Final thesis

Evaluation of development platforms building financial online applications

by

Daniel Svensson

LIU-IDA/LITH-EX-A—11/025—SE

2011-06-21
The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Daniel Svensson, 2011
Final Thesis

Evaluation of development platforms
building financial online applications

By
Daniel Svensson

LIU-IDA/LITH-EX-A—11/025—SE

2011-06-21

Supervisor: Pontus Klamfeldt
CRM Treasury Systems AB

Examiner: Kristian Sandahl
Department of Computer and Information Science, Linköping University
Abstract

CRM Treasury Systems AB is one of the Nordic Countries’ leading companies within treasury management. Their main product CRM Finance is a software program for financial administrators with roots back to the early 90s. Computer usage and software has changed considerably since then and the company is currently considering choosing a new technology to base future products upon.

In this thesis Oracle ADF, Microsoft Lightswitch beta 1 and Silverlight with WCF RIA Services, are compared in order to select a suitable technology to base future applications on.

The frameworks are used to implement a simple application with a small number of screens displaying database data in order to get to know the frameworks and capture the time required to get started. After the basic applications have been implemented in all the three frameworks and evaluated it is decided to continue by implementing a number of more advanced authorization and internationalization scenarios. It is decided to continue with only two of the selected frameworks due to lack of time once the simple application has been implemented successfully. Lightswitch beta 1 is chosen for discontinuation due to an uncertain future for the application as it is the first public available beta of the product.

Based on the experience from the implementations performed, the time required for them as well as how the platforms fit the company’s requirements the .NET platform with Silverlight + WCF RIA Services is recommended as the base for future applications.
Acknowledgements

I would like to thank CRM Treasury Systems AB for allowing me to perform this thesis and for trusting me to choose technology for their future software. I would especially like to thank my supervisor Pontus Klamfeldt for the time spent in order to allow me to perform the thesis.

I would also like to thank my parents Peter Svensson and Monika Svensson for providing me and my fiancée with a home until we had our own, allowing me to work with the thesis at the company.

This thesis would not have been possible if it was not for the support of my fiancée Linda Larsson who has supported me throughout the whole thesis, both with proofreading and allowing me to spend the time required for it.
# Table of contents

1 Introduction........................................................................................................................................................................1
   1.1 Background ........................................................................................................................................................................1
   1.2 Purpose and problem statement............................................................................................................................................2
   1.3 Scope and limitations...............................................................................................................................................................3
   1.4 Methodology............................................................................................................................................................................3
   1.5 Report structure.....................................................................................................................................................................4
   1.6 Glossary ...................................................................................................................................................................................4
   1.7 Abbreviations .........................................................................................................................................................................9

2 Design theory ..............................................................................................................................................................................10
   2.1 Architecture ...........................................................................................................................................................................10
       2.1.1 Layering .........................................................................................................................................................................10
       2.1.2 Client-Server, 3-tier and N-tier...................................................................................................................................11
   2.2 Design ....................................................................................................................................................................................12
       2.2.1 Model-View-ViewModel (MVVM)................................................................................................................................12
       2.2.2 Model View Controller (MVC)................................................................................................................................14

3 Environments under evaluation ...................................................................................................................................................16
   3.1 Oracle ADF .............................................................................................................................................................................16
       3.1.1 Overview .......................................................................................................................................................................16
       3.1.2 ADF BC the Business Service ....................................................................................................................................17
       3.1.3 ADF View layer ...............................................................................................................................................................17
       3.1.4 Validation .........................................................................................................................................................................21
       3.1.5 Authentication and Authorization ................................................................................................................................21
       3.1.6 Internationalization .......................................................................................................................................................21
   3.2 .NET Framework: Silverlight and WCF RIA Services .............................................................................................................23
       3.2.1 Overview .......................................................................................................................................................................23
       3.2.2 Silverlight – The Presentation Layer ................................................................................................................................24
       3.2.3 Entity Framework – The Data Access framework ........................................................................................................25
       3.2.4 WCF RIA Services – Hosting the business layer ..........................................................................................................26
       3.2.5 Validation .........................................................................................................................................................................29
       3.2.6 Authentication and authorization ..................................................................................................................................29
List of figures
Figure 1 An example of layering ........................................................................................................... 10
Figure 2 A N-tier architecture ............................................................................................................. 11
Figure 3 Overview of the MVVM pattern ............................................................................................ 12
Figure 4 Overview of the MVC design pattern .................................................................................... 14
Figure 5 Oracle ADF architecture ...................................................................................................... 16
Figure 6 Task flow for the ADF application ......................................................................................... 18
Figure 7 Silverlight application architecture ....................................................................................... 23
Figure 8 Overview of WCF RIA Services ............................................................................................. 27
Figure 9 Visual Studio's resource editor ............................................................................................... 30
Figure 10 Lightswitch architecture ..................................................................................................... 32
Figure 11 A lightswitch project ........................................................................................................... 32
Figure 12 Lightswitch standard shell with 3 screens opened ............................................................... 34
Figure 13 Lightswitch UI designer ....................................................................................................... 35
Figure 14 Navigation flow of the basic applications .......................................................................... 39
Figure 15 Database layout used .......................................................................................................... 40
Figure 16 Initial look of ADF program ................................................................................................. 45
Figure 17 Entity objects generated by ADF ........................................................................................ 45
Figure 18 Silverlight business project template .................................................................................... 48
Figure 19 JDeveloper dialog for configuring update permissions ....................................................... 51
Figure 20 Time used to implement basic application ......................................................................... 66
Figure 21 Number of search results for the different technologies .................................................... 67
Figure 22 Time required for adding authentication and authorization ................................................. 73
Figure 23 Time used for implementing localization .......................................................................... 75
Figure 24 Startup screen ...................................................................................................................... i
Figure 25 Choose selection screen ...................................................................................................... i
Figure 26 Transaction screen ............................................................................................................... ii
Figure 27 Settings screen .................................................................................................................... ii
Figure 28 Text and a button in Code example 3 as shown by the JDeveloper designer ...................... iii
Figure 29 The page in Figure 28 as displayed in a web browser ......................................................... iv
Figure 30 DataControls with generated collections and operations for the ADF application .......... v
Figure 31 Text and a button in Code example 4 as displayed by the Visual Studio designer .......... vi
Figure 32 Text and a button in Code example 4 as displayed when running the application .......... vi
Figure 33 Deployment alternatives for Lightswitch application ........................................................... vii
Figure 34 JDeveloper's entity UI where authorization can be enabled for the entity ....................... viii
Figure 35 JDevelopers's "Select Text Resource" window ................................................................. ix
List of code examples

