced when needed by the smart client. This synchronization process is highly configurable, which makes the Data-Centric Architecture suitable in many scenarios. Because the database is handling the synchronization process, extraordinary business logic and local data logic is required on the client to ensure data is valid and dealt with correctly. This is illustrated in Figure 3 on page 12, where the Business Logic layer and the Local Data Logic layer are placed between the User Interface Logic and the local relational database. [1]

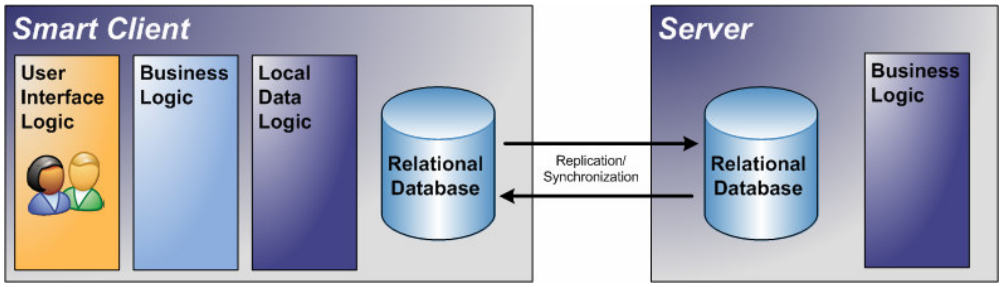


Figure 3: Overview of the Data-Centric Architecture [1]

However, there are also some major disadvantages in relation to the Data-Centric Architecture. One disadvantage is that the client and the server are very tightly coupled, because of the fact that the two relational databases must be able to synchronize using a common protocol, which typically implies that the two relational databases must be of the same kind. The use of these special communication protocols can limit the use behind firewalls. The tightly coupled nature of the Data-Centric Architecture also implies that interaction with multiple disparate systems can be difficult to achieve, as these systems can be based on different database systems. [1]

Another disadvantage is that it is required that the clients have direct access to the central database in order to utilize the Data-Centric Architecture, which in some situations is not allowed for administrative or political reasons. In such situations the Data-Centric Architecture is less appropriate.

The Data-Centric Architecture requires the use of third-party software that in many cases will impose limitations on what devices the smart client can run on, which typically implies that it cannot be run on smaller devices.

Below is an overview of the advantages and disadvantages of the Data-Centric Architecture:

Advantages:

- Robust data conflict detection at the column and row level of the database
- The local database provides data validation and constraints
- Built-in change tracking and synchronization
- Advanced data querying functionality supported
- Caching and updating easy to implement
- Time-saving implementation

Disadvantages:

- Requires third-party software
- The client and the server are tightly coupled

- Special communication protocols can limit the use behind firewalls
- Not possible to easily interact with multiple disparate systems
- Database systems can have limited features on smaller devices

The Data-Centric Architecture has many great time-saving advantages which make the synchronization and replication easy to implement in a smart client because of the built-in features provided by relational database systems. However, the architecture though also has some major disadvantages. As the smart client is tightly coupled to the server, there can be problems behind firewalls, and the use of this architecture requires direct access to the central database server, which is less appropriate when developing smart clients for existing systems. These properties make it an inappropriate architecture for the described problem domain, but it presents some interesting concepts, which can be used as inspiration to a solution strategy.

2.2.2 Service-Oriented Architecture

The Service-Oriented Architecture (SOA) is another design architecture, where the client interacts with services on the network through service request to provide the functionality needed by the smart client. The Service-Oriented Architecture is not a new concept, but has been around for some time now and utilized in connection with many different Remote Procedure Call (RPC) technologies such as CORBA, RMI and DCOM. The problem with these technologies is that they are tightly coupled to the programming platform and typically require special communication ports to be open, which makes interoperability between different systems difficult to achieve, especially if these systems also reside behind internet firewalls. Many of these communication technologies were initially intended to be platform independent and thought to be the new universal communication standard, but this wide cross-platform interoperability did not become a reality, which meant that the interest in the Service-Oriented Architecture in the beginning was limited.

The introduction of Web Services has renewed the interest in the Service-Oriented Architecture as this technology has become widely accepted as the standard communication technology used to achieve interoperability between different platforms and applications. Today, Web Services are implemented in almost every programming language and more and more software vendors are providing standard Web Service interfaces to their systems, which make interoperability between these different systems easier to achieve. [1]

The success of Web Services lies in the combination of the following four technologies:

- Hypertext Transport Protocol (HTTP) as the primary communication protocol, which today is one of the most interoperable of all communication

protocols, because it is so widely accepted and used, and therefore rarely is blocked by firewalls.

- Extensible Markup Language (XML) combined with Simple Object Access Protocol (SOAP) is used as the payload format. A large number of software vendors have adopted the XML standard, and due to this high popularity it is one of the most interoperable data formats today.

- Web Services Description Language (WSDL) to specify the service interfaces, which makes it possible for different programming platforms to utilize Web Services. WSDL specifies a protocol- and encoding-independent mechanism for Web Service providers to describe how interaction with the services offered can be performed. The WSDL standard is being worked out by the W3C[1], which defines the standard as: "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate." [25]

- Universal Description, Discovery and Integration (UDDI) protocol to provide service registry, which enables companies and applications to quickly, easily and dynamically find and use Web Services over the internet. This protocol is not a necessity when using Web Services, but can be used to achieve scalability and improve performance in large systems. [6]

These technologies are widely used today, so when talking about Service-Oriented Architecture you indirectly refer to the use of Web Services as communication mechanism, even though the Service-Oriented Architecture is a concept, which was developed many years before Web Services was introduced as described earlier. This means that SOA and Web Services today are two different terms very much related to each other.

The Service-Oriented Architecture makes it possible for the client to interact with multiple disparate services, which is an advantage compared to the Data-Centric Architecture, which is limited in the sense that it typically can only interact with one backend system. Another advantage of the Data-Centric Architecture is that the client is not forced to have a relational database instance installed locally and that the database used on the client

[1] World Wide Web Consortium, see http://www.w3.org for more information.

does not need to be tightly coupled to a central database server, which makes it possible for the smart client to fit on smaller devices. This loose coupling provides much more flexibility when developing smart clients compared to the Data-Centric Architecture.

The use of services gives better maintainability, because the business logic is located in the services, which makes error location and correction easy without affecting any clients, as long as the service interface is not changed. SOA also makes it possible to reuse code, as the developed services can be used by many different clients and systems, which can use the same services in different context. Figure 4 illustrates how a smart client accesses a server's data using a service, where the business logic associated with the service assures that no illegal operations can be made to the relational database on the server. The server provides its services through the use of service interfaces, which is encapsulated by service agents on the client side. The local data on the smart client can be handled in many different ways, which will be analysed in more detail in section 3.1. [1]

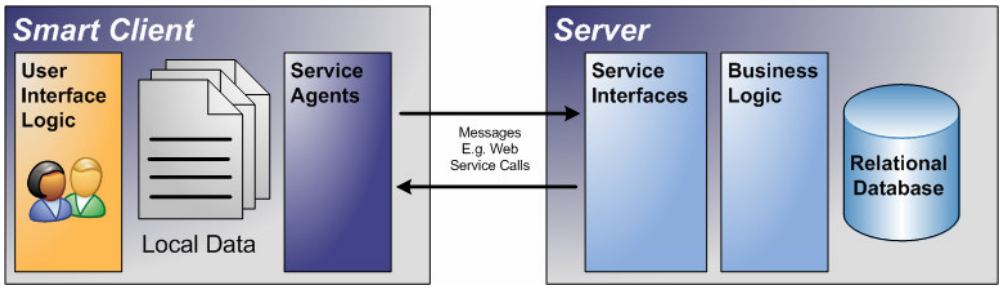


Figure 4: Overview of the Service-Oriented Architecture [1]

The main disadvantage of the Service-Oriented Architecture is that it does not provide built-in data store, change tracking and synchronization as the Data-Centric Architecture does. This gives the developers more flexibility when designing smart clients, but also makes it more complex to implement. Another disadvantage is that Web Services have a fairly high overhead because of the use of XML to transport data, which could give bad network utilization and cause poor performance of the smart client. [7]

The overall advantages and disadvantages of the Service-Oriented Architecture are summarized below:

Advantages:

- Caching and synchronization very flexible
- Platform independence
- Loose coupling
- Possibility of interacting with multiple disparate services
- Communication with database always through Business Logic layer
- Possibility to fit on small devices

Disadvantages:

- More complex implementation
- No built-in change tracking and synchronization
- Local data storage not provided by default
- Communication overhead can lead to poor network utilization

The current trend is that more and more systems are utilizing the Service-Oriented Architecture by using Web Services to make interaction between the different systems as easy and simple as possible. Today, standardized Web Services are a highly accepted communication mechanism and the amount of Web Services in the world is growing rapidly. This fact combined with the loosely coupled nature and the high flexibility of the Service-Oriented Architecture makes it a compelling design architecture when developing smart clients.

2.2.3 Summary

Two main architectures: the Data-Centric Architecture and the Service-Oriented Architecture have been analysed. The Service-Oriented Architecture will form the basis architecture of this master thesis project, because of its loosely coupled nature, flexibility and the fact that the described problem domain specifies that communication must be performed using Web Services.

However, the Data-Centric Architecture has some very interesting features, especially in the way data is handled locally and how changes to data results in a synchronization process. These features and concepts will be taken into account when providing a solution strategy in section 2.4.

2.3 Caching Web Services

Caching is a feature highly necessary for occasionally connected smart clients in a Service-Oriented Architecture, which is the basis architecture in this master thesis project. Without the possibility of caching Web Services, these smart client applications will have very limited functionality in an offline scenario.

Caching of data communication has some general advantages, as it approaches the following three issues:

- Availability
- Performance
- Scalability

Availability is the main cause for using a cache, as it makes the application functional even when the server is unreachable; network connection may be unavailable or the server may be down for some other reason.

The two other advantages, performance and scalability, are not the primary reason for using a cache to handle the offline scenario, but are side effects that should be utilized as much as possible. The performance of applications can be improved radically by using caching because data can be taken directly from the cache instead of retrieving the data from a server every time data is needed. This also improves the scalability of the overall system because the load on the server is reduced as fewer requests from the clients are required for them to operate.

SOA represents new challenges in relation to caching, which primarily is due to the diverse set of operations that such services provide and the fact that the SOA generally has been designed without regard to caching and hence offers little support for this.

Much research has been conducted in the area of file system, database and web page caching, and many techniques and strategies on how to solve the related caching problems are well documented and implemented. These traditional systems export relatively straight forward operations, such as read, write, open and close and cache managers for these systems are therefore rather simple to implement. This is not the case with the Service-Oriented Architecture, as each service exposes its' own unique interface. A cache for the Service-Oriented Architecture must somehow be able to categorize each operation exported by the service in order for the cache to handle the service correctly. This could indicate that there is a possibility that any SOA will require a specialized cache manager.

The current strategies for providing availability of files, databases and web pages are consequently not directly applicable to the Service-Oriented Architecture. Therefore, a new approach is required to provide cached access to Web Services in a Service-Oriented Architecture for occasionally connected applications. In the following subsections, two different techniques on how to cache Web Services will be discussed in more detail.

The first technique is based on a transparent client cache proxy, which is characterized by the ability that it can be applied to existing solutions without requiring any changes made to the client or the server. This technique is primarily based on research conducted in [2] and [3].

The second technique addresses the challenge of caching Web Services from another perspective by realizing that smart clients must be aware of the fact that they at times can be disconnected from the server in order to handle the offline scenario properly and give a good user experience. For this reason a built-in cache framework is considered. This technique requires that the caching mechanism must be built into the client, for which reason it cannot be utilized by existing clients, but only is relevant when developing new client solutions.

2.3.1 Transparent Client Cache Proxy

The idea behind making a transparent client cache proxy is to make an easy and interoperable solution to apply caching to an existing client which consumes Web Services. The client application will instead of communicating directly with the server's Web Services communicate with a caching proxy on the client side. The caching proxy will then be responsible for responding the client application with cached data and, if necessary, send requests to the Web Service when the client obtains connectivity. The solution is visualized in Figure 5 on page 18.

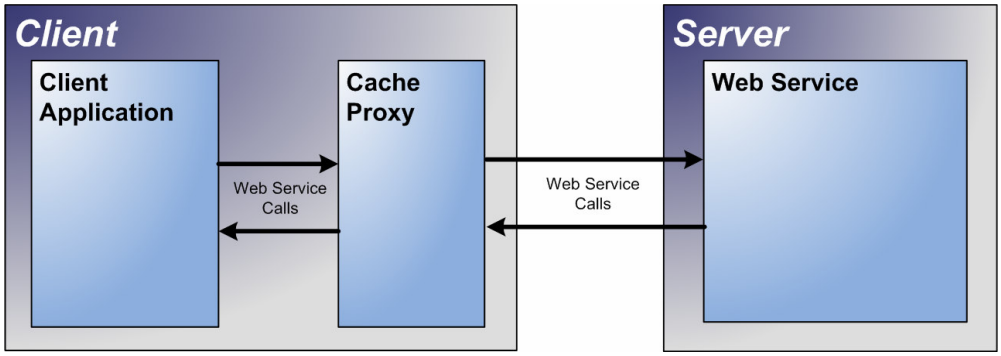


Figure 5: Architectural overview of the Transparent Client Cache Proxy [2]

The intention of using a seamless caching proxy is to make caching available to the client without having to change the client application, the Web Services or how they communicate with each other. These properties make it an out-of-the-box solution, which makes the client-side transparent cache proxy very appealing.

To get a better understanding of how the caching proxy works, Figure 6 on page 19 will be used to discuss the implementation in more detail. The caching proxy consists of a client interface and a server interface. When a request arrives from the client application on the client interface, the proxy has to decide whether it is dealing with a HTTP message or a Web Service SOAP message. HTTP messages are sent directly to the server interface because the caching proxy is intended for Web Service messages only.

In a connected situation, Web Service messages are sent to the connected module, which is responsible for sending the Web Service request to the server interface and cache the response for later use.

In a disconnected situation, Web Service requests are sent to the disconnected module which examines the cache for any valid responses and returns the response to the client interface, if possible. Furthermore the disconnected module places the request in a write-back queue for replay when the client can connect to the Web Service.

The thoughts behind the caching proxy are interesting, but implementing such a solution in a real world application has some major drawbacks, which cause some complex issues that need to be solved. Some of these issues occur when caching Web Service responses because

the caching proxy, who consumes a Web Service, is unaware of how to deal with particular Web Service calls. For this reason changes to the WSDL standard described in section 2.2.2 is necessary for the caching proxy to be fully transparent. To deal with these new challenges additional attributes has been proposed to the WSDL standard to comply with the lack of information in caching scenarios. However, the research in this area still has some major challenges which have not been solved. [2]

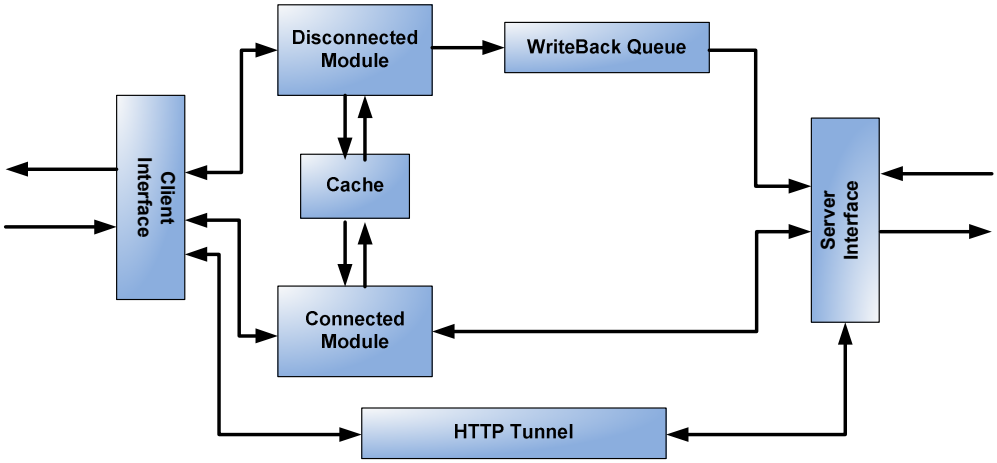


Figure 6: Components of the Transparent Client Cache Proxy [2]

The attributes in the WSDL document need to tell the following about each Web Service call:

- Is it cacheable?
- When should cached responses expire?
- Does the Web Service call need to be played back[2]?
- What should the default response in an offline scenario be?

The transparent client cache proxy needs to be aware of whether a particular Web Service call is cacheable or not. Caching some Web Service responses does not make sense, as these responses could be irrelevant at a later stage, e.g. a Web Service providing the current time. If a call is cacheable an expiration policy is needed to know when the cached data has become stale and must be refreshed to operate.

Replay ability is required, if a Web Service call permanently changes the state of the server. If this is the case the cache proxy will queue up the requests for execution at a later stage when the Web Service is available. The replay attribute is not a necessity but will lower bandwidth usage and server resource usage. A default response is necessary in case no response has been cached for a particular request and the caching proxy is offline. Then,

[2] If the Web Service call changes the state of the server, it needs to be replayed when connectivity is achieved.

there is a need for a default response when the caching proxy receives a request from the client application.

These properties are impossible for the cache proxy to determine by itself and must therefore somehow be specified by the Web Service in the WSDL document for the caching proxy to remain fully transparent.

Keeping consistency in the cache of the proxy is one of the more difficult issues to solve. Strong consistency is not feasible in a disconnected situation, because it is obvious that data at this stage cannot be updated by the cache. In such a situation, it is desirable that the proxy somehow should invalidate its cached data or make temporary local changes, so that subsequent responses would reflect that the performed changes are registered by the proxy. This situation and related aspects are discussed more deeply in [2]. The result of the research discussed in this paper is a prototype of a transparent client cache proxy which automatically invalidates cached data when data gets inconsistent because of changes made by the client. This invalidation of cached data is, however, not a feasible solution in the offline scenario, because the valuable cached data just expires. The paper discusses how to make a smarter cache consistency algorithm, but this is very troublesome and will extend the requirements of the WDSL standard even more.

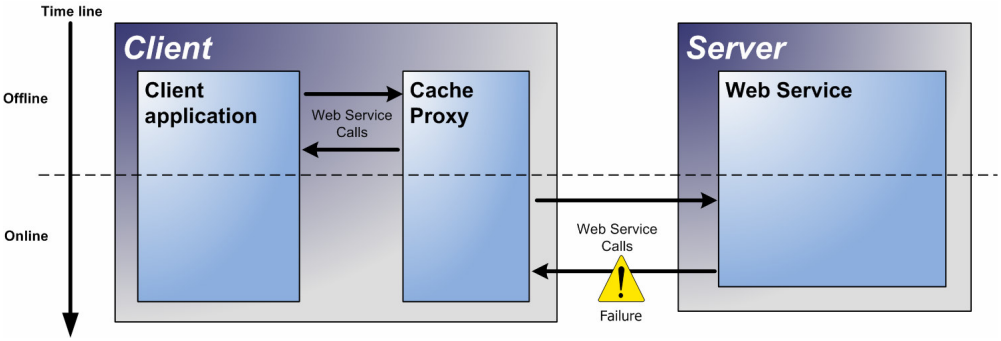


Figure 7: Conflict Resolution Issue

The caching proxy also has bad conflict resolution, as no standard for conflict resolution for Web Services exists. The conflict resolution issue is illustrated in Figure 7. If the client application performs an update operation when it is offline, it would receive a default response from the cache proxy telling the client that the request has been successfully processed. When the client gets online, the cache proxy will attempt to perform the request which it has held in a queue. If the request now fails, the cache proxy has no knowledge of how to deal with the failed request, and redirecting the failure to the client application is not possible. In such situations, the user experience of the application would be poor, as the user never gets the failure notifications.

The advantages and disadvantages of the transparent client cache proxy are as follows:

Advantages:

- Transparent out-of-the-box solution
- Very easy to implement in existing client application

Disadvantages:

- Bad data consistency
- Bad conflict resolution
- Bad user experience
- Changes to the WSDL standard are necessary for the cache proxy to become fully transparent

Generally, there are some major issues with this solution, which in business critical systems can have vital consequences and would require extensions to the WSDL standard to make the cache proxy fully transparent. For some less critical systems this caching solution could be very attractive because it delivers a transparent out-of-the-box solution which is very easy to implement, but would require the cache proxy to be customized for the situation.

2.3.2 Built-in Cache Framework

The concept behind the built-in cache framework solution is based on the assumption that smart client applications need to be mobile-aware in order to handle the disconnected scenario properly and to achieve a functional and user-friendly smart client application. Although the transparent client cache proxy solution is seen as an appealing solution in the sense that it offers offline support on existing client applications without alteration, it is not considered a feasible solution, because it is fundamentally limited in that the functionality needed to create correct and well-performing smart client applications in an intermittently-connected environment often requires the cooperation of both application and user. The built-in cache framework offers a better handling of service errors and conflict resolution issues compared with the transparent client cache proxy, which leads to more user-friendly smart client applications.

As smart clients share common design goals, they also share design features and techniques, which makes it suitable to utilize a framework that generalizes and optimizes the common cases as much as possible. A built-in cache framework provides the components and architecture necessary for smart clients to continue operation in a disconnected scenario, which is supported by caching functionality of Web Services in a Service-Oriented Architecture through a simplified application programming interface.

Research in this area has mainly been conducted in connection with the Rover Toolkit [22] and Microsoft's Offline Application Block [4]. The Rover Toolkit is a set of software tools for POSIX[3] compliant UNIX platforms that enables and simplifies the construction of mobile-aware applications with the use of Remote Procedure Calls as communication mechanism. Microsoft has in connection with their smart client technology analyzed how an intermittently-connected environment can be supported by these smart clients on the Windows platform. This research has resulted in an Offline Application Block that provides reusable code and recommendations that can be used to build mobile-aware applications. The Offline Application Block is based on the use of Web Services in a Service-Oriented Architecture, which makes it the most interesting of the two solutions and will for this reason be described in more details in the following paragraphs.

The Offline Application Block combined with the Service-Oriented Architecture makes it possible for smart clients to interact with whatever services are required. Rather than making direct changes to locally held data as in the Data-Centric Architecture, the Offline Application Block instead focuses on the service requests themselves. These service requests may lead to state changes on the client or the server, but such changes are seen as by-products of the service request and must therefore be handled separately.

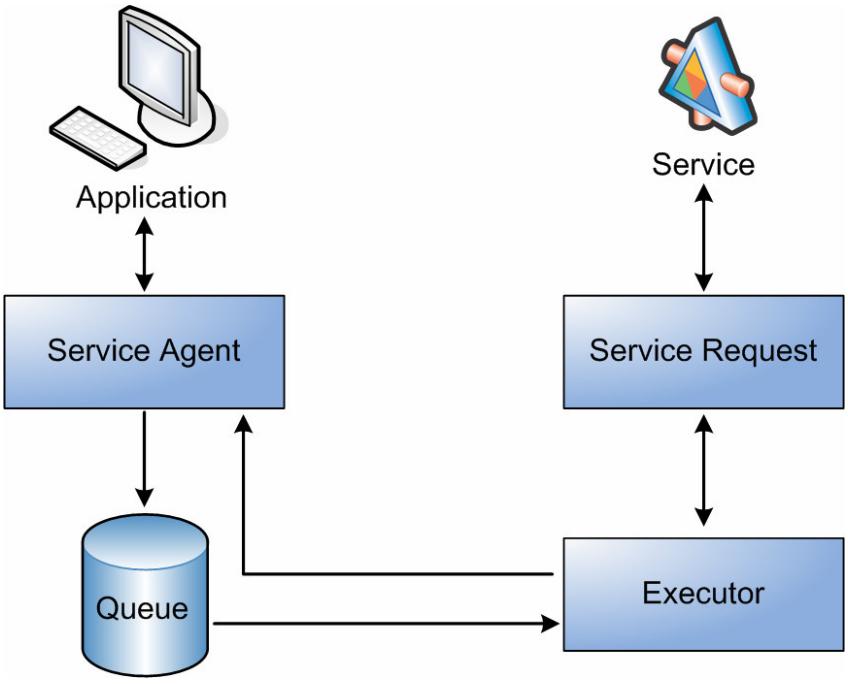


Figure 8: Infrastructure supporting the offline scenario [4]

[3] The Portable Operating System Interface (POSIX) is a set of standards developed by IEEE.

For the smart client to be able to support the offline scenario, the Offline Application Block provides an infrastructure that allows the details of the service requests to be stored locally so that they can be executed when the smart client obtains network connectivity. This infrastructure and its related elements are illustrated in Figure 8.

The infrastructure consists of the following four main elements:

- **Service Agent**
  Access to the different services is provided by service agents. These service agents manage all client interaction with the service and encapsulate all of the necessary logic to allow the smart client to create service requests.

- **Service Request**
  All the necessary details of the individual service requests are encapsulated in service request objects, which are persisted in the service request queue until the executor component is ready to process them. The service request object is responsible for making the actual service request.

- **Service Request Queue**
  The task of the queue is to function as a persistent storage for the service request objects.

- **Executor**
  The executor is responsible for taking the service request from the service request queue and executing them when the smart client obtains network connectivity. The executor also informs the service agent once the service request has been completed, so that the service agent can inform the smart client.

The Offline Application Block is founded on the basis of the described infrastructure and composed of the subsystems illustrated in Figure 9, which are loosely coupled components. Together, these subsystems provide the capabilities needed to detect the presence or absence of network connectivity, cache the required data and synchronize this data with the server when a network connection becomes available.

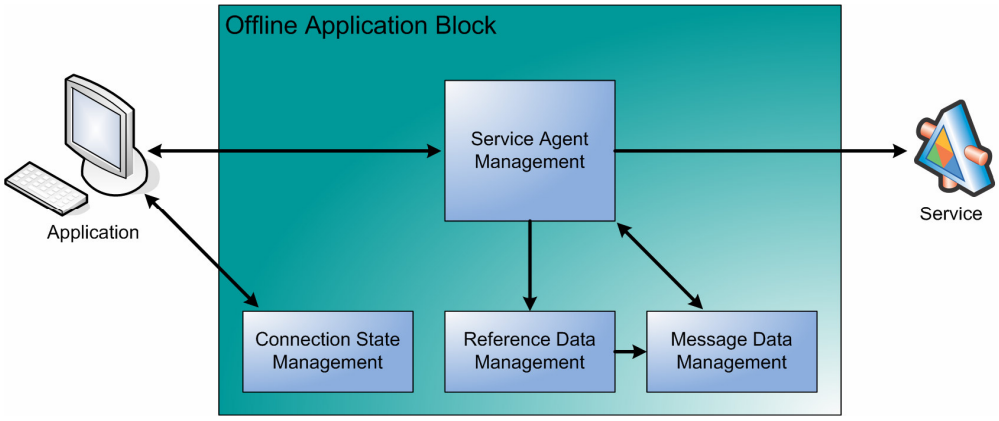


Figure 9: High-level view of the Offline Application Block architecture [4]

The individual subsystems are described more closely in the following paragraphs:

- **Connection State Management**
  This subsystem has the responsibility of detecting whether an application is online or offline. The Offline Application Block supports two ways of determining the connection state: manually or by an automated process. The automated process is the most advanced of the two ways that interprets the connectivity of multiple layers including the network layer and the actual application connection to detect the application's connection state. The Connection State Management subsystem changes the application's behaviour depending on the connection state.

- **Service Agent Management**
  The Service Agent Management subsystem interacts with the Message Data Management, Reference Data Management and the server. The subsystem has the responsibility of coordinating the individual service requests and returning task completion notifications back to the application.

- **Reference Data Management**
  The Reference Data Management subsystem works with the Service Agent Management and Message Data Management subsystems to download reference data from the server to the local computer. In most cases, this reference data is

read-only data, which is needed by the application to complete one or more workflows. The subsystem also has the responsibility of keeping the reference data current with the data on the server, which is done by storing messages in the Queue that is used to download the needed reference data. The Executor handles these messages by connecting to the relevant service when online and performing the actual download of the reference data. The Reference Data Management subsystem also provides the ability to secure the cached data by allowing encryption and decryption.

- **Message Data Management**
  Message data is the data that the application creates when offline in connection with a workflow process. This message data is stored by the Message Data Management in the Queue when offline, and handled by the Executor when the application obtains connectivity. The Executor handles the message data by removing it from the Queue and making the necessary Service Request to synchronize the message data with the server, and this data is then synchronized with the server.

Reference data is primarily seen as static data by the Offline Application Block that only occasionally need to be changed by the application. When locally cached data is changed, the Offline Application Block marks the changed data as dirty, which prevents the Offline Application Block from refreshing this data as it ages and thereby avoids overwriting the changed data with older values from other sources. After the changed data is synchronized with the remote service, the data is marked as clean and becomes eligible for refreshing.

The Reference Data Management subsystem uses another of Microsoft's application blocks called the Caching Application Block[4] to store the cached data. The Caching Application Block is part of the Enterprise Library for .NET Framework 2.0, and provides support for persisting of data using an isolated storage provided by the .NET Framework, a SQL Server, an in-memory storage or a memory-mapped file storage. Regardless of which storage mechanism used the Caching Application Block provides all the functionality needed to retrieve, add, and remove cached data. Configurable expiration and scavenging policies are also part of the Caching Application Block's functionality.

The Caching Application Block handles each data item separately using unique keys, which makes it easy to add and retrieve data items, but difficult to make more advanced data queries as supported in the Data-Centric Architecture. This data querying functionality must instead be handled manually by the developer, which results in implementation to be done to achieve this kind of functionality.

[4] The Caching Application Block can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/EntLibJan2006_CachingAppBlock.asp

The Offline Application Block provides an interesting basic architecture for a built-in cache framework. It provides some basic building blocks that give the developer a detailed level of information of what goes on in the Offline Application Block and gives the flexibility to make smart clients work with almost any service. Although the Offline Application Block is very flexible, the interaction between the different components is complex and difficult to understand. To use the Offline Application Block requires a good understanding and knowledge of its architecture and the different components' functionality, which makes the Offline Application difficult to use. The complex interaction also requires additional encapsulation which results in more implementation that needs to be done by the developer.

The advantages and disadvantages of the built-in cache framework are summarized below:

Advantages:

- Common design features and techniques are generalized
- More functional and user-friendly smart client application
- Better handling of service errors and conflict resolution issues

Disadvantages:

- Requires that clients are implemented using the built-in cache framework
- Not suitable for existing client applications

The built-in cache framework is a very interesting solution to the challenges related to caching Web Services in a Service-Oriented Architecture. The Offline Application Block provided by Microsoft represents interesting ideas on what functionalities a built-in cache framework should provide and how this can be achieved. The ability to create more correct and well-performing smart client applications than with the transparent client cache proxy makes it an appealing approach.

2.4 Solution Strategy

The two main design architectures, the Data-Centric and Service-Oriented, have been discussed in relation to developing a feasible offline capable solution for smart clients that must interact with existing systems. The two approaches have their advantages and disadvantages, but the Service-Oriented Architecture is chosen as the basic architecture, because of its loosely coupled nature, flexibility and the fact that the described problem domain specifies that communication must be performed using Web Services. In many situations, the Data-Centric Architecture is also not a feasible solution, because it requires direct access to the server's relational database. Many company policies will only allow

making changes to databases through the application layer, which makes the Data-Centric Architecture hard to utilize in these situation.

The problem with the Service-Oriented Architecture is though that it requires a lot of development effort to handle the local offline data. This is not needed with the Data-Centric Architecture, as the developer only needs to worry about making changes to local data and not calling any remote services, which helps simplify the implementation process.

The idea is to combine some of the advantages from the Data-Centric Architecture with the Service-Oriented Architecture in a Hybrid Architecture. Figure 10 illustrates this Hybrid Architecture, where the client side of the architecture now consists of a Local Data Storage and Data Mapping Logic. The Data Mapping Logic is responsible for mapping the individual Service Agents with the Local Data Storage. This mapping is much like the subscriptions known from the Data-Centric Architecture, which will be used by the client to synchronize its local data with the relevant Web Services.

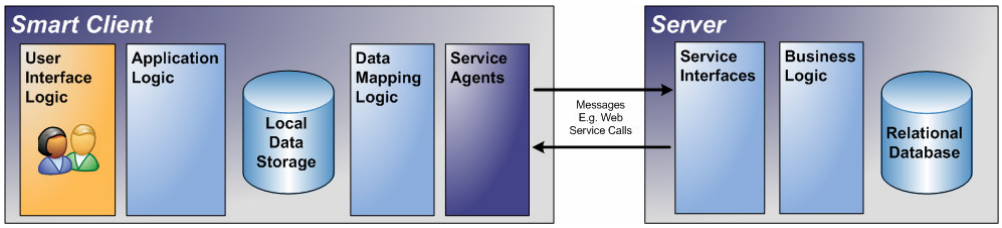


Figure 10: Architectural overview of the Hybrid Architecture

By combining the two described architectures in the Hybrid Architecture the following advantages will be utilized from the Data-Centric Architecture in the Service-Oriented Architecture:

- Caching and updating easy to implement
- Advanced data querying functionality supported
- Time-saving implementation

To achieve these advantages in the Hybrid Architecture a framework providing the needed functionality will be built. The framework must be able to cache Web Services, which have been examined in the two caching mechanisms; the transparent client cache proxy and the built-in cache have been examined. The transparent client cache proxy had some vital issues, as discussed, which made it less attractive and as a result it has been decided to focus on the built-in cache mechanism. The Offline Application Block will not be utilized directly, because it is too complex for the developer to utilize. Instead, it will be used as inspiration for making a new built-in cache which can help utilize some of the advantages of the Data-Centric Architecture to make a more simple framework. The combination of a hybrid architecture and a built-in cache is outlined in Figure 11 which represent the solution strategy

of this master thesis project. The built-in cache framework's ability to create more correct and well-performing smart client applications than with the transparent client cache proxy makes it the most appealing approach.

The Offline Application Block provided by Microsoft presents interesting ideas on what functionalities a built-in cache framework should provide and how this can be achieved. The concept behind the Offline Application Block is focused on the service requests themselves that may lead to state changes on the client, but such changes are seen as by-products of the service requests. This is a different approach than the concept of the chosen Hybrid Architecture, where changes are thought to be made directly to the data, which then causes execution of service requests.

The Offline Application Block will not be utilized directly, because it is not suitable for the Hybrid Architecture and requires that the developer must handle the local data in the implementation. Instead the Offline Application Block will be used as inspiration for designing a new Built-in Cache Framework, which can help utilize the described advantages of the Data-Centric Architecture to make a framework that fits into the Hybrid Architecture and is simple to use for the developer. The combination of the Hybrid Architecture and the Built-in Cache Framework is outlined in Figure 11, which represents the solution strategy of this master thesis. The framework that will be developed will be referred to as the Offline Framework.

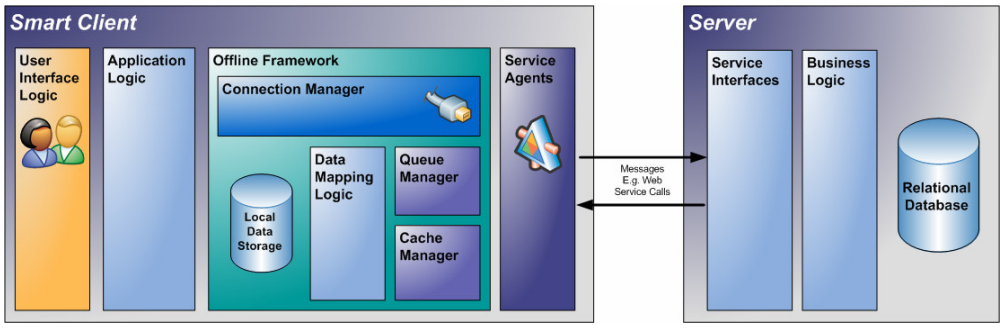


Figure 11: Solution Strategy

To fully understand Figure 11 an overview of the components in the Offline Framework will be made:

- **Connection Manager**

  The Connection Manager is responsible for constantly monitoring the connection state of the smart client application. On state changes the Connection Manager must inform the other components that the state has changed. In this way, the components are constantly aware of the current connection state.

- **Local Data Storage**
  The Local Data Storage is the physical data storage which consists of cached data. This storage should be easily accessible through the Offline Framework and support advanced data querying functionality.

- **Data Mapping Logic**
  The Data Mapping Logic is responsible for mapping changes made to the local data storage into Web Service requests, which is added to the queue handled by the Queue Manager. The Data Mapping Logic also contains information about what Web Services to utilize to refresh stale data.

- **Queue Manager**
  The Queue Manager is responsible for queuing outgoing Web Service requests in a queue, which is executed when the connection state of the smart client changes to online.

- **Cache Manager**
  The Cache Manager is responsible for keeping data up-to-date by using mapping information given by the Data Mapping Logic.

This overview should give an understanding of the necessity of each component in the Offline Framework. The most important challenges with the Offline Framework will be analysed and discussed in more detail in chapter 3.

2.5 Summary

This chapter has provided a description of the problem domain with specifications of the framework needed by PwC. The relevant design architectures for smart client solutions have been discussed and the Service-Oriented Architecture was chosen as the basic architecture. The analysis of the Data-Centric Architecture presented some interesting ideas and concepts.

Caching of Web Services was analyzed using existing theory. This analysis presented a Transparent Client Cache Proxy and a Built-in Cache Framework. Both approaches were interesting ideas, but the Built-in Cache Framework was seen as the most appealing approach for the framework needed.

The analysis of design architectures and caching Web Services has resulted in a solution strategy, which will be the foundation for further analysis, design and implementation of the Offline Framework in this master thesis.

3 Challenges of the Offline Framework

This chapter analyzes the most important challenges of the Offline Framework, which will be based on the solution strategy described in section 2.4. The two challenges that will be discussed further are how to handle the local data storage and how to perform efficient synchronization and replication.

The purpose of this chapter is to:

- analyze how the local data storage can be provided
- provide a strategy for the local data storage
- analyze how to handle synchronization and replication efficiently

After having read this chapter, the reader will have obtained the necessary understanding of the strategies chosen for the local data storage and synchronization and replication. These strategies will be used later in the design and implementation of the Offline Framework, which will be described in more detail in chapter 5.

3.1 Local Data Storage

This section analyzes the local data storage component of the Offline Framework as illustrated in Figure 12. The local data storage is one of the most important capabilities that the Offline Framework must provide, without this capability an offline scenario would be very difficult to handle for an occasionally connected application. Providing this local data storage can be achieved in many different ways, which is described in more detail in the following subsections.

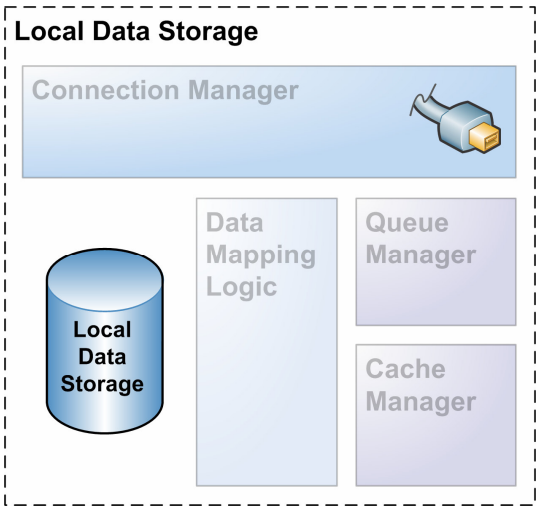


Figure 12: Local Data Storage

The local data storage should have the following properties:

- **Fast**
  Good performance of the local data storage is essential for smart clients, as it will have great impact on the user experience. Searching through and manipulating with the local data should therefore be relatively fast operations.

- **Scalable**
  The local data storage must be able to scale well so that occasionally connected applications developed using the offline framework can be extended with more functionality and local data, without being limited to a certain amount of data. The amount of memory used by the local data storage must therefore be held at a minimum.

- **Secure**
  As the local data storage in many cases will contain sensitive business information, such as customer information, it needs to be secured to prevent unauthorized access. The possibility for an authorized user to easily extract all data from the local storage should also be held at a minimum, because many companies want to minimize their employees' possibility to achieve a complete view of all data used in their occasionally connected application, especially if it contains sensitive business information. This concept is also known as "principle of least privilege", which requires that a user only must be able to access such information and resources that are immediately necessary. [23]

- **Easy-to-use**
  Because the local data storage is such a central component of the Offline Framework, it is important that it is as easy to use for the developer as possible. The primary principle when designing the local data storage should therefore be based on *simplicity* such that the behaviour exposed by the Offline Framework is as obvious and consistent as possible.

The following strategies have been considered in the analysis process to fulfil the specified requirements as best as possible:

- In-Memory
- Simple File Based
- Embedded Database

These three different strategies for implementing the local data storage in the Offline Framework will be discussed further in the following subsections, where advantages and disadvantages of each strategy are analyzed more deeply.

3.1.1 In-Memory

The strategy of having all the local data storage in-memory is a relatively simple and straightforward solution, because at some point of time all data needs to be fetched into memory when the different Web Services are called, so it is in some way natural to keep the local data storage in-memory.

The advantages of this strategy are that it is relatively simple to implement as changes made to the local data easily can be tracked and that the process of replicating and synchronizing the local data storage can be performed by simple and fast operations. The fact that the local data storage is located in-memory also gives fast access to the data.

At some point of time, the local data, however, needs to be persisted on disk in order for the Offline Framework to be able to reload the data in a disconnected scenario. This can be achieved by either serializing the data as objects, saving the data in binary format or as XML, which can be encrypted making the local data storage secure. A straightforward solution is to load the persisted data at program start-up and save the in-memory data when the occasionally connected application is closed. This will work fine as long as the application does not close unexpectedly, which would mean loss of data. To avoid this problem, the in-memory local data storage could persist all its data every time a change is made, but this would make the framework very disk- and resource-intensive, which is not wanted. The local data storage therefore has to rely on the occasionally connected application to close properly each time in order not to lose any of the changes made.

The fact that all of the local data storage is located in-memory makes it a very memory intensive solution. This might be a fine and effective solution when dealing with small amounts of data, but because of the data is being held as objects in memory even relatively small amounts of data will require a lot of memory. As a consequence the in-memory strategy is not a scalable solution, which is its greatest disadvantage.

Another disadvantage is that the in-memory strategy lacks data querying functionality known from database systems like joining tables, which in many cases is needed when working with the local data. This functionality can of course be achieved manually, but will require more implementation to be made by the developer using the Offline Framework, which conflicts with the easy-to-use requirement.

To summarize the advantages and disadvantages of the in-memory strategy are listed below:

Advantages:

- Easy to implement
- Fast replication and synchronization
- Fast access to data

Disadvantages:

- Not a scalable solution
- Poor data querying functionality
- Problematic persisting of data

The in-memory strategy is discarded as a solution for the local data storage because the advantages with fast access to data and easy implementation does not compensate for the disadvantages with poor scalability and limited data querying functionality.

3.1.2 Simple File Based

Another strategy is to base the local data storage on simple files containing all the data needed by the Offline Framework. The data could naturally be saved as XML or in binary format to save disk space and be secured by encrypting the files.

The primary advantage of this strategy is that it solves the scalability problem described with the in-memory strategy as all data now are kept in files located on disk and only the necessary data is loaded into memory when needed by the occasionally connected application. The simple filed based strategy also makes it possible for the local data storage to be run on small devices as it is not very memory intensive and does not depend on other software components that could be a limiting factor.

Even though the strategy makes use of simple files to store the local data it can be a challenging task to find a good and effective solution on how to save only changes made to the data, especially if the files used for storage also need to be encrypted. Another disadvantage is that the strategy will have a bad performance when searching through the local data due to the fact without any supporting information this would have to be performed linearly. To solve this problem some kind of indexing functionality as known from database systems would have to be implemented, but this is not a simple task, for which reason much time and many resources would have to be used to achieve this functionality. The strategy stills lacks data querying functionality like joining different tables as the in-memory strategy does.

The advantages and disadvantages of the simple file based strategy are summarized below:

Advantages:

- More scalable handling of data
- Possibility to run on small devices

Disadvantages:

- Bad performance
- Poor data querying functionality

The simple file based strategy is discarded as a solution because of its bad performance and that it still lacks basis data querying functionality needed by the developer. So far, the analysis has indicated that the local data storage needs to support the most basis database functionalities, such as indexing and joining of tables. Therefore, the last strategy analyzed will be based on using an embedded database as the local data storage.

3.1.3 Embedded Database

The strategy of having the local data stored in an embedded database solves many of the described problems with the in-memory and simple file based strategies. The advantage with this strategy is that the embedded database is optimized to handle data effectively, which makes it a much more scalable solution than the two other strategies. The use of indexing and other database techniques ensures high performance of the local data storage, and the built-in data querying functionality simplifies the implementation that needs to be made by the developer using the Offline Framework.

The disadvantages of this strategy is mainly that the Offline Framework will be dependant on a third-party embedded database solution, which in many cases will impose limitations on the circumstances the Offline Framework that can be used and typically result in larger installation size and more complex deployment. The third-party embedded database normally impose restrictions on what devices it can run on, which typically implies that it cannot be run on small devices. The security of the embedded database strategy is very dependent on how the embedded database is implemented and the security options provided. Encryption of the data is typically not supported and therefore problematic to achieve. Access to the data in the embedded database is normally controlled at file level, which implies that users need read access to the data file and therefore indirectly have access to all data contained in the embedded database.

To summarize the advantages and disadvantages of the embedded database strategy are listed below:

Advantages:

- Scalable solution
- High performance
- Built-in data querying functionality

Disadvantages:

- Requires third-party software
- More complex deployment
- Limited use on small devices
- Problematic encryption of data

The embedded database strategy is very appealing as it provides the needed data querying functionality known from the Data-Centric Architecture combined with good scalability and high performance.

3.1.4 Chosen Strategy

The strategy that mostly fulfils the specified properties is the embedded database strategy as it is fast, scalable and easy to use for the developer because of the built-in data querying functionality. The security property is the only property, which is not fully satisfied by the embedded database strategy for the reason among others that encryption of the local data with this strategy is problematic.

The disadvantage that the embedded database strategy requires third-party software, which can lead to a more complex deployment, is not seen as a critical factor, but as a challenge that must be taken into account when designing the offline framework. The limited use on small devices is not seen as a relevant limitation, because the Offline Framework is not supposed to be used with such small devices.

The advantages are estimated to compensate for the disadvantages, and the embedded database strategy is therefore chosen as the strategy for the local data storage.

3.2 Synchronization and Replication

This section analyzes how efficient synchronization and replication can be achieved in the Offline Framework, which is a central process of the framework. The components involved are Connection Manager, Data Mapping Logic, Queue Manager and Cache Manager as illustrated in Figure 13, where these components are highlighted.

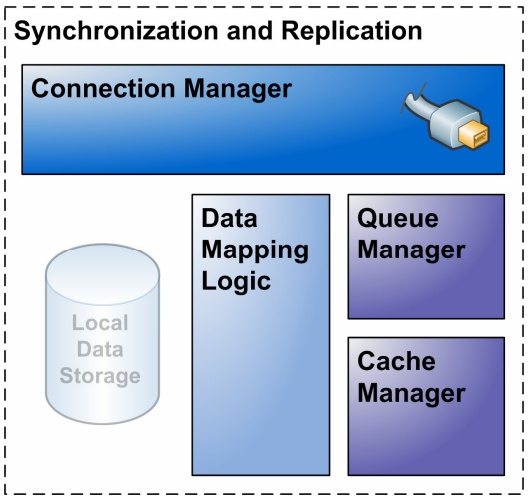


Figure 13: Synchronization and replication

The process of data synchronization and replication is a process that keeps two systems' data equivalent by exchanging information. The two terms synchronization and replication are in this master thesis defined as follows:

- **Replication:** "A process where all data is copied from one entity to another entity to keep the two entities equivalent".

- **Synchronization:** "A process where only the data changes are exchanged between two entities to keep the two entities equivalent".

The Service-Oriented Architecture is very flexible by its nature and does not have any standard mechanism for synchronizing or replicating the data exposed by the individual Web Services. The following considerations in relation to obtaining synchronization and replication in the Offline Framework are based on the Web Services having a CRUD[5] design. To illustrate how such a CRUD design affects a Web Service's methods, an example will be used. The example is a Web Service used to manage employee information. Utilizing the CRUD design, the Web Service's methods could look like the following:

- **Create**

  A method to create the individual employees must exist and could look like this: createEmployee(name, address, salary), which returns an employee number for the employee created.

[5] CRUD – Create, Read, Update and Delete

- **Read**

  It must somehow be possible to retrieve all information on the employees. This could be achieved by having a getEmployees() method, which return a list containing all the employees' information.

- **Update**

  Updating the employees' information could be achieved by having a single method for updating all of the information at once, for example updateEmployee(employeeNumber, name, address, salary),
  or by using multiple methods for each value, such as
  updateName(employeeNumber, name),
  updateAddress(employeeNumber, address) and
  updateSalary(employeeNumber, salary), where the employee number is used as key.

- **Delete**

  Deletion of a employee in the system could be achieved by having a deleteEmployee(employeeNumber) method, where the employee number is used as key.

The Service-Oriented Architecture is still very flexible even when the Web Service methods are limited to be based on the CRUD design. The Data Mapping Logic's responsibility is to map the CRUD-based Web Service methods to the Local Data Storage, making it is possible for the Queue Manager and Cache Manager to handle the synchronization and replication of the local data with the use of the connection state provided by the Connection Manager.

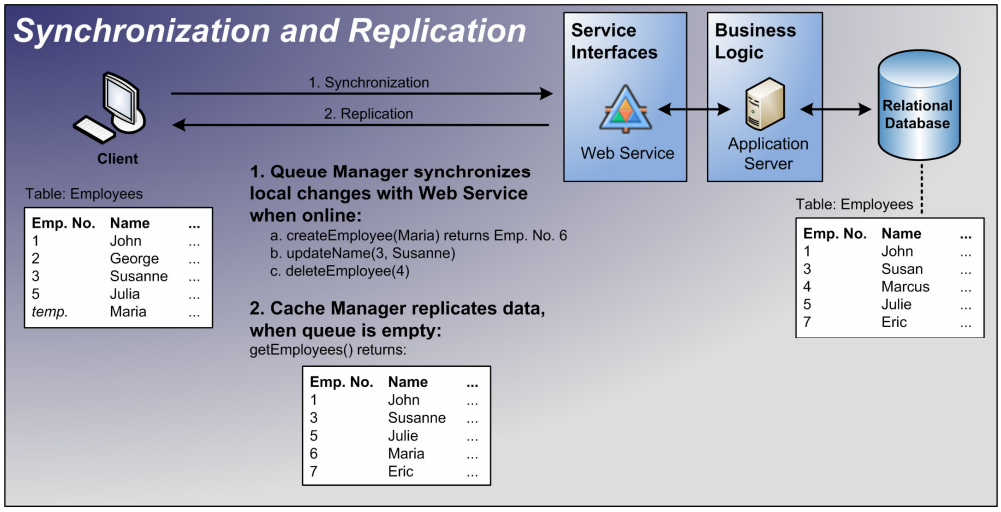


Figure 14: Synchronization and replication using CRUD based Web Services

Figure 14 illustrates how synchronization and replication can be achieved in the Offline Framework using CRUD-based Web Services. The figure illustrates that the client application in an offline state has made some changes to its locally held data. An employee named Maria was created, and because the employee numbers in this scenario are generated on the server, a temporary employee number is created on the client. This temporary employee number will be replaced by the correct employee number, when the client obtains connectivity and the Queue Manager executes. The name of the employee with employee number 4 is changed from Susan to Susanne, and the employee Marcus is deleted.

In the meantime, other users have made changes to the data on the server. The employee George has been deleted, the name of the employee with employee number 5 has been changed from Julia to Julie, and the employee Eric has been created.

The Queue Manager's task is to queue the changes made to the local data storage by utilizing the information held in the Data Mapping Logic. When the client obtains connectivity, the Queue Manager then synchronizes the local changes with the relevant Web Services as illustrated in the figure.

The Cache Manager's task is to update the local data storage in a given interval to keep the client's data as consistent as possible with the data on the server. The only way of doing this with the described CRUD design is to utilize the read method to retrieve a full list of the employees from the server, which contains the changes made by the Queue Manager. The locally held table of employees is then overwritten by the list retrieved, and the local data storage on the client is now consistent with the data held on the server.

The replication performed by the Cache Manager is a simple process and can be used with Web Services that follow the simple CRUD design, but it has some vital disadvantages. These disadvantages are mainly that the replication performed is far from scalable as all data needs to be retrieved and that it requires more processing on the client to update its local data storage. To solve this problem the following subsection will discuss possible solutions for optimizing the replication process performed by the Cache Manager.

3.2.1 Optimizing Replication

There are many different ways of optimizing the replication process performed by the Cache Manager because of the fact that Web Services can be implemented in very different ways. This subsection will consider solutions to optimizing replication, which are based on:

- using a Change Tracking Service
- using hash values
- using timestamps with special requirements to the data

The first solution of using a Change Tracking Service is illustrated in Figure 15 on page 40, which is based on the same scenario as the one used for Figure 14. The figure illustrates

that the Change Tracking Service is added as an extra layer between the Application Server and the Web Service. The Change Tracking Service's task is to track the changes made to the data on the server using metadata[6], which can be utilized by the Web Service to send only the changes necessary to update the client's data to make it consistent with the server's data. The Queue Manager handles the local changes in the same way as described earlier, but the Cache Manager now performs a synchronization instead of a replication, as it only receives the changes from the server containing the necessary metadata, which can be combined with the local data to make it consistent with the server's data.

The solution improves the process of synchronizing the local data storage with the server's data as it utilizes two-way synchronization. This means that only changes have to be propagated in both directions from the client to the Web Service, which makes the synchronization process more scalable to the amount of data and furthermore scales better to the amount of users with offline clients that need to exchange data frequently.

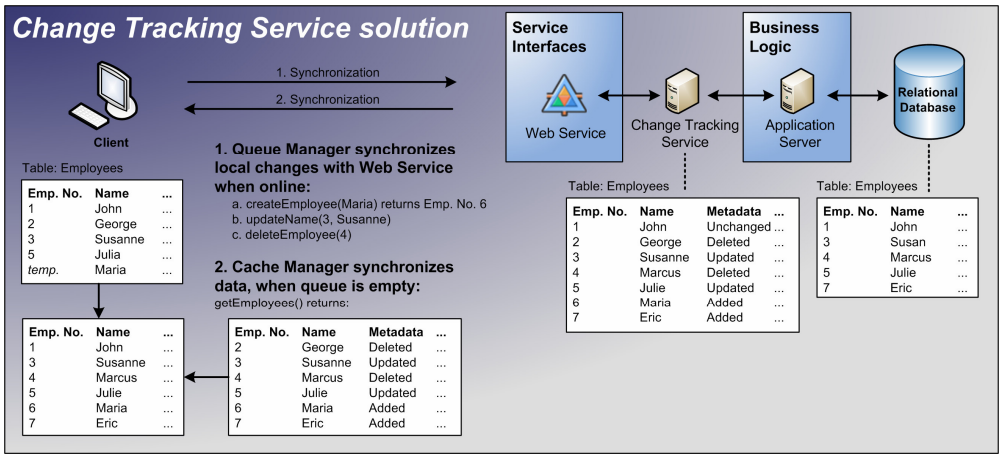


Figure 15: Optimizing the replication process using the Change Tracking Service solution

The Change Tracking Service, however, adds more complexity to the implementation, as it can be problematic to achieve the desired change tracking functionality, especially in existing systems that were not designed to support change tracking. The fact that the solution adds advanced requirements to the implementation on the server side makes it difficult to accomplish within the scope of this master thesis project.

The second solution is based on using hash values to optimize the replication process performed by the Cache Manager. The concept is to utilize the hash values to determine, if any changes have been made to the data on the server and only perform a replication, if this is the case. In this way, bandwidth and processing power on the client can be saved. The solution is illustrated in Figure 16.

[6] Definition for metadata: Data that is used to describe other data.

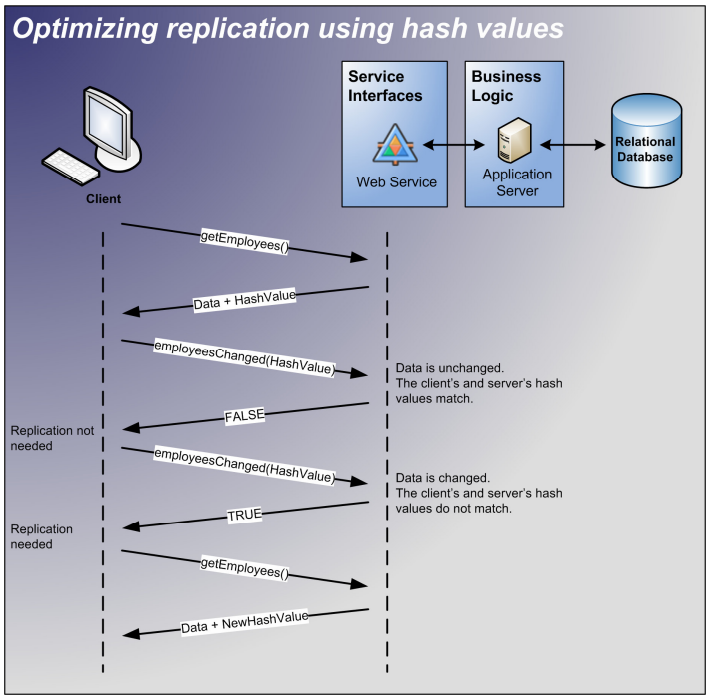


Figure 16: Optimizing replication using hash values

Every time data is replicated from the Web Service to the client, an additional hash value is encapsulated in the Web Service response. The hash value is unique for the current table of data replicated to the client and is used at a latter point to determine whether data on the server side has changed by sending the hash value back to the server side. In that way, the client can easily check if changes have occurred on the server without transferring large amounts of data. The solution does not improve the actual replication, but ensures that it is only performed when necessary. The simplicity of the solution makes it easy to implement on the server side, which makes it an appealing solution for optimizing the replication process performed by the Cache Manager.

The third solution is based on the use of timestamps to optimize the replication process performed by the Cache Manager. The solution has two requirements to the data that needs to be replicated; there must be timestamps on every record to indicate time of creation and modification and records must not be deleted. If these two requirements are fulfilled, the solution can be utilized.

The concept of this simple timestamp solution is that, when the client wants to replicate data from a Web Service, a timestamp indicating the time of last replication is added to the request. This last time of replication can then be utilized by the server to retrieve the records created or modified since this time, which is sent to the client. The client then updates the records, which already exist in the local data storage and add those that do not exist. If the

client has never replicated the data from the Web Service before, a special value is used as timestamp, which tells the server to send all data to the client.

The solution is appealing because it is relatively easy to implement, if the requirements are fulfilled, and improves the performance of the replication as only the records created and modified are transferred by the Web Service.

3.2.2 Conflict Detection and Resolution

Conflict detection and resolution are very important aspects of the synchronization process, because conflict detection and resolution are the logic deciding what data to persist when the same data on the offline client and the server has been changed since the latest synchronization. The action to take on different conflict scenarios depends on what type of data we are dealing with.

Within the area of database synchronization, the two terminologies pessimistic and optimistic data concurrency is widely used. Pessimistic data concurrency is a well-known technique to effectively prevent data concurrency conflicts. With pessimistic data concurrency one client is able to keep a lock over an amount of data to prevent other clients from altering this data. When the client has completed modifying the data, the lock can be released, so other clients are able to change the data. The pessimistic data concurrency approach is illustrated in Figure 17. [1]

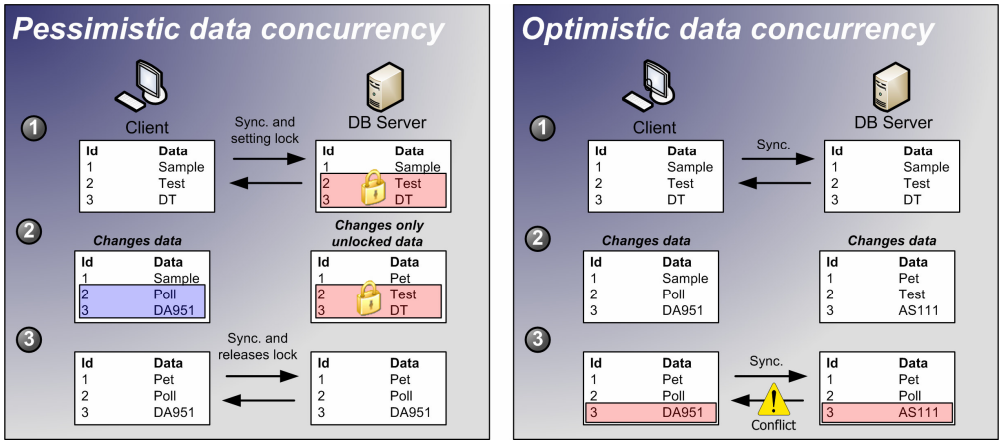


Figure 17: Illustrates pessimistic and optimistic data concurrency

The greatest advantage of locking data is that data concurrency conflicts are prevented, because it is guaranteed that only one client can make changes to shared data at a time. Despite of this great advantage the pessimistic approach has some major drawbacks, because locking data affects scalability and usability, as clients trying to change locked data need to wait for a period of time before the data is unlocked. This has especially major drawbacks in systems in which the client will be taken offline for longer periods of time.

The optimistic approach does not use lock mechanisms, and changes to the same data may occur on more clients at the same time. The approach is called optimistic, because instead of taking any special considerations, the approach tries to resolve any detected conflicts at the final synchronization stage, which is illustrated in Figure 17. The optimistic approach is more feasible for offline capable clients, because these clients typically are offline for periods of time that makes the pessimistic approach unsuitable. [1]

The matrix illustrated in Figure 18 shows the different possible conflicts that can occur when performing an optimistic synchronization of the two databases; DB A and DB B. One of the most common types of conflict is the concurrent update, which also was illustrated in Figure 17.

DB B \ DB A	New	Deleted	Updated	Synchronized/ Unchanged	Not Existing
New	Conflict	Conflict	Conflict	Conflict	B replaces A
Deleted	Conflict		Conflict	Delete target	
Updated	Conflict	Conflict	Conflict	B replaces A	B replaces A
Synchronized/ Unchanged	Conflict	Delete target	A replaces B		B replaces A
Not Existing	A replaces B		A replaces B	A replaces B	

Figure 18: Conflicts that can occur in an optimistic synchronization process [20]

To detect and resolve conflicts when they occur, some additional metadata is required to know what have happened to the data and when it has happened.

Possible useful metadata to detect and resolve conflicts could be:

- Timestamps at row level
- Sequence numbered updates
- Row state (Unchanged, New, Updated, Deleted)

Timestamps at row level can keep track of when the row was changed, and in combination with a row state it is possible to detect conflicts. It is important that the different clients and the server synchronize their internal time in order to obtain correct conflict resolution. A sequence number in combination with a row state can also be very useful, but will at the conflict resolution stage set some limitations at the decision making, because the timestamp information is not available.

To resolve conflicts it is in some cases most appropriate to ask the user about what data should be persisted and in other cases automated resolution logic is most appropriate to

decide what action to take. Methods for conflict resolution can vary depending on which method that is appropriate for the concrete data.

Possible methods for conflict resolution could be:

- Timestamp based conflict resolution (last update wins)
- Server or client side updates always wins
- Client user or server administrator makes a decision

Timestamp based conflict resolution can in many situations be a suitable reconciliation technique, because the latest update is often the most updated data.

Another technique is to decide that the server or the client overrules any conflicts that may arise. This technique is very simple to implement but can have unwanted outcomes under some circumstances.

The manual approach to conflict resolution is to ask the user or the server administrator what to do when conflicts occur. In this way, it is possible to make sure that data is persisted correctly. The disadvantage of this approach is that the manual decision making can be very annoying for the user.

An appealing solution for the Service-Oriented Architecture is that conflict detection and resolution is made on the server side, and error messages are sent to the clients, when synchronization for some reason is impossible. This error information can then be shown to the user, which can perform actions that will either solve the problem or somehow avoid the conflict of happening.

3.3 Summary

Three different data storage strategies have been analysed by comparing these strategies' advantages and disadvantages. The embedded database strategy has been chosen, because it is fast, scalable and easy to use for the developer. A further analysis of which embedded database technology to utilize in the actual implementation of the Offline Framework will be made in section 4.2.

An analysis on how to achieve efficient synchronization and replication in the Offline Framework has been provided. The ideas and concept described in this analysis will be taken into account when the Offline Framework is implemented, which is described in chapter 5.

4 Technology Analysis

This chapter analyzes relevant technologies utilized to develop the Offline Framework. Two areas that must be fulfilled to develop the Offline Framework within the timeframe of this master thesis are local data storage and a communication framework. These two types of components require a lot of effort to develop and are out of scope for this project to develop from scratch.

The purpose of this chapter is to:

- analyze relevant technologies which can be utilized
- analyze how these technologies can help solve the problem domain

The following subsections focus on the communication framework Windows Communication Framework followed by how to handle the local data storage.

4.1 Windows Communication Foundation

Windows Communication Foundation[7] (WCF) is a comprehensive communication framework for building distributed applications. It is still in the stage of development and offers a structured programming model which is independent on the transportation used. WCF supports a wide range of communication technologies, such as Web Service but primarily Microsoft based technologies, such as the Web Service standard and the Microsoft based technologies COM+, MSMQ, .NET Remoting. Even though there are a number of communication protocols available ready for use, the framework is designed for a high degree of customization, which makes it possible to implement custom communication protocols. In this chapter the main focus is on Web Services in connection with WCF because the Web Service protocol is an open standard which furthermore is a platform-independent communication protocol.

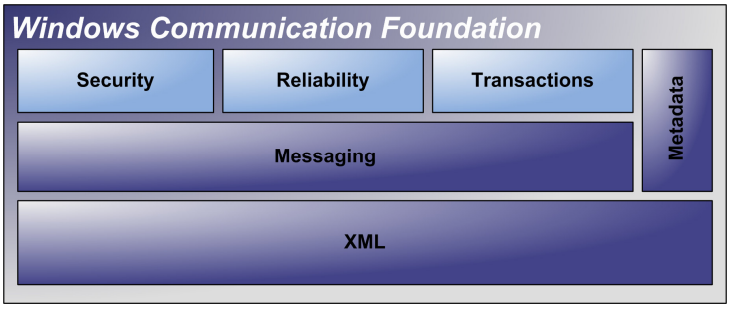


Figure 19: Modular support of Security, Reliability and Transactions

[7] Formerly known under the codename "Indigo".

The latest standards within Web Services have been implemented in the WCF framework, which supports reliable and transaction-based Web Service interactions. Also the security in WCF has been drastically improved compared to current communication frameworks and can be easily implemented by the developer because of the unified programming model. Figure 19 illustrates how security, reliability and transaction support has been built on top of the existing Web Service standard in a modular way, which makes it easy to add the support of these features to existing Web Services.

4.1.1 Architectural Overview

In this section, an introduction to the overall architecture of WCF will be presented to make the reader familiar with the new communication framework. To enable communication between two parties an Endpoint at each party must be configured. Each Endpoint contains information on where, how and what to connect to. The Address within an Endpoint consists of information on where to connect to. Figure 20 illustrates a caller which connects to a service. [16]

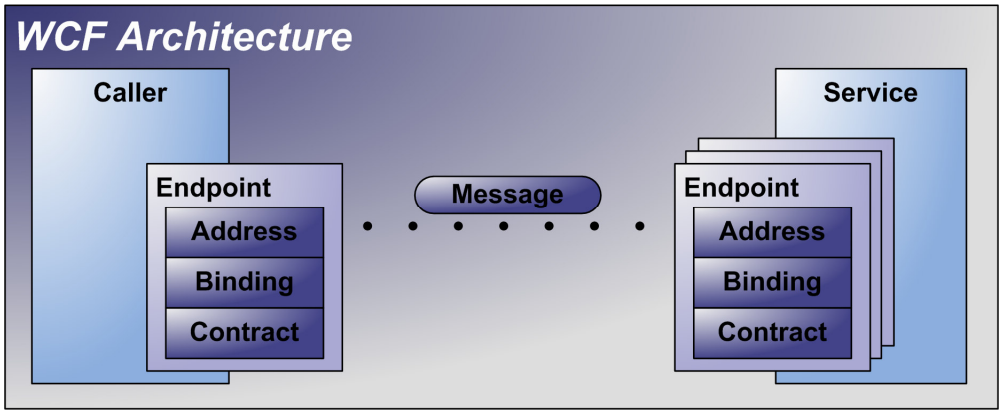


Figure 20: Architectural overview of Windows Communication Foundation [17]

The Binding consists of information on how a connection is made, such as:

- What transport is used to communicate (e.g. Web Service, .NET Remoting)?
- What data encoding is used?
- What type of security is used?
- Is reliable messaging enabled?
- Are transactions used?
- Should communication be streamed or buffered?

The contract specifies what operations are made available by the service for servicing the caller. Each operation in a contract defines the actual semantic of the messages exchanged, which is needed for the caller and the service to understand the meaning of the data exchanged.

The architecture of the Windows Communication Foundation is very flexible, which means that almost any communication protocol can be modelled using this framework.

4.1.2 Reliability

The Web Service standard WS-ReliableMessaging is a standard which is built on top of existing Web Service standards. WS-ReliableMessaging is defined by IBM and Microsoft as "A protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures". [8]

The absence of a standard for reliable messaging has for many years been a crucial challenge for developers making business critical applications with the use of Web Services. This lack of support in the Web Service standard have brought along non-proprietary application-level solutions to make some limited support for reliability. These solutions have the side effect that the Web Service consumer has to implement support for non-proprietary reliability solutions which is time consuming and hard to implement. For Web Services to be the main communication protocol in the future a standard for reliability is a necessity, because many enterprises depending on business critical systems cannot rely on unreliable protocols, where there is no knowledge of whether a request was executed or not. This scenario can most likely leave the service in an inconsistent state.

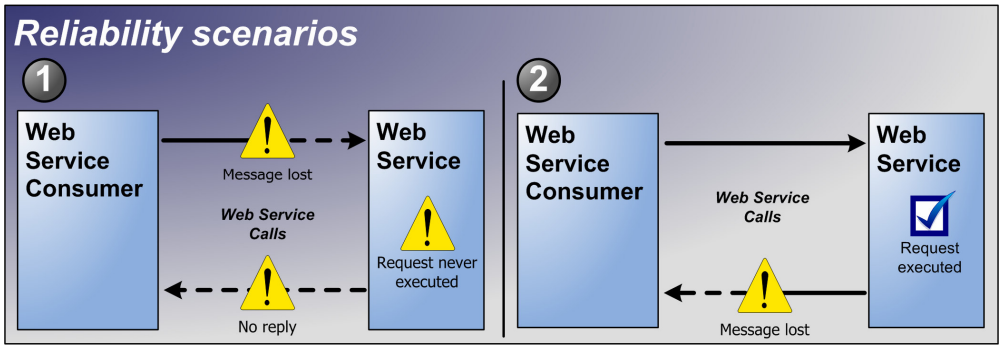


Figure 21: Reliability scenarios

To illustrate the issue more carefully two scenarios are presented in Figure 21. In scenario one, the Web Service consumer sends a request to be executed by the Web Service, but occasionally the message somehow was lost on the way to the Web Service, which never received the request and therefore obviously was not able to answer the request. So the result of this scenario is that the Web Service consumer times out.

In the scenario two, the request from the Web Service consumer is sent to the Web Service, which executes the request alright. The Web Service replies with a valid response, which somehow gets lost and is never received by the Web Service consumer. The result is a timeout identical to scenario one. The main point of describing these two scenarios is to illustrate that the Web Service consumer does not have any knowledge of whether the request has been executed or not, as both requests timeout. The obvious countermove to this timeout is to resend the request and hopefully get a response, but when resending a request it is possible that the request already has been executed and for that reason gets executed twice. If the Web Service request e.g. adds a specific amount to a row in the database of the Web Service, this could cause the Web Service to add the specified amount twice, which was not the intention of the Web Service consumer.

As discussed it is highly important to define a standard, which correctly handles these issues. Even though these issues do not arise particular often, it is a major weakness of standard Web Services. The standard WS-ReliableMessaging has been implemented in WCF and is very easy to take advantage of as it is merely a configuration option.

4.1.3 Transactions

WS-Transaction [9] is another new standard, which has been made to improve Web Services for use in connection with business critical operations. Figure 22 illustrates a scenario, in which transactions in Web Services are critical to maintain consistency among the different systems.

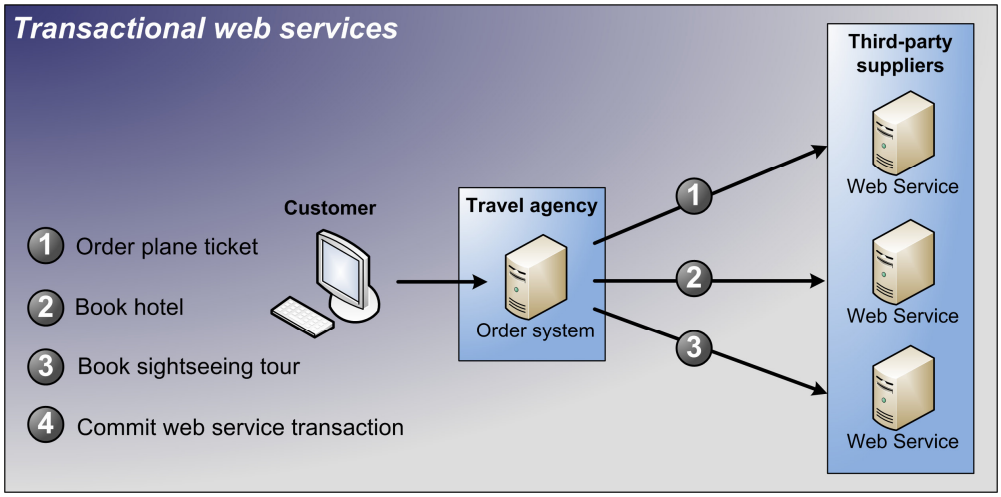


Figure 22: Transactional Web Services example

The scenario illustrates a person booking a vacation. This person has chosen a destination, a hotel and one or more sightseeing tours. When the person presses the order button the items should be ordered straight away at third-party suppliers. But this is not a

straight forward operation, because one of the orders at the third-party supplier could fail for many different reasons, e.g. the plane could have been overbooked. The situation where not all orders have been completed is a situation that must be prevented, because the customer cannot use a hotel room without a plane ticket. In such a scenario, the support of transactions is a necessity, because all three orders must be performed as one atomic action.

4.1.4 Security

Security is a very important subject in relation to making an offline framework, because it is necessary to communicate with a central server. As earlier stated, communication must be secured and for that reason the following subsections will make an overview of the security in WCF. Security is not the main focus of this project, for which reason the chapters will not go deeply into the security of WCF, but only outline the most important and relevant authentication options and communication security options.

4.1.4.1 Authentication & Authorization

Security of Web Services has been improved with the new standard WS-Security, which is supported by all major companies within the Web Service area. WS-Security has been fully implemented in WCF and supports mainly three different security mechanisms:

- Token-based security
- Microsoft's NTLM / Kerberos
- Certificate-based security [10]

Token-based security is the most simple form of security in WCF, which lets the Web Service decide how to authenticate usernames and passwords. The Web Service can e.g. compare an incoming set of username and password against the usernames and passwords stored in a central database. The advantage of token-based security is that it is very easy to interop with, because of the simple implementation of the security mechanism. WS clients not fully complying with the WS-Security standard should then easily be able to make the needed implementation to connect with a Web Service using this security mechanism. [11]

Another approach for authenticating users is to use Microsoft's NTLM or Kerberos authentication technologies. Kerberos is one the most secure algorithms today for authentication and has been improved compared to NTLM. Furthermore, the Kerberos authentication is an open standard[8], which NTLM was not. One of the big advantages using

[8] The RCF1510 can be found at http://www.ietf.org/rfc/rfc1510.txt

one of these two authentication technologies is the option for single sign-on[9], which only requires the user to remember one single set of username and password and only logon once. [12] [13]

Certificate-based authentication with the certificate standard X.509v3 is a third way of authenticating users. Certificate-based authentication does not require the authenticator to be part of a Windows Domain, which is the case with the NTLM/Kerberos authentication method. Instead the authenticator must have been issued a certificate by a Certificate Authority (CA).

4.1.4.2 Communication

To secure the communication between the Web Service client and the Web Services there are primarily the following two approaches:

- Transport security (SSL or Encrypted VPN)
- Message security (WS-Security)

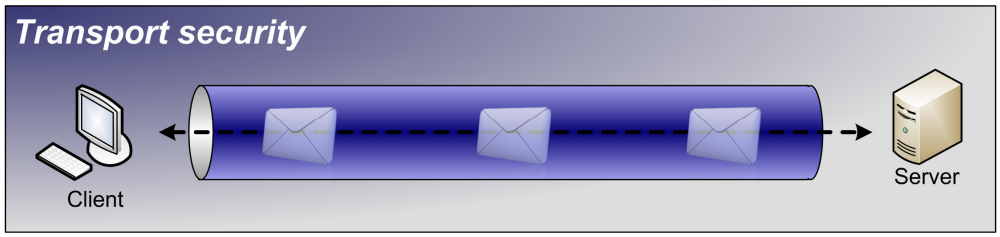


Figure 23: The transport layer is encrypted and thus keeping integrity and confidentiality

Transport security can utilize SSL[10] to make a secure connection between the client and the server by using asymmetric encryption, which provides confidentiality and integrity. Combining this transport security with one of the authentication methods discussed above, a secure Web Service can be made. Another approach to secure the transport layer could be using VPN[11] with encryption enabled, which instead of making an end-to-end security from the client to the server makes a secure connection between the client and the network of the company. Figure 23 illustrates how the transportation is secured between the client and the server. [14]

---

[9] Is defined by http://en.wikipedia.org as "Single sign-on (SSO) is a specialized form of software authentication that enables a user to authenticate once and gain access to the resources of multiple software systems."

[10] SSL is an abbreviation of Secure Sockets Layer and has been approved as a standard by the Internet Engineering Task Force (IETF).

[11] VPN is an abbreviation of Virtual Private Network

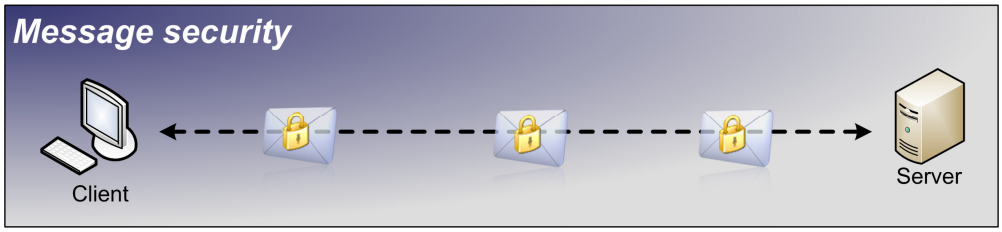


Figure 24: Message security means encrypting the content of every message

Instead of securing the transport layer, another way of securing the communication is to sign and encrypt each message sent to and from the Web Service using the WS-Security standard. The advantage is that integrity and confidentiality are always held independent of the transport used for the communication. So message security may be the best option when the communication needs to be on multiple types of transport. Figure 24 illustrates how each message is secured over an insecure transport layer. [14] [15]

4.2 Local Data Storage

As mentioned earlier in section 3.1 on page 31, the local data storage is a very central part of the Offline Framework. One of the main philosophies behind the Offline Framework is that developers should easily and intuitively be able to access and make changes to the locally cached data.

One solution to this could be to implement methods that give the developer the ability to read from and write to the cached data. However, this would easily become a complex and time-consuming development process, especially if more advanced data querying functionality need to be supported, which is the case with the Offline Framework.

Another and more attractive solution is to make use of the data access framework that already exists in the .NET Framework. This framework is called ADO.NET and it provides the basis data querying functionality needed in a smart client. For this reason it has been decided that the local data storage shall utilize ADO.NET for providing data access to the cached data. An overview of ADO.NET will therefore be given in subsection 4.2.1.

The easy deployment that thin clients possess is an important feature that smart clients also should be able to support as described in section 1.1. To achieve this feature it must be taken into account when implementing the Offline Framework. This will especially have an impact when choosing which embedded database the local data storage will be based on. It is important to keep the size of the installation package and the complexity of the installation to a minimum, and it is therefore required that the embedded database must be as lightweight as possible. This will be analyzed in more detail in subsection 4.2.2.

4.2.1 Overview of ADO.NET 2.0

ADO.NET 2.0 is Microsoft's latest data access technology which is a highly integrated part of the .NET Framework. ADO.NET 2.0 was released with the .NET Framework 2.0 and is an update of the ADO.NET object model, known from the initial version of the .NET Framework.

ADO.NET is an expansion of ActiveX Data Objects (ADO) with some of the key concepts retained. It provides access to structured data from diverse sources and is designed around a primary principle: *simplicity*. The standardized programming model provided ensures that the behaviour exposed by the framework is obvious and consistent, which is one of the main reasons that ADO.NET has become such a successful technology.

The new ADO.NET 2.0 object model is nearly the same as the previous version of ADO.NET, but Microsoft has introduced a few key changes. Most of these enhancements are related to either the .NET Data Provider model or the SQL Server Managed Data Provider.

The .NET Data Provider model ensures that ADO.NET can provide the same basic functionality regardless of the actual data storage system used. This layer of abstraction makes it easy to change from one data storage system to another as data providers must implement the common interfaces defined by the .NET Data Provider model. The data providers can implement these key functions in a way that fully utilizes the characteristics of the underlying data storage system, which makes data access through ADO.NET powerful and effective.

The enhancements to the .NET Data Provider model are primarily that the DataAdapter class has been simplified even more and also introduces a new use of batch updates, which can improve performance. In addition, the two key classes, DataSet and DataTable, now offer a richer set of methods and properties that fill most of the holes in the previous ADO.NET version, which among other things implies that both classes now are serializable to and from XML and can be used as arguments for Web Service calls. The main enhancements to the SQL Server Managed Data Provider are related to various innovations that are provided with the new SQL Server 2005, primarily improving asynchronous commands and making it possible to take advantage of a new query notification mechanism. [18] [21]

The ADO.NET architecture is composed of two central components that provide the data access and data manipulation functionality: the DataSet and the .NET Framework Data Provider. This architecture is visualized in Figure 25.

The DataSet is the core component of the disconnected architecture of ADO.NET and is explicitly designed for data access independent of any data source. The DataSet contains a collection of one or more DataTable objects, which is made up of the actual rows and columns of the data and a collection of any related constraints. The DataSet can contain relational information of its DataTable objects in a collection of DataRelation objects, which

makes it possible for the DataSet to imitate the underlying data source's relational information.

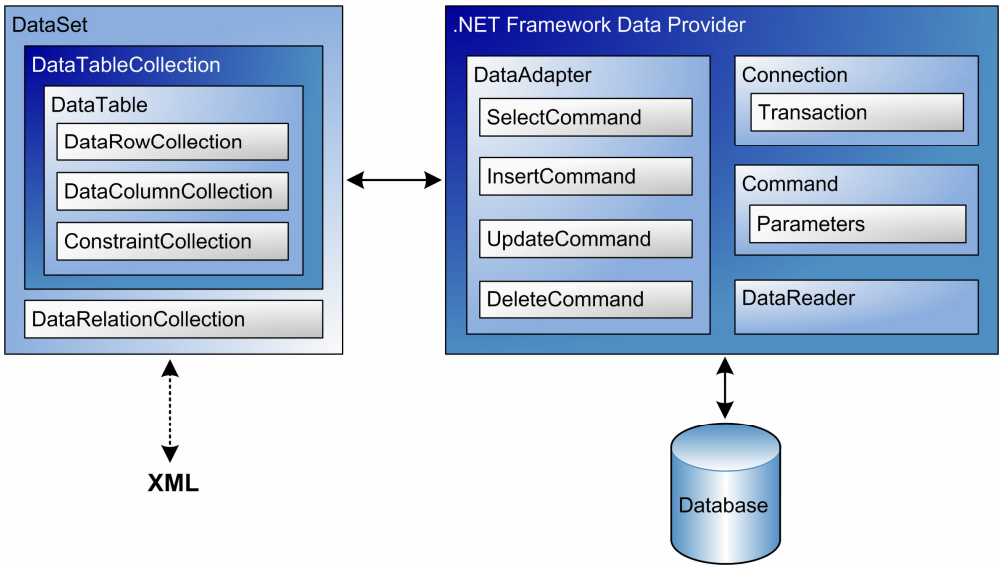


Figure 25: Overview of the ADO.NET architecture [19]

The ADO.NET architecture's other core component is the .NET Framework Data Provider. This component is explicitly designed for data manipulation and fast access to the underlying data storage system. The Connection object provides connectivity to the underlying data storage system and handles transactions, if supported. The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The DataReader object provides read-only access to data through a high-performance stream connected to the underlying data storage system.

The most interesting component of the .NET Framework Data Provider is the DataAdapter object, which acts as a bridge between the DataSet object and the underlying data storage system. The DataAdapter object provides this functionality by utilizing Command objects to execute SQL commands at the data storage system to both load the DataSet with data and reconcile changes made to the data in the DataSet back to the data storage system. [19]

The ADO.NET's DataAdapter object combined with the use of DataSets significantly simplifies the development process of applications needing access to data storage systems. This property combined with the success of ADO.NET makes it an excellent choice to provide easy and intuitive access to the Local Data Storage in the Offline Framework. The Offline Framework will therefore be designed using the ADO.NET Framework.

4.2.2 Embedded Database

This section analyzes which embedded database solution is the most appropriate for local data storage in the Offline Framework. Many different embedded database solutions have been considered, but it has been chosen only to include the most interesting solutions in this analysis.

The embedded database solution must fulfil the following requirements:

- Support for ADO.NET 2.0
- Lightweight installation
- Transaction support
- High performance

The following three Microsoft solutions were first considered to be used as the local data storage:

- Microsoft Jet 4.0
- Microsoft SQL Server Desktop Engine 2000
- Microsoft SQL Server 2005 Express Edition

The Microsoft Jet 4.0 is a well-known solution that is used by Microsoft Access as its underlying database engine. However, the solution was quickly rejected as a possible solution for the embedded database, because Microsoft Jet 4.0 does not support the use of transactions, which is a critical requirement for the local data storage to maintain data consistency.

The Microsoft SQL Server 2005 Express Edition is an updated version the Microsoft SQL Server Desktop Engine 2000, which means that these two solutions are closely related. Both solutions have support for ADO.NET 2.0, transaction support and high performance, which makes them interesting choices for the local data storage. The main disadvantage of these two solutions is that their installation must be performed separately and they both have a heavy footprint when installed. For this reason these two solutions was discarded as a solution for the local data storage. [26]

In the search for alternative embedded database solutions, many different solutions, such as MySQL, ProgreSQL, Firebird, SQlite and SharpHSQL, were considered. After having examined and tested these different solutions more closely, the most interesting and appealing solution appeared to be the Firebird solution. For this reason a further description of the Firebird solution will be given in the following paragraphs.

Firebird[12] is an open source relational database system offering a fully featured and powerful database solution with excellent concurrency support and high performance. Firebird is derived from Borland InterBase 6.0 source code and has been used in production systems under a variety of names since 1981. The fact that the Firebird technology has been used for more than 20 years makes it a very mature and stable product. Firebird is licensed under the InterBase Public License v.1.0, which means that it is completely free of any registration, licensing or deployment fees, and may be deployed freely for use with any third-party software, whether commercial or not.

The embedded version of the Firebird database has all the features supported in the full server version including full ACID[13] compliant transactions on a very small footprint. The solution is very lightweight as only 8 files are required with a total footprint of approximately 6 MB. No installation is necessary as the files only need to be copied to the applications directory for it to be able to utilize the functionalities provided by the Firebird database solution.

A separate open source project provides an ADO.NET 2.0 Data Provider for the Firebird solution, which is called the Firebird ADO.NET Data Provider 2.0. This Data Provider makes it easy to utilize the Firebird database solution with the ADO.NET 2.0 framework. Both projects have communities that offer good support.

Many improvements have been made in the second release candidate of the Firebird 2.0 database compared to the latest stable Firebird 1.5.3 version, which makes the newest version the most appealing solution. The disadvantage is that the Firebird 2.0 database does not exist in a stable version, which can be problematic in production environments. Another disadvantage is that the Firebird solution does not support encryption of its database files. [27]

To summarize the advantages and disadvantages of the Firebird solution are listed below:

Advantages:

- Full ACID compliant transactions
- ADO.NET 2.0 support
- Very small footprint
- High performance
- No installation needed
- Embedded version has all the features from the server version
- Community offering good support

---

[12] The Firebird database and the related Firebird ADO.NET 2.0 Data Provider can be found at http://www.firebirdsql.org.

[13] ACID stands for Atomicity, Consistency, Isolation, and Durability.

Disadvantages:

- Firebird 2.0 is only available as a release candidate
- Encryption of database file not supported

The Firebird solution is the most appealing embedded database of the different solutions considered in the analysis. The fact that the Firebird 2.0 currently is available only as a release candidate is not seen as a critical disadvantage, because it is estimated to be released in a stable version in short time. The disadvantage that the Firebird solution does not support encryption of its database files is also not seen as a critical disadvantage as the security of the overall solution can be solved by other means, e.g. restricting access to the database file or utilizing other encryption technologies.

The Firebird solution is chosen as the embedded database for the local data storage as it fulfils all the requirements. The implementation described in chapter 5 will therefore utilize the Firebird solution.

4.3 Summary

In this chapter, relevant technologies within communication and local data storage have been analysed.

Windows Communication Foundation has been examined and is considered to fulfil a key requirement described in the problem domain, which is the requirement for secure communication and authentication. This requirement is fulfilled because WCF supports several ways of securing communication and authentication. Furthermore, WCF has support for reliable communication, which can have vital consequences if not supported as discussed in section 4.1.2.

Firebird has been chosen as the local data storage because this database solution fulfils the requirement of a high performance local data store providing fast access to the local data. This embedded database also makes data queries and manipulation easy for the developer.

The chosen technologies will be utilized in the design and implementation of the Offline Framework, which is discussed in chapter 5.

5 The Offline Framework

This chapter describes the design and implementation of the Offline Framework, which is designed and developed on the basis of the conducted research. The Offline Framework is implemented using Microsoft's .NET 2.0 Framework and written in the programming language C#. The implemented Offline Framework is a prototype that should be seen as an architectural proof of concept, and there is therefore still room for improvements.

The purpose of this chapter is to:

- provide an architectural overview of the Offline Framework developed
- describe the most relevant components of the Offline Framework

After having read this chapter, the reader should have an understanding of the architecture and components of the Offline Framework.

5.1 Architectural Overview

This section provides an overall design architecture of the Offline Framework, and a description of the interactions between the different components inside and outside of the Offline Framework will be given.

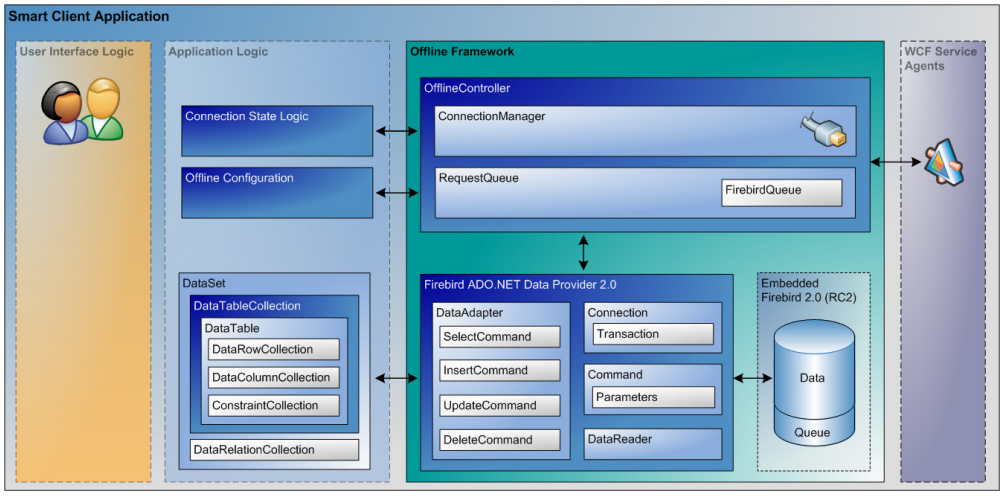


Figure 26: Architectural overview of the Offline Framework

Figure 26 illustrates the components provided by the Offline Framework and what the developer must implement in order to utilize the Offline Framework. The Offline Framework is made with inspiration from the Built-in Cache Framework described in section 2.3.2 and the Data-Centric Architecture described in section 2.2.1. The implementation is

based on the solution strategy described in section 2.4. The Offline Framework handles many of the complex tasks, which the developer formerly was forced to implement. The implementation of offline capable smart clients is simplified, because the Offline Framework ensures that data is up-to-date and that changes automatically are propagated to a central Web Service.

The Offline Framework consists of the following central components:

- **OfflineController**
  The OfflineController is the main object of the Offline Framework and is responsible for coordinating the interactions between the different components of the Offline Framework. The Cache Manager, which keeps the local data storage up-to-date, is contained in the OfflineController object. The OfflineController object is described in more details in section 5.2.

- **ConnectionManager**
  The ConnectionManager object monitors the connection state of the application. The object is utilized by the Offline Framework and in the Application Logic. The ConnectionManager object is described in more details in section 5.4.

- **RequestQueue**
  The Queue Manager is implemented in the RequestQueue object, which is responsible for synchronizing the changes made to the local data storage with the relevant Web Service methods. The RequestQueue object utilizes a FirebirdQueue object, which provides the necessary functionality to persist the queue in the embedded Firebird database.

- **Firebird ADO.NET Data Provider 2.0**
  This third-party component provided by the Firebird project enables the Offline Framework to utilize the ADO.NET framework to take advantage of the features in the Data-Centric Architecture.

- **Embedded Firebird 2.0 (RC2)**
  The Local Data Storage is achieved by utilizing the embedded Firebird 2.0 (RC2) relational database. This embedded database is responsible for persisting the local data and queue.

The described components are the components provided by the Offline Framework, which can be utilized by developers to implement offline capable smart clients more easily. The developer must provide a Service Agent for the Web Service that is to be used by the

Offline Framework and make the necessary implementation in the Application Logic. The Web Service must be based on the CRUD design described in section 3.2, which is relatively easy to implement and a requirement that many existing Web Services already fulfil.

The developer must provide the following implementation in the Application Logic in order to utilize the Offline Framework:

- **Connection State Logic**
  Using the connection state provided by the Offline Framework the developer can inform the user of the current state of the smart client. The smart client can also change the functionality available according to its connection state. Finally, the user should be able to force the smart client to go offline or online, if possible.

- **Offline Configuration**
  The developer must provide a configuration of the Offline Framework in order to function correctly. This configuration is used by the OfflineController and RequestQueue to keep the local data storage up-to-date and synchronize the local changes made with the relevant Web Service methods. The Offline Configuration corresponds to the Data Mapping Logic illustrated in Figure 11 in section 2.4.

- **Data access and data manipulation**
  The main interaction with the Offline Framework is made using the data access and data manipulation functionality provided with the ADO.NET framework's DataSet's object combined with the Firebird ADO.NET Data Provider in the Offline Framework. The developer primarily has to concentrate on making changes to the local data and not on performing Web Service calls, as this is handled automatically by the Offline Framework, which simplifies the implementation of the smart client application.

An overall description of the architecture of the developed Offline Framework has been outlined with an overview of which components are provided by the Offline Framework and which parts must be implemented by the developer.

The diagram in Figure 27 on page 60 illustrates the interaction between the components of the Offline Framework and provides an understanding of how data is propagated and replicated by the framework.

The diagram is divided into three parts, which are the Connection Manager, Queue Manager and Cache Manager. The Connection Manager activates the Queue Manager, when the application changes connection state to online. Then the Queue Manager examines

whether any requests have been queued and if so the first request in the queue is executed. If the execution went well, the request is removed from the queue and the process is repeated until the queue is empty. If the execution of a request fails, the developer has the opportunity to take action and decide whether the request should be removed without being executed. If this is chosen, the related local data is marked as dirty and the request is removed from the queue. Marking the data as dirty indicates that the data is inconsistent and needs to be refreshed as soon as possible. If it is chosen not to remove the failed request, the application is forced to go offline.

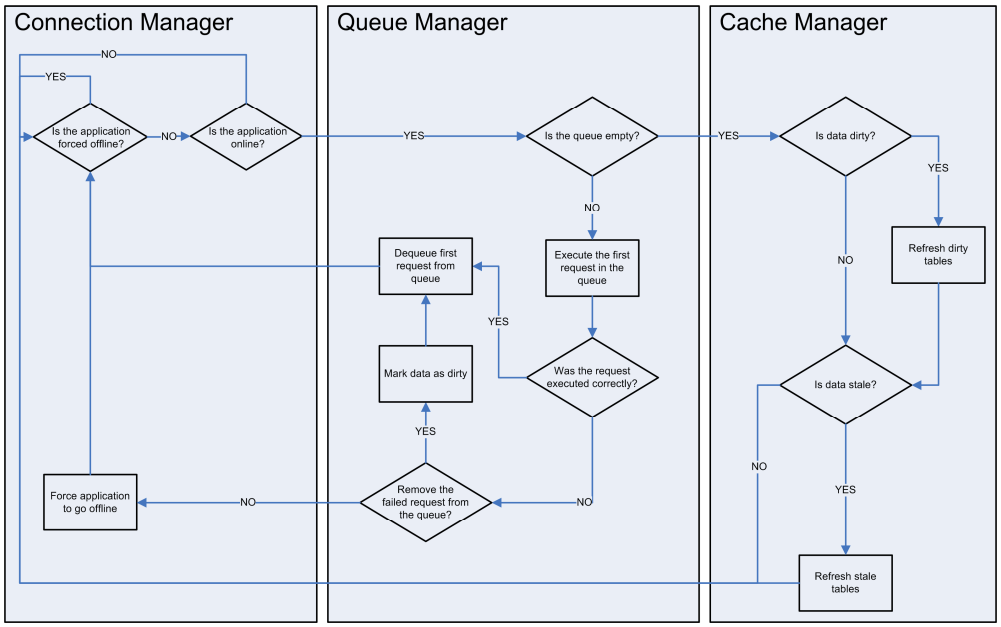
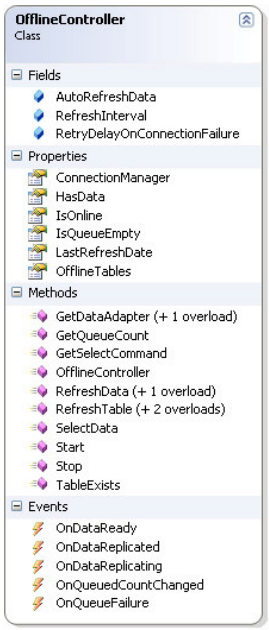


Figure 27: Interaction between the Offline Framework's components

When the queue is empty the Cache Manager is activated. The purpose of the Cache manager is to refresh local data when needed. It checks whether any dirty data exists and if so it refreshes the dirty data. Afterwards it examines whether the data is stale, which is the case if a refresh interval has been exceeded. After refreshing the local data, the process ends up in the Connection Manager in which the flow chart is repeated.

This overview provided a description of how data is replicated and synchronized in the Offline Framework. To get a better understanding of the object model used in the Offline Framework the most interesting classes are described in more details in the following sections.

5.2 OfflineController



The OfflineController is the most central component of the Offline Framework as it coordinates the interactions between all the other components of the architecture. The Offline Controller has a Connection Manager, which makes the developer capable of managing the connection state of the application. When the OfflineController is started two separate threads are stated. One thread is used for constantly checking the connection state and another thread is used for running the queue execution and refreshing tasks. The OfflineController appends a GUID[14] to each row in the tables in the local data storage to simplify the implementation in the RequestQueue.

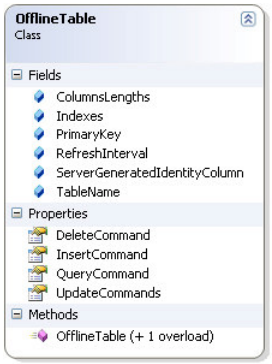
The property OfflineTables is used to configure the Offline Controller and is therefore an essential part of the Offline Framework. The configuration of the Offline Framework is described in more detail later in this section.

Fields:	
AutoRefreshData	Indicates whether the OfflineController should automatically refresh stale data or it should be done manually.
RefreshInterval	Gets or sets the default refresh interval.
RetryDelayOnConnectionFailure	On Web Service connection failure, a retry will be performed with the specified interval.
Properties:	
ConnectionManager	The ConnectionManager manages the offline/ online state of the application. The connection manager is discussed further in section 5.4.
HasData	Indicates whether data ever has been replicated. If not, there is no data in the local data storage.
IsOnline	Indicates whether the application state is online or offline.
IsQueueEmpty	Indicates whether the queue is empty.
LastRefreshDate	Indicates the date and time of the latest refresh performed in the local data storage.
OfflineTables	Contains an array of OfflineTable objects to configure replication process for each table.

[14] GUID - Globally Unique Identifier

Methods:	
GetDataAdapter	Used to retrieve a new data adapter to perform queries and make changes to the local data storage.
GetQueueCount	Gets the number of queued Web Service calls.
GetSelectCommand	Gets an empty SelectCommand.
OfflineController	Is the constructor.
RefreshData	Forces data to be refreshed
RefreshTable	Forces a single table to be refreshed
SelectData	Returns a DataTable object based on a select statement.
Start	Starts the OfflineController.
Stop	Stops the OfflineController.
TableExists	Indicates whether one or more specific tables exist in the local data storage.
Events:	
OnDataReady	Fires when data is ready to be manipulated. This happens once.
OnDataReplicated	Fires when a replication has been performed.
OnDataReplicating	Fires when a replication is being performed.
OnQueueCountChanged	Fires when the number of items in the request queue changes.
OnQueueFailure	Fires on queue execution failures and makes it possible to take action on these failures.

Table 1: OfflineController class described



The OfflineTable object holds the configuration information needed for the Offline Framework to synchronize and replicate data from a Web Service in a table in the local data storage. When an OfflineTable is created, various settings need to be configured, such as ColumnLengths, Indexes, PrimaryKeys, IdentityColumns and TableName, which are described further in the following table.

Fields	
ColumnLengths	Indicates the lengths that should be used for the specified columns in the local data storage.
Indexes	Indicates the indexes that should be created in the local data storage to optimize the performance of the queries preformed.
PrimaryKey	Indicates which columns which make up a primary key.
RefreshInterval	Indicates the interval for data to be refreshed.
ServerGeneratedIdentityColumn	Indicates an identity column if one exists. Server generated identity column must be handled specially by the Offline Framework.
TableName	Indicates the name of the table in the local data storage.
Properties	
DeleteCommand	Configures how a row should be deleted.
InsertCommand	Configures how a row should be inserted.
QueryCommand	Configures how data can be refreshed.
UpdateCommands	Configures how data can be updated.
Methods	
OfflineTable	Is the constructor.

Table 2: OfflineTable class described

The OfflineTable class has four key properties QueryCommand, InsertCommand, UpdateCommd and DeleteCommand, which inherits from the class BaseCommand as illustrated in Figure 28. These four properties provide the core configuration of how to replicate data and synchronize changes made with a Web Service.

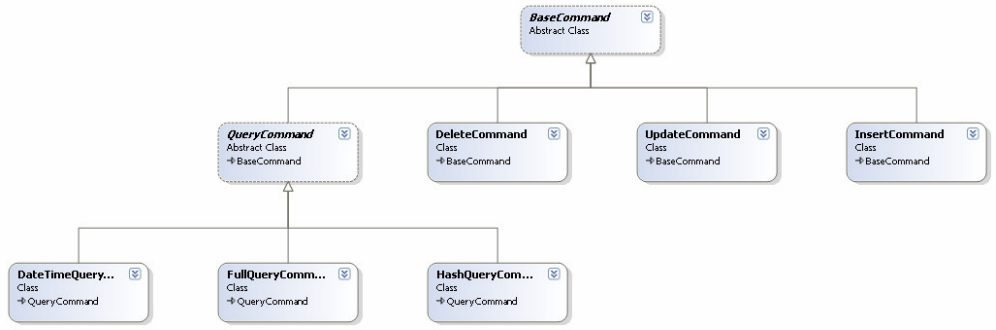


Figure 28: Illustrates classes which inherit from BaseCommand

The following is a description of the individual classes and their function:

- **QueryCommand**
  The QueryCommand is used to configure how data is replicated from the Web Service to the local database. Basically, a Web Service method is defined, which can be used to retrieve all required data for a single offline table. The DateTimeQueryCommand, FullQueryCommand and HashQueryCommand are inherited from QueryCommand. These classes make it possible to perform the replication using different techniques, which are described below:

  - **FullQueryCommand**
    The FullQueryCommand always replicates all data from the Web Service and has by no means been optimized.

  - **HashQueryCommand**
    The HashQueryCommand is optimized by utilizing hash values to track, if any changes have occurred in the data from the Web Service as described in section 3.2.1.

  - **DateTimeQueryCommand**
    The DateTimeQueryCommand is optimized by only replicating changes from the Web Service by utilizing timestamps in the server database as described in section 3.2.1.

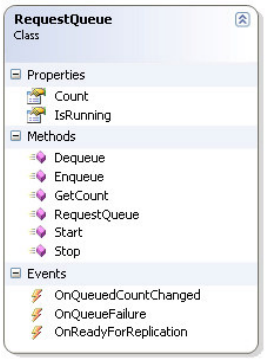
- **InsertCommand**
  The InsertCommand configures what Web Service method to call, when a new row is created in the related offline table.

- **UpdateCommand**
  The UpdateCommand configures what Web Service method to call, when a row is updated in the related offline table. Multiple UpdateCommand objects can be configured, so different Web Service methods can be used for each column in a table. In this way it is possible to perform updates at column level and not only at row level, which makes the Offline Framework more flexible.

- **DeleteCommand**
  The DeleteCommand configures what Web Service method to call when a row is deleted from the related offline table.

This section provided an overview of the OfflineController class in the Offline Framework and how the OfflineController is configured using the OfflineTable object.

5.3 RequestQueue



The task of the RequestQueue is to queue outgoing Web Service calls for execution at a later stage when the application is able to connect to the Web Service. The RequestQueue is a component of the Offline Framework, which is primarily accessed by the OfflineController that automatically queue method calls, when changes are made to the local data storage.

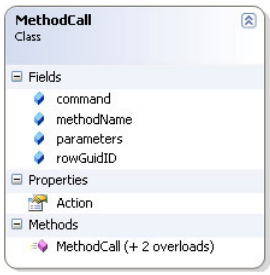
The RequestQueue is implemented as a FIFO[15] queue, which means that the first Web Service call queued is the first one to be served when the queue is run. This ensures that changes are propagated in the right order. A special table is created in the Firebird database, which is utilized by the RequestQueue to store the queued Web Service calls.

Properties:	
Count	Indicates the number of currently queued Web Service calls.
IsRunning	Indicates whether the thread is running.
Methods:	
Dequeue	Removes a Web Service call from the queue.

[15] Acronym for First-In, First-Out - see http://en.wikipedia.org/wiki/FIFO for more info.

Enqueue	Adds a Web Service call to the queue.
RequestQueue	Is the constructor.
Start	Starts the thread.
Stop	Stops the thread.
Events:	
OnQueuedCountChanged	Fires when the number of queued Web Service calls changes.
OnQueueFailure	Fires when a Web Service fails to be executed.
OnReadyForReplication	Fires when the queue is empty and replication can be performed.

Table 3: RequestQueue class described

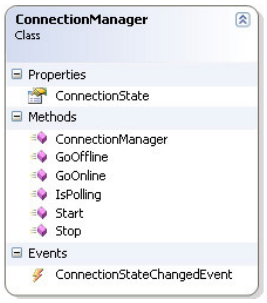


Web Service calls are encapsulated in an object called MethodCall, which encapsulates all relevant information such as the Web Service method name to call, parameters and the related GUID of the row which is changed. The encapsulation is used to gather all relevant information in one object, which can be stored in the queue in the embedded Firebird database for later use.

Fields:	
command	Is a reference to the command, which has been triggered.
methodName	Is the method name on the Web Service which should be called.
parameters	Is an array of parameters which should be used to call the Web Service method.
rowGuidID	Is the GUID that relates the method call to a unique row in the local data storage.
Properties:	
Action	Indicates the type of action which is performed. The property can either be Delete, Insert or Update.
Methods:	
MethodCall	Is the constructor.

This section provided an overview of the RequestQueue class in the Offline Framework and described how MethodCall objects are used to persist Web Service requests in the queue.

5.4 ConnectionManager



The ConnectionManager is inspired by the Offline Application Block described in section 2.3.2 and is used for keeping track of the current connection state of the smart client application. The ConnectionManager has its own polling thread, which is used for constantly checking the connection state of the application. When the state changes the event ConnectionStateChangeEvent is fired to inform other components of the connection state change. The ConnectionManager class is described in Table 4.

Properties:	
ConnectionState	Indicates which state the ConnectionManager currently is in. The state can be Online, Offline or Unknown, in relation to which Unknown is used if the connection state cannot be determined.
Methods:	
ConnectionManager	Is the constructor.
GoOffline	Forces the ConnectionManager to change its state to Offline.
GoOnline	Forces the ConnectionManager to try to change its state to Online.
IsPolling	Indicates whether the thread is running.
Start	Starts the polling thread.
Stop	Stops the polling thread.
Events:	
ConnectionStateChangedEvent	Fires when the connection state changes.

Table 4: ConnectionManger class described

This section provided an overview of the ConnectionManager class, which is inspired by the implementation made in the Offline Application Block described in section 2.3.2.

5.5 Summary

In this chapter an architectural overview of how the Offline Framework is implemented has been provided. The implemented architecture complies with the solution strategy chosen in section 2.4. The central process of propagating changes and replicating data has been examined in a diagram to illustrate the interaction between components of the Offline Framework.

The most relevant classes and components in the Offline Framework have been outlined to give the reader an understanding of the structure of the framework. All together, this has provided the reader with an overview of how the Offline Framework has been implemented.

6 Case Study

This chapter describes how the developed Offline Framework has been utilized to solve a case study, which has been carried out in cooperation with PwC. The purpose of this case study is to develop an offline capable smart client solution, which will improve and ease the time registration in PwC's current ERP system. A brief introduction to the case scenario can be found in section 1.2.2.

The purpose of this chapter is to:

- provide an analysis of the Maconomy ERP system
- describe the overall design used in the case study
- provide an overview of the implementation made
- describe the tests performed on the implemented solution

After having read this chapter, the reader will have obtained an understanding of how the Offline Framework has been utilized in the case study to develop a proof of concept for an offline capable smart client solution, which solves a practical problem at PwC.

6.1 Analysis

The time sheet functionality of the Maconomy ERP system is central to this project, because the smart client has to interact with this part of the system. For this reason a short description of the Maconomy ERP system will be made.

The company Maconomy is a global provider of business management solutions for project and knowledge intensive organizations. Their main product is an ERP system called "Maconomy", which is primarily designed for agencies, consulting companies, audit and tax consulting companies, research institutes and PR agencies.

The Maconomy system is based on an application server, which stores data in a database. The system can be accessed either directly using a rich client application or via a web portal using an internet browser as a thin client. The communication between the application server and the rich client and the web portal is based on a proprietary communication protocol developed by Maconomy.

The rich client makes it possibly to access advanced parts of the Maconomy system, but is complex to use. Maconomy mainly focuses on developing a more user-friendly interface in their web portal than in the rich client. The idea behind this approach is that normal users access the system via the web portal, while back office and finance employees use the rich client to carry out more advanced and demanding tasks. For this reason more and more

business logic is located in the web portal. The architecture of the Maconomy system is illustrated in Figure 29.

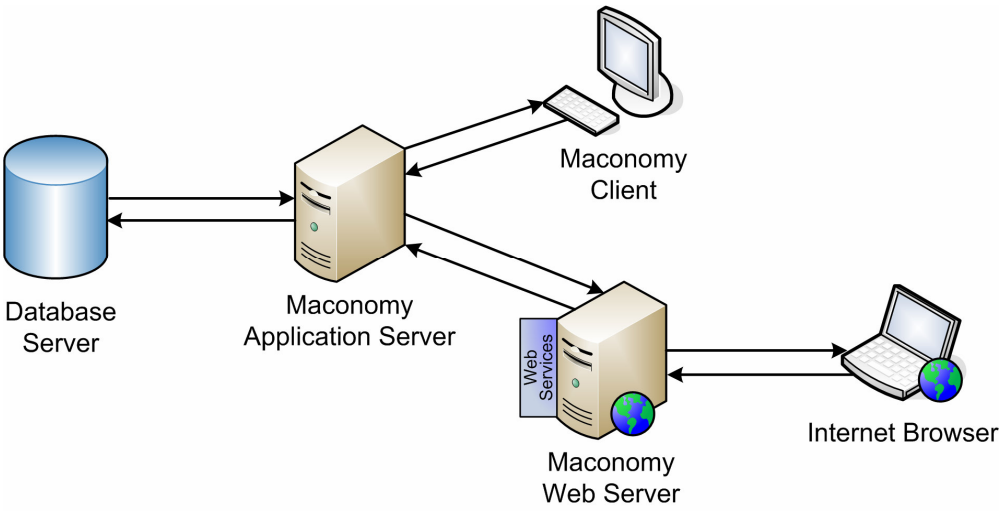


Figure 29: Architecture of the Maconomy ERP system

Maconomy focuses on making their solution highly customizable. For this reason it is possible to change the layout of every dialog in the rich client, but not possible to change the business logic of dialogs or add new dialogs. The web portal is more customizable, due to the fact that it allows not only layout changes of the existing Maconomy web components, but also development of customized web components, which can interact with other systems. This illustrates that it is in the web portal as to which the main development is taking place at the moment.

Maconomy has developed their own proprietary programming language called M-Script and several supporting frameworks to ease the development of web components in the web portal. The M-Script programming language contains a special API for the Maconomy system, which makes it possible to carry out any action that can be made with the rich client. Maconomy uses this functionality in the web portal to focus on workflows and customers can use it to create customized web components.

A framework for Web Services is provided by Maconomy, which makes it possible for other systems to interact with the Maconomy system. This functionality is closely related to the web portal and these Web Services are implemented using the M-Script programming language.

The Web Service framework for the Maconomy system is very easy to use, but does not support the new Web Service standards for Transactions, Reliability and Security as discussed in section 4.1. This is a challenge that must be taken in to consideration in relation to the overall system design.

6.1.1 Specifications

The primary objective of the case study is to develop an offline capable time registration client, which should make it possible for PwC's employees to carry out time registration without an internet connection or physically being at the office. Another objective of the case study is to prove that the Offline Framework developed is feasible in solving a practical problem.

PwC has the following central requirements for the smart client:

- **Functionality as existing web-based application**
  The functionality of the smart client should have the same look and feel as the current web-based application, which only provides online support. This functionality will be described in more details in section 6.1.2.

- **Changes must be performed through the application server**
  All changes to the Maconomy ERP system are required to be made through the application server.

- **Secure communication**
  Secure communication is needed because business critical data is going to be transferred between the smart client and the Web Service.

- **User-friendly and responsive user interface**
  The user interface is required to be responsive and user-friendly which the current web-based client applications have not been able to do efficiently. The offline client should provide fast searching capabilities to easily and efficient find jobs and tasks.

The following analysis, design and implementation in this case study are based on the specifications described.

6.1.2 Time Registration Functionality

This section describes the time registration process and the special behaviours of time registration in the Maconomy ERP system more closely. The web-based Maconomy application is analyzed and utilized to describe the functionality, which should be adapted to the new offline smart client solution.

As shown in Figure 30 on page 72, the web-based Maconomy application provides a well-arranged grid, which provides an overview of the time sheet lines for a given week. The grid provides the option for creating, updating and deleting the individual time sheet lines. Each time sheet line consists of a given job number with a related task number together with time

used on the given job and task for each day in the week. It is also possible to specify a remark to indicate more precisely what have been executed on the given task.

The creation of time sheet lines is troublesome and far from user-friendly, because the user needs to know the job number and the task number in order to quickly create a time sheet line. The user, therefore, typically needs to open a search window, which is done by pressing CTRL-G, in order to find the job number of a specific job. Performing these searches in the Maconomy web portal is a slow process, which have a negative affect on the user experience as the user quickly gets frustrated over the poor response times. The same slow search process must be performed when finding the relevant task number, because remembering the different task numbers is not feasible as the jobs can have different task lists to choose from. These searches should be optimized in the smart client to provide a more user-friendly and responsive user interface.

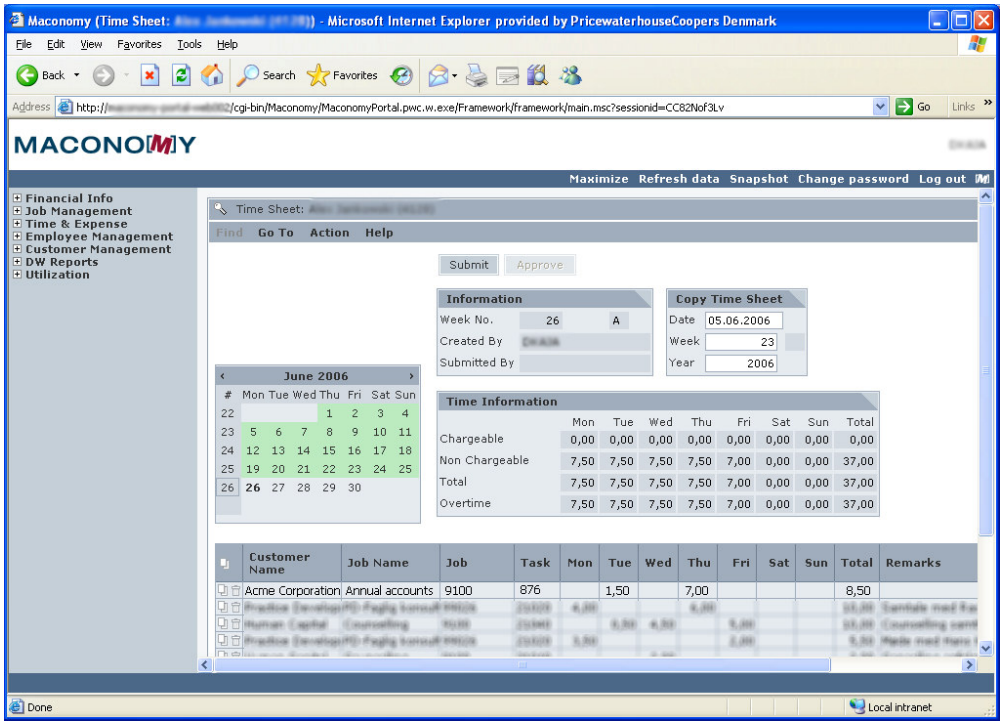


Figure 30: The web based Maconomy client application

In the Maconomy ERP system, time sheets are to be submitted weekly by each employee, which will lock these submitted time sheets for further changes. This functionality is necessary, because the time registrations is posted in the system, which is used to send invoices to clients and to provide reports that are used to measure the financial development of the company. For that reason time sheets need to be locked at some point to be sure no changes occur back in time. A month calendar illustrates what weeks are handed in and what weeks are not. This is done by colouring the individual weeks in the calendar red or green to

indicate the state of the time sheets; where green indicates that the time sheet has been submitted and red indicates that the time sheet is due to be submitted.

A table showing an overview of the time registered in the selected time sheet is placed on right-side of the calendar. The table shows chargeable time, non-chargeable time and overtime distributed between the individual weekdays, and totals are also calculated.

An overview of the relevant functionality that the offline client should adapt from the web-based Maconomy application has now been given. To get a better understanding of the underlying data for the time sheet component the following section focuses on the data structure utilized by the component.

6.1.3 Data Structure

Maconomy is a comprehensive ERP system that has over 650 different tables in its underlying database. Describing this data structure would be a very large and time-consuming task, which is why only the tables relevant for the case study are described below. Figure 31 gives an overview of these tables showing only the most relevant attributes.

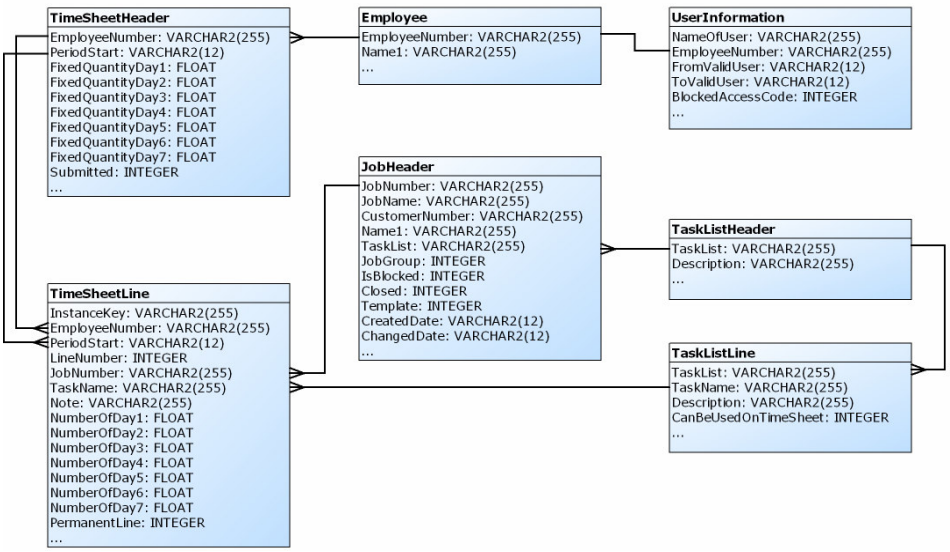


Figure 31: Overview of the relevant Maconomy tables

The UserInformation table contains all users registered in the Maconomy ERP system and in this case study it is primarily used to map a Windows domain user to an employee in the system. PwC has chosen that an employee's Maconomy username (NameOfUser) and Windows domain username must be the same, which makes mapping a Windows domain user to a Maconomy user a simple task.

Attribute:	Description:
NameOfUser	The Maconomy username.
EmployeeNumber	The employee number of the user.
FromValidUser	The date that the user is valid from.
ToValidUser	The date that the user is valid to.
BlockedAccessCode	Indicates whether the user has access to the system.

Table 5: UserInformation schema

The Employee table contains PwC's former and present employees. The main purpose of using this table is to retrieve an employee's name.

Attribute:	Description:
EmployeeNumber	The employee's unique number.
Name1	The name of the employee.

Table 6: Employee schema

An employee registers his or hers hours spent in time sheets in the Maconomy ERP system. Maconomy divides these time sheets into weeks. There exist two different kinds of weeks: normal weeks and split weeks. The reason for having split weeks is to be able to separate one month's registered hours from another's. Split weeks occur when the last day of a month is not a Sunday, and instead of one normal week a split week is created for both the current and next month. An employee finishes a time sheet by submitting it, which locks the time sheet for further registration.

Attribute:	Description:
EmployeeNumber	The employee number.
PeriodStart	The date for the first day in the time sheet. This date is either a Monday and/or the first day of the month.
FixedQuantityDay1..7	The employee's fixed working hours from Monday (1) to Sunday (7). This can be used to calculate the employee's overtime.
Submitted	Indicates whether the time sheet has been submitted or not.

Table 7: TimeSheetHeader schema

The individual time sheets can have one or more lines that contain the actual hours registered. To register time a job number and a related task number must be specified. The task number must be selected from the list of tasks, which is specified by the selected job's TaskList attribute.

Attribute:	Description:
EmployeeNumber	The employee number.
PeriodStart	The date for the first day in the time sheet. This date is either a Monday and/or the first day of the month.
LineNumber	The number of the line.
JobNumber	The job that the time is registered on.
TaskName	The related task for the time registered.
Note	Optional remark for the line.
NumberOfDay1..7	The hours registered from Monday (1) to Sunday (7).

PermanentLine	Indicates whether the time sheet line is a permanent line that easily can be copied to future time sheets.

Table 8: TimeSheetLine schema

The JobHeader table contains all jobs in the Maconomy ERP system and has the following relevant attributes:

Attribute:	Description:
JobNumber	The job's unique number.
JobName	The name of the job.
CustomerNumber	The job's related customer number.
Name1	The name of the related customer.
TaskList	The tasklist that shall be used when registering hours on this job on time sheets.
JobGroup	Indicates whether the job is an internal or external job.
IsBlocked	Indicates whether the job is blocked for registration on time sheets.
Closed	Indicates whether the job is closed. If the job is closed time registration is not possible.
Template	Indicates whether the job is a template job. Template jobs are not relevant in connection with time registration.
CreatedDate	The date for when the job was created.
ChangedDate	The date for when the job was last changed.

Table 9: JobHeader schema

The different task lists are contained in the TaskListHeader table, which has the following interesting attributes:

Attribute:	Description:
TaskList	The unique identifier for the task list.
Description	The description for the task list.

Table 10: TaskListHeader schema

The task lists can have one or more lines, which are contained in the TaskListLine table.

Attribute:	Description:
TaskList	The related task list.
TaskName	The name of the task.
Description	The description of the task.
CanBeUsedOnTimeSheet	Indicates whether the task can be used to register time on a job.

Table 11: TaskListLine schema

The overview of the data structure completes the analysis of the Maconomy ERP system and provides the reader with necessary knowledge to understand how the system utilizes the information in the database.

6.2 Design

This section describes the overall design used in the case study to make an offline capable client interact with the Maconomy ERP system. An overview of the overall design used in the case study is illustrated in Figure 32, which shows the existing system on the left side of the dotted line and the solution for solving the offline capability issue on the right side of the dotted line. Data exchange with the Maconomy system is done through a Web Service on the Maconomy Web Server, which is the only way to make changes to the system, because corporate policies state that changes to the database may only take place through the Maconomy Application Server as stated in the specification in section 6.1.1.

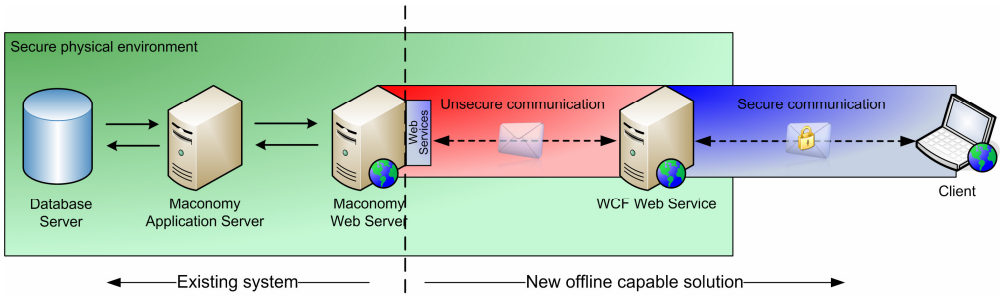


Figure 32: The overall system design

The Maconomy Web Service framework does not support the new standards such as Reliability, Transactions and Security, which is why an additional WCF Web Service is used to provide a reliable and secure communication to the client. By using the additional WCF Web Service the following benefits can be utilized:

- Reliable messaging
- Windows Kerberos single sign-on
- Encrypted & signed message exchange

The messages exchanged between the WCF Web Service and the smart client may be sent over an unreliable internet connection, where loss of messages is likely to occur. This is why reliability is an important technology to utilize, as it prevents the problems described in section 4.1.2 from arising.

The Kerberos single sign-on feature is also a very secure and user-friendly authentication technology, which can be utilized straight out of the box in Windows Communication Foundation. In collaboration with the security standard, which WCF supports, messages can be signed and encrypted. The combination of these features in the Windows Communication Foundation provides a very secure and reliable communication mechanism.

In the figure, the secure physical environment is indicated with a green colour. Within this area insecure messages can be exchanged between application and services without being

tampered with or viewed by unauthorized persons. The security requirements within this area can therefore be lowered and communication can be performed without encrypting and signing data. Outside the secure physical environment the communication between the offline smart client and the WCF Web Service secured by utilizing the features in WCF described. The two Web Services ensure that changes are performed through the application server and that the communication to the client is secured.

Efficient search functionality will be provided in the smart client to fulfil the requirement of a user-friendly and responsive user interface. This functionality will be achieved utilizing the high-performing local database combined with the development of an auto-complete feature. Auto-completion is an efficient way to make searches as you type. If an employee searches for a customer, he/she just types in a part of their name and a control pops up with matching customers. This functionality makes it easy to find customers and makes the application more user-friendly.

The overall design described is utilized in the following section, in which the implementation of the Maconomy Web Service, WCF Web Service and the Offline Smart Client will be described.

6.3 Implementation

This section provides an overview of the implementation made in the case study. The main focus is on how the Offline Framework is utilized in the implementation of the smart client for the Maconomy ERP system.

The section is divided in to three subsections, where the first two subsections. The first two subsections briefly describe the Maconomy Web Service and the WCF Web Service. These two subsections give the reader an understanding of how the Web Services are implemented. The third subsection gives an overview of how the offline smart client have been implemented and how the described Web Services are utilized in order to achieve the desired functionality.

6.3.1 Maconomy Web Service

The purpose of the Maconomy Web Service is to provide an interface to the Maconomy ERP system, which can be utilized by the smart client solution. The Web Service is implemented using the M-Script language. This subsection does not go into details of how the Web Service has been implemented using the M-Script language, because this is not seen to be relevant for the understanding of the overall system and would merely confuse the reader. The source code for the Maconomy Web Service can be examined more closely on the enclosed CD-ROM.

Table 12 provides an overview and a short description of the individual methods exposed by the Maconomy Web Service, which in the Maconomy ERP system is known as TimeSheetService.

- **CreateTimeSheetLine** (*username* As string, *password* As string, *nameOfUser* As string, *periodStart* As date, *jobNumber* As string, *taskName* As string, *note* As string, *numberOfDay1* As double, *numberOfDay2* As double, *numberOfDay3* As double, *numberOfDay4* As double, *numberOfDay5* As double, *numberOfDay6* As double, *numberOfDay7* As double, *permanentLine* As long) As string
  Creates a time sheet line and return the corresponding InstanceKey.
- **DeleteTimeSheetLine** (*username* As string, *password* As string, *nameOfUser* As string, *instanceKey* As string) As boolean
  Deletes a time sheet line and returns true if successful.
- **SetHours** (*username* As string, *password* As string, *nameOfUser* As string, *instanceKey* As string, *dayInWeek* As long, *hours* As double) As boolean
  Sets hours on a time sheet line for the day defined by dayInWeek and returns true if successful.
- **SetJobNumberAndTaskName** (*username* As string, *password* As string, *nameOfUser* As string, *instanceKey* As string, *jobNumber* As string, *taskName* As string) As boolean
  Sets Job Number and Task Name on a time sheet line and returns true if successful.
- **SetNote** (*username* As string, *password* As string, *nameOfUser* As string, *instanceKey* As string, *note* As string) As boolean
  Sets Note on a time sheet line and returns true if successful.
- **SetPermanentLine** (*username* As string, *password* As string, *nameOfUser* As string, *instanceKey* As string, *permanentLine* As long) As boolean
  Sets PermanentLine on a time sheet line and returns true if successful.
- **SubmitTimeSheet** (*username* As string, *password* As string, *nameOfUser* As string, *periodStart* As date) As boolean
  Submits a time sheet and returns true if successful.

Table 12: Methods for TimeSheetService

Authentication against the Maconomy Web Service is provided in a very simple and insecure way, as username and password are sent in clear text, which only is acceptable because the message exchange is within the secure physical environment. The username and password provided to these methods must be an administrative user with the rights to perform the actions needed for all other users. The different methods must also be provided with the Maconomy username (nameOfUser) in order for the Web Service to know for which employee the actions must be performed.

The Maconomy Web Service has no methods for reading data, as this is performed directly from the WCF Web Service to minimize the load on the Maconomy's web server and application server.

6.3.2 WCF Web Service

The purpose of the WCF Web Service is to act as a layer between the Maconomy ERP system and the offline smart client ensuring that communication to the client is secure and reliable. The authentication mechanism available in WCF also makes it possible for the smart

client to automatically authenticate itself against the WCF Web Service. This subsection does not go into details as to how the WCF Web Service has been implemented. The source code for the WCF Web Service can instead be examined more closely on the enclosed CD-ROM.

Table 13 provides an overview and a short description of the individual methods exposed by the WCF Web Service, which consists of methods for reading data for the database directly and methods that utilize the Maconomy Web Service.

- **GetJobs** () As DataSet

  Get all jobs for the authenticated user.

- **JobsChanged** (hash As string) As boolean

  Indicates whether jobs has been changed since a given hash value.

- **GetTaskLists** () As DataTable

  Get all task lists for the authenticated user.

- **TaskListsChanged** (hash As string) As boolean

  Indicates whether task lists has been changed since a given hash value.

- **GetTimeSheets** () As DataTable

  Get all time sheets for the authenticated user.

- **TimeSheetsChanged** (hash As string) As boolean

  Indicates whether time sheets has been changed since a given hash value.

- **GetTimeSheetLines** () As DataTable

  Get all time sheet lines for the authenticated user.

- **TimeSheetLinesChanged** (hash As string) As boolean

  Indicates whether time sheet lines has been changed since a given hash value.

- **CreateTimeSheetLine** (*periodStart* As date, *jobNumber* As string, *taskName* As string, *note* As string, *numberOfDay1* As double, *numberOfDay2* As double, *numberOfDay3* As double, *numberOfDay4* As double, *numberOfDay5* As double, *numberOfDay6* As double, *numberOfDay7* As double, *permanentLine* As long) As string

  Creates a time sheet line and return the corresponding InstanceKey.

- **DeleteTimeSheetLine** (*instanceKey* As string) As boolean

  Deletes a time sheet line and returns true if successful.

- **SetHours** (*instanceKey* As string, *dayInWeek* As long, *hours* As double) As boolean

  Sets hours on a time sheet line for the day defined by dayInWeek and returns true if successful.

- **SetJobNumberAndTaskName** (*instanceKey* As string, *jobNumber* As string, *taskName* As string) As boolean

  Sets Job Number and Task Name on a time sheet line and returns true if successful.

- **SetNote** (*instanceKey* As string, *note* As string) As boolean

  Sets Note on a time sheet line and returns true if successful.

- **SetPermanentLine** (*instanceKey* As string, *permanentLine* As long) As boolean

  Sets PermanentLine on a time sheet line and returns true if successful.

▪ **SubmitTimeSheet** (*periodStart* As date) As boolean Submits a time sheet and returns true if successful.

Table 13: Methods for WCF Web Service

The Web Service methods are based on a CRUD design, which is a requirement for the Web Service to be feasible with the Offline Framework.

6.3.3 Maconomy Smart Client

The purpose of this subsection is to provide a brief overview of the implementation made in the construction of the Maconomy Smart Client for the case study. The Maconomy Smart Client utilizes the WCF Web Service discussed in section 6.3.2 to interact with the Maconomy ERP system. This subsection also acts as an example of how the Offline Framework can be used to provide offline/online support in smart clients by illustrating how the Offline Framework is configured and initialized in the Maconomy Smart Client.

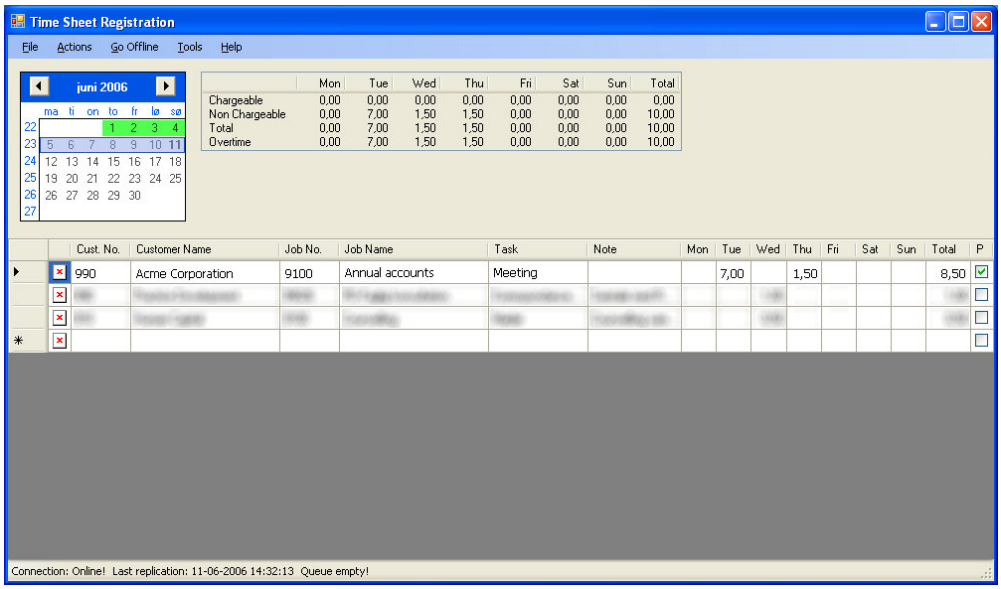


Figure 33: Maconomy Smart Client

A screen shot of the Maconomy Smart Client is shown in Figure 33 to give an understanding of how the final result looks. The user interface is inspired by the Maconomy Web Application and therefore basically has the same layout, which will make the users able to quickly understand how the application works. The grid displaying the individual time sheet lines has been improved by providing an auto-completion functionality for the Customer Number, Customer Name, Job Number and Job Name columns and a dropdown list for the Task column displaying the description of the relevant tasks instead of the task number. These two improvements simplify the creation and modification of time sheet lines

dramatically and have made the Maconomy Smart Client much more responsive and user-friendly compared to the existing Maconomy Web Application. The check box in the P column is provided for the user to easily be able to mark time sheet lines as permanent, which makes it possible to quickly insert these permanent lines in a new time sheet.

For the Maconomy Smart Client to be able to communicate with the WCF Web Service a proxy class must be created. This proxy class can be automatically generated by a tool included in the WinFX SDK, which utilizes the WSDL information provided in the Web Service. The proxy class is configured using a XML file that contains the necessary information for the proxy class to work. Table 14 illustrates how the WCF proxy has been configured.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name=""
        address="http://dk-macdev001/SmartClient/MaconomyService.svc"
        binding="wsHttpBinding"
        bindingConfiguration="Binding1"
        contract="IMaconomyService"/>
    </client>
    <bindings>
      <wsHttpBinding>
        <binding name="Binding1" maxReceivedMessageSize="10000000">
          <security mode="Message">
            <message clientCredentialType="Windows" />
          </security>
         <reliableSession enabled="true" />
        </binding>
      </wsHttpBinding>
    </bindings>
    ...
  </system.serviceModel>
</configuration>
```

Table 14: Configuration for the MaconomyServiceProxy

The binding chosen is the wsHttpBinding, which is a WCF binding that supports the new Web Service standards described in section 4.1. The maxReceivedMessageSize property is used to configure a maximum of how much data the client is able to receive from a Web Service response. The value has been configured to 10000000 bytes, because the messages sent from the WCF Web Service in the case study can contain relatively large amounts of data.

Because Web Services use XML as the payload format as described in section 2.2.2, there can be a relatively large overhead when sending data. For this reason it has been decided to compress the XML data exchanged to improve bandwidth utilization. To achieve this, a third-party component[16] has been utilized to provide the desired compression and decompression.

[16] The component "WS-Compression for WCF" can be found at http://weblogs.asp.net/cibrax/archive/2006/03/29/441398.aspx

The communication is made secure by enabling message security with Windows authentication. If possible, the authentication is performed using Kerberos authentication, if not NTLM authentication is used. The binding has also been configured for reliable sessions, which ensure that messages are delivered exactly-once as discussed in section 4.1.2.

The Offline Framework is used together with the implemented proxy class to provide the needed offline support in the Maconomy Smart Client. Table 15 illustrates how this is achieved in the Maconomy Smart Client. The Offline Framework is initialized by creating an instance of the OfflineController and providing it with a Type object of the proxy class that must be used, in this case the MaconomyServiceProxy. Using this Type object the OfflineController can create new instances of the proxy class when needed. When the OfflineController has been correctly configured, the OfflineController can be started using its start() method. This activates the thread for refreshing data and queuing outgoing calls to the Web Service, which is the process described in section 5.1.

```
// Initialize offline controller with proxy
MaconomyServiceProxy proxy = new MaconomyServiceProxy();
OfflineController controller = new OfflineController(proxy.GetType());

// The controller must be configured here. This is described later.

// The controller thread for queuing and replicating data is started
controller.Start();
```

Table 15: Sample of how to initialize the framework

The configuration is a very essential part of the Offline Framework and describes how changes made to the local database should be transformed into Web Service calls and what Web Service calls that can be used for replicating data.

Table 16 illustrates a sample of the configuration used in the Maconomy Smart Client. The sample provides a basic understanding of how the Offline Framework can be configured and is described using comments. The tables that must be available offline must be configured for the Offline Framework to be able to handle the data correctly. Table 16 illustrates the most relevant parts needed for configuring the table TimeSheetLines.

```
// Create an instance of the OfflineTable to hold the configuration for
// the TimeSheetLines table, where "INSTANCEKEY" is a server-generated identity.
OfflineTable cmd = new OfflineTable("TimeSheetLines", "INSTANCEKEY");

// Configures refresh interval to every 2 minutes.
cmd.RefreshInterval = new TimeSpan(0, 2, 0);

// Configures primary key and indexes.
cmd.PrimaryKey = new string[] { "InstanceKey" };
cmd.Indexes.Add(new string[] { "PeriodStart" });

...

// Configures a QueryCommand for replicating data
// In this case a special HashQueryCommand is used to
```

```
// only replicate if any changes have occurred on server.
cmd.QueryCommand = new HashQueryCommand("GetTimeSheetLines",
"TimeSheetLinesChanged", "DataSetName");

// Configures the insert command for the TimeSheetLines table,
// which uses the Web Service method CreateTimeSheetLine. The string array
// specifies what column values is to be used as parameters for the Web Service
// request.
cmd.InsertCommand = new InsertCommand("CreateTimeSheetLine", new string[] {
"PERIODSTART", "JOBNUMBER", "TASKNAME", "NOTE", "NUMBEROFDAY1", "NUMBEROFDAY2",
"NUMBEROFDAY3", "NUMBEROFDAY4", "NUMBEROFDAY5", "NUMBEROFDAY6", "NUMBEROFDAY7",
"PERMANENTLINE" });

...

// Configures an update command for the TimeSheetLines table
// that uses the Web Service method SetNote(), which is
// triggered when changes occur in the column "NOTE" in the local database.
// If a change occur the belonging INSTANCEKEY and new NOTE is send as parameter
// in a Web Service request.
updateCmd = new UpdateCommand("SetNote");
updateCmd.ColumnTriggers.Add("NOTE");
updateCmd.ColumnArgs.Add(new UpdateCommandArg("INSTANCEKEY", false, true));
updateCmd.ColumnArgs.Add(new UpdateCommandArg("NOTE"));
cmd.UpdateCommands.Add(updateCmd);

...

// Configures a delete command for the TimeSheetLines table
// that uses the web service method DeleteTimeSheetLine() with
// the parameter INSTANCEKEY from the local database
cmd.DeleteCommand = new DeleteCommand("DeleteTimeSheetLine", "INSTANCEKEY");

// Adds the OfflineTable to the list of tables configured
controller.OfflineTables.Add(cmd.TableName, cmd);

...
```

Table 16: Sample of how to configure the OfflineController

When the Offline Framework has been configured, queries and updates can easily be made to the local database using a DataAdapter. The GetDataAdapter() method on the configured OfflineController is used to retrieve such a DataAdapter.

```
string sqlQuery = "SELECT * FROM TimeSheetLines";
FbDataAdapter dataAdapter = controller.GetDataAdapter(sqlQuery);
DataTable timeSheetLines = new DataTable("TimeSheets");
dataAdapter.Fill(timeSheetLines);

// Make changes to the DataTable timeSheetLines here
// The DataTable could e.g. be bound to a DataGridView

dataAdapter.Update(timeSheetLines);
```

Table 17: Sample of how to make queries and updates using the DataAdapter

Table 17 briefly illustrates how easily changes can be made to the local data storage without the developer needing to concentrate on whether the smart client is online or offline. The development of smart client applications is in this way simplified with the use the Offline Framework.

6.4 Test

Several tests have been performed to examine whether the application is operating as expected. The main focus has been put on performing functional tests, because the developed Offline Framework and the smart client solution are prototypes, and such functional tests quickly reveal whether the concept of the overall system is functional. The functional tests performed can be found in Appendix C. All the functional tests were performed with a successful result.

Unit testing of the Offline Framework has to some extend been performed. These tests were made using the unit-testing framework NUnit[17] and can be found on the enclosed CD-ROM under the OfflineFramework_test project in the SmartClient directory. The main focus has not been put on performing unit tests, because these tests primarily examine the individual methods and not whether the overall concept of the smart client solution is working as expected, which in this case is most relevant.

The smart client solution has furthermore been tested by performing a user test in the production environment of PwC's ERP system, Maconomy, by two employees, which have functioned as test persons. The user test illustrated that the smart client solution was operating correctly in the production environment and only a few minor bugs was reported by the two test persons. The bugs reported were mainly related to the functionality of the smart client application and not the functionality provided by the Offline Framework. The minor bugs reported and a change request can be found in Appendix D. Generally, the test persons were very satisfied with the smart client solution, especially in relation to the great responsiveness of the smart client compared with the existing web portal solution.

The performed tests have illustrated that the developed Offline Framework and smart client solution is functioning correctly with only some minor bugs. The case study has in this way illustrated a proof of concept of the Offline Framework.

6.5 Summary

In this chapter, a case study has been examined to give the reader an understanding of how to utilize the Offline Framework. The Maconomy ERP system has been analysed, and an overall design of the solution used in the case study has been provided.

An overview of the relevant data structure and the implemented Web Services have been briefly discussed to give an understanding of the implemented smart client solution and how the WCF Web Service has been configured with the Offline Framework.

Finally, functional tests and user tests have been performed successfully. The Maconomy Smart Client proves that the concept of the implemented Offline Framework can be used to ease the development of offline capable smart client applications.

[17] NUnit can be found at http://www.nunit.org.

7 Future Work

After having analysed, designed and implemented a prototype of an Offline Framework, there are still some issues that have to be solved. The most important issues are as follows:

- **More efficient synchronization**
  In section 3.2.1, an approach for supporting two-way synchronization was presented. This approach has not been implemented because it was estimated that the approach could not be implemented within the timeframe of this master thesis project. Analyzing how to support two-way synchronization without changing the existing application server and database server is future work.

- **Conflict detection and resolution**
  Conflict detection and resolution, which is discussed in section 3.2.2, has not been implemented because this area must be analysed further in order to find a feasible solution, which can comply with the requirements of the problem domain. The fact that the existing system cannot be changed combined with no available metadata makes it very difficult to implement conflict detection and resolution. A possible solution is to provide the necessary metadata by implementing the Change Tracking Service discussed in section 3.2.1.

- **Transaction support**
  The Queue Manager in the Offline Framework currently handles the changes made to the local data storage in separate transactions when synchronization is performed. The Offline Framework should be extended so that the WS-Transaction standard supported by the Windows Communication Foundation is utilized to ensure that multiple changes made to the local data storage in a transaction also are executed in a transaction by the Queue Manager, when connectivity is achieved.

- **Support for multiple Web Services**
  The prototype of the Offline Framework currently only supports interaction with one Web Service. The Offline Framework should be extended so that it can be configured to interact with multiple Web Services. This feature is not difficult to achieve in the current version of the Offline Framework, but was not implemented in the prototype because it was not a necessary feature to illustrate the proof of concept of the Offline Framework.

This was an overview of some the main areas which must be examined in details in the future. The main challenge of these areas is to retain the simplicity of the Offline Framework

when adding more functionality, because there is a risk that these areas can make the framework more complex to use.

8 Conclusion

The main objective of this master thesis has been to analyze and design a generic and flexible framework that can be used to develop smart clients that must provide offline/online support to existing systems. The following results have been obtained in this master thesis project.

The main design architectures of occasionally connected applications have been analysed and a new Hybrid Architecture have been presented, which combines some of the advantages of the Data-Centric Architecture on top of the Service-Oriented Architecture. Based on further analysis, a Built-in Cache Framework solution was chosen to provide the offline functionality necessary in the Hybrid Architecture. The combination of the Hybrid Architecture and the Built-in Cache Framework resulted in a solution strategy, which was used as the foundation for further analysis, design and implementation of an Offline Framework.

The overall challenge of this master thesis project has been to provide a solution that does not require the existing systems to be changed. The solution must utilize simple Web Services to interact with these systems, which makes it difficult to achieve efficient synchronization and replication. The topic requires more research and is considered as future work.

An analysis of the most important challenges of the Offline Framework has been provided. The challenges analysed were how to provide a local data storage and how to achieve efficient synchronization and replication in the Offline Framework. An embedded database strategy was chosen for the local data storage and the analysis presented solutions on how to achieve efficient synchronization and replication in the Offline Framework.

Based on the research conducted, a prototype of the Offline Framework has been designed and implemented. The prototype was utilized in a case study in cooperation with PricewaterhouseCoopers, which resulted in a working offline capable smart client solution that fulfilled the specification given by PricewaterhouseCoopers. The smart client developed utilizing the Offline Framework resulted in an advanced and user-friendly client application for making time registrations in the Maconomy ERP system at PricewaterhouseCoopers. This smart client illustrated that the objective of this master thesis project has been successfully achieved.

The implemented Offline Framework successfully achieved the desired advantages of the Data-Centric Architecture by simplifying the implementation of caching and updating of local data and providing advanced data querying functionality, which makes implementation of smart clients using the Offline Framework time-saving. However, the developed Offline Framework must be seen as a prototype that requires future work in order to become fully stable and suitable for production environments.

Bibliography

[1] D. Hill, B. Webster, E. A. Jezierski, S. Vasireddy, M. Al-Sabt, B.Wastell, J. Rasmusson, P.Gale and P. Slater
"Smart Client Architecture and Design Guide"
Microsoft Corporation, 2004

[2] Justin Reabow and Darryl Pillay
"Caching of XML Web Services to Support Disconnected Operation"
Department of Computer Science, University of Cape Town, October 2004

[3] Venugopalan Ramasubramanian and Douglas B. Terry
Caching of XML Web Services for Disconnected Operation
Microsoft Research, December 2004

[4] "Smart Client Offline Application Block"
Microsoft Corporation, 2004

[5] Keith Yedlin
"Extracting Business Value Through Smart Clients"
Microsoft Corporation, PowerPoint Presentation, May 2005
http://www.windowsforms.net/Samples/Go%20To%20Market/Smart%20Client/Smart%20Client.ppt

[6] Douglas K. Barry
"Web Services and Service-Oriented Architectures"
Barry & Associates, Inc.
http://www.service-architecture.com

[7] Dennis Sosnoski
"Improve XML transport performance, Part 1"
Sosnoske Software Solutions, Inc., June 2004
http://www-128.ibm.com/developerworks/xml/library/x-trans1.html

[8] "Reliable Message Delivery in a Web Services World: A Proposed Architeture and Roadmap"
IBM Corporation and Microsoft Corporation, White Paper, March 2003
http://www-128.ibm.com/developerworks/library/specification/ws-rmdev

[9] William Cox, Felipe Cabrera, George Copeland, Tom Freund, Johannes Klein, Tony Storey, Satish Thatte
"Web Services Transaction (WS-Transaction)"
BEA Systems, International Business Machines Corporation, Microsoft Corporation, January 2004
http://dev2dev.bea.com/pub/a/2004/01/ws-transaction.html

[10] William Tay
"Securing Your WCF Service"
TheServerSide.NET, March 2006
http://www.theserverside.net/tt/articles/showarticle.tss?id=SecuringWCFService

[11] Anthony Nadalin, Chris Kaler, Ronald Monzillo, Phillip Hallam-Baker
"Web Services Security UsernameToken Profile 1.1"
OASIS Standard Specification, February 2006
http://www.oasis-open.org/committees/download.php/16782/
wss-v1.1-spec-os-UsernameTokenProfile.pdf

[12] Robyn Lorusso
"Learning Guide: Authentication"
SearchWindowsSecurity.com, March 2005
http://searchwindowssecurity.techtarget.com/originalContent/
0,289142,sid45_gci1067230,00.html

[13] Jan De Clercq
"Windows Server 2003 security infrastructures: Chapter 5, 'Kerberos'"
Elsevier Digital Press, 2004
http://searchwindowssecurity.techtarget.com/searchWindowsSecurity/
downloads/DeClercq05.pdf

[14] Shweta Gaur
"Security Options for .NET Web Services"
SYS-CON Media, Inc., 2004
http://www2.sys-con.com/ITSG/virtualcd/Dotnet/archives/0107/
guar/index.html

[15] Scott Seely
"Understanding WS-Security"
Microsoft Corporation, October 2002

http://msdn.microsoft.com/library/en-us/dnwssecur/html/understw.asp

[16] "Windows Communication Foundation"
Microsoft Corporation
http://windowssdk.msdn.microsoft.com/library/default.asp?url=/library/en-us/WCF_con/html/fd327ade-0260-4c40-adbe-b74645ba3277.asp

[17] "Programming the Windows Communication Foundation"
Microsoft Corporation, PowerPoint Presentation
http://windowscommunication.net/collateral/downloads/ProgrammingWCF.ppt

[18] Glenn Johnson
"Programming Microsoft ADO.NET 2.0 Applications: Advanced Topics"
Microsoft Press, 2006

[19] "ADO.NET Architecture"
Microsoft Corporation, MSDN Article
http://msdn2.microsoft.com/en-us/library/27y4ybxw.aspx

[20] "Sync4j SyncServer Developer's Guide"
Funambol
http://download.forge.objectweb.org/sync4j/sync4j_server_ds_developer_guide.pdf

[21] Dino Esposito
"What's New in ADO.NET 2.0 for SQL Developers"
Developer.com, April 2005
http://www.developer.com/db/article.php/3494396

[22] Anthony D. Joseph, Joshua A. Tauber and M. Frans Kaashoek
"Mobil Computer with the Rover Toolkit"
M.I.T. Laboratory for Computer Science, Cambridge, U.S.A., 1997

[23] "Principle of least privilege"
Wikipedia, April, 2006
http://en.wikipedia.org/wiki/Principle_of_least_privilege

[24] "Smart Client Definition"
Microsoft Developer Network
http://msdn.microsoft.com/smartclient/understanding/definition/

[25] Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana
"Web Services Description Language (WSDL) 1.1"
World Wide Web Consortium, Note, March 2001
http://www.w3.org/TR/wsdl

[26] Dan Letecky
"Using Firebird SQL in .NET"
DotNetFirebird, February 2005
http://www.dotnetfirebird.org

[27] "About Firebird"
http://www.firebirdsql.org/index.php

A Content on CD-ROM

The enclosed CD-ROM contains the following directories:

- **Maconomy**
  The Maconomy directory contains the Maconomy Web Service implemented. The Web Service is written using the M-Script programming language and is contained in the file TimeSheetService.1.ms.

- **SmartClient**
  The SmartClient directory contains the Visual Studio 2005 solution file, which gathers the different project used in the implementation of this master thesis project.

  The solution contains the following projects:

  - Client:
    This project contains the implementation of the GUI interface.

  - MonthCalendar:
    This project contains a third-party component, which is utilized by the Client project.

  - OfflineFramework:
    This project contains the implementation of the Offline Framework.

  - OfflineFramework_test:
    This project contains the unit tests performed.

  - Service:
    This project contains the implementation of the WCF Web Service.

B Installation Requirements

The following requirements must be fulfilled to be able to utilize the solution made in this master thesis project:

- **Oracle database**
  The solution developed requires a username and password for an Oracle database user with read access to the following tables in the Maconomy ERP system: UserInformation, Employee, TimeSheetHeader, TimeSheetLine, JobHeader, TaskListLine.

- **Maconomy ERP system**
  The solution developed sets the following requirements to the Maconomy ERP system:

  - InstanceKey functionality enabled on time sheet lines.
  - The implemented TimeSheetService is correctly installed.
  - There exists a possibility to map a Windows domain user to an employee in the Maconomy ERP system

- **Client computers**
  The solution developed depends on that the following components are installed on the client computers:

  - Microsoft .NET Framework 2.0
  - WinFX Runtime Components – February CTP

C Functional Tests

The following functional tests have been performed on the smart client application developed in the case study, which utilizes the Offline Framework:

- **Test 1:**
  Create a new time sheet line with a selected job, task, note and hours
  Expected result:
  A time sheet line is created in the local database and a request to the CreateTimeSheetLine() method is queued for later execution. When the application gets online the request is executed against the Web Service.

- **Test 2:**
  Change the job and task on an existing time sheet line.
  Expected result:
  The job and task on a time sheet line are changed in the local database and a request to the SetJobNumberAndTaskName() method is queued for later execution. When the application gets online the request is executed against the Web Service.

- **Test 3:**
  Change the note on an existing time sheet line.
  Expected result:
  The note on a time sheet line is changed in the local database and a request to the SetNote() method is queued for later execution. When the application gets online the request is executed against the Web Service.

- **Test 4:**
  Change the hours on an existing time sheet line.
  Expected result:
  The hours on a time sheet line is changed in the local database and a request to the SetHours() method is queued for later execution. When the application gets online the request is executed against the Web Service.

- **Test 5:**
  Delete an existing time sheet line.
  Expected result:
  The selected time sheet line is deleted in the local database and a request to the DeleteTimeSheetLine() method is queued for later execution. When the application gets online the request is executed against the Web Service.

- **Test 6:**
  Try to create a new time sheet line in a submitted time sheet.
  Expected result:
  This should not be possible.

- **Test 7:**
  Physically remove network connectivity.
  Expected result:
  The application should automatically go offline.

- **Test 8:**
  Physically attach network connectivity.
  Expected result:
  The application should automatically go online.

- **Test 9:**
  When the application is online the user selects to take offline
  Expected result:
  The application should be forced into offline mode.

- **Test 10:**
  When the application is forced in to offline mode the user selects to go online.
  Expected result:
  The application should go into online mode.

D Feedback from User Test

The user test performed on the production system of the Maconomy ERP system at PricewaterhouseCoopers resulted in the following bug reports and change requests:

Bug reports:

- Removal of all text in the 'Note' field gives a '"System.DBNull' cannot be converted to a 'System.String"' error message when the change is tried executed by the queue. Reported by Kim Bryndum on 16 May 2006.
- When error box from the queue is displayed the main program window is not locked, which gives the user the ability to make changes to data that makes the program freeze. Reported by Kim Bryndum on 16 May 2006.
- Nothing happens when trying to start the application (WinFX is not installed). Reported by Kim Bryndum on 22 May 2006.
- A time sheet submitted in the Maconomy Web Portal is not registered correctly as submitted by in the smart client application. Reported by Søren Burchardt on 30 May 2006.

Change requests:

- Better information to the user when a replication is occurring, e.g. in the form of a message box showing the progress of the replication and locking user access to the main program. Suggested by Kim Bryndum on 16 May 2006.