Code example 1 Example XAML for a ComboBox allowing the user to choose between all items in the AvailableCultures collection and updates SelectedItem property upon selection. 13

Code example 2 Part of a ViewModel exposing two properties bound to by the ComboBox in Code example 1. 14

Code example 3 An ADF Faces page displaying the text “Some text” and a button. 19

Code example 4 XAML for a Silverlight page displaying some text and a button 24

Code example 5 Alternative syntax for LINQ statement in Code example 7 26

Code example 6 Example of a WCF RIA Services query function 28

Code example 7 Client side code for querying a DomainService 28

Code example 8 Where-statement limiting visible selections 52

Code example 9 EL expression setting visibility for control depending on role membership 52

Code example 10 WCF RIA Services query method with code to limit the selections returned (yellow background) 54

Code example 11 Code used to enable button based on role membership 54

Code example 12 Example of how an attached property "RequiresRole" could be used in XAML 55

Code example 13 Code for accessing a resource from C# (Top) and Java (Bottom) 76

Code example 14 Server side function placed on ADF BC tier for marking transactions as allowed x

Code example 15 View layer code for calling the confirm function on the ADF BC tier xi

Code example 16 Client side code calling custom update function “ConfirmTransaction” for all selected items of a data-grid in Silverlight xii

Code example 17 A WCF RIA Services “custom update” method for marking transactions as confirmed in the Silverlight prototype xii

Code example 18 WCF RIA code for updating transactions with authorization checks added xii

Code example 19 Implementation of a custom authorization annotation for use with WCF RIA Services xiii

Code example 20 Use of the Display attribute to configure the label for a property xiii

Code example 21 Code used to convert unspecified DateTime from UTC to local time xiii
1 Introduction

1.1 Background

CRM\(^1\) Treasury Systems is a company that has been working within treasury management since 1984. Their main product, CRM Finance is a software program with functionality for loan administration, foreign exchange risk management, cash management etc. The product is used in finance departments, corporate banks and treasury centers of around 70 large multinational and domestic corporations in the Nordic countries as of 2011.

The typical end user is a financial administrator in the corporate sector that is used to working with Microsoft Excel. Large volumes of transactions set the requirement for the customer to use professional software as Excel is usually not accepted by auditors for financial risk management and administration of financial instruments for larger transaction volumes.

CRM Finance was originally developed using Oracle Forms v 2-4 and database v 5 – 6 around 1990-1994. The last client-server version of CRM finance is using Oracle Forms 6 and is based on client server architecture with a fat client, where UI- and business-logic is intermixed. The application runs against an Oracle database hosting both a database and some additional application logic in the form of PL/SQL functions. Currently the program is made up of around 270 screens which are accompanied with a number of separate programs (which can be run from inside the forms application) written in C using embedded SQL. Many of the C programs provide customer-specific import and exports while some contain more general logic. The programs run as windows applications, which required them to be installed and maintained on each pc where they were used.

As the current product is about 20 years old a new foundation based on modern tools and techniques is planned. Software as a service (SaaS) is one of the visions for future applications which among other things promote the ability to access the software through a web browser. The shift of platform is partly a response to that Oracle as of 2009 recommends developers to begin to adopt their newer ADF platform instead of continuing to develop for the forms runtime (1). Another of the main reasons is that the current product has started to suffer from limited testability and a long legacy, with increased maintenance as a result.

This thesis will evaluate some different development environments as the basis to build future applications with a modern look and feel.

---

\(^1\) CRM stands for Currency Risk Management and should not be confused with Customer Relationship Management
1.2 Purpose and problem statement

The aim of this master thesis is to evaluate different platforms and their preferred approach for building modern enterprise level applications. This is performed in order to select a candidate for to base future applications upon as well as to gathering knowledge of best practices for the platforms.

Some of the general requirements for a future development platform were:

- It should be future proof in the sense that
  - The platform used should be widespread, popular with much documentation (the technical platform should be useable for at least the next decade)
  - It should be easy to recruit developers
- The number of programming languages should be kept low so that all developers can work with all parts of the application
- It should be easy to debug the application
- The development platform should support automated testing
- The platform should provide support for authorization, internationalization, as well as validation.

In order to reach the goal the following questions were raised:

- What architectural styles and design patterns are promoted for the different frameworks?
- How easy is it to get started using the different frameworks?
- How are different non-functional features such as authorization, internationalization and validation supported?
- How maintainable and testable does the code become?
- What is the performance of the different platforms?
1.3 Scope and limitations
After some initial search three contestants that matched the initial requirements were selected for evaluation. The environments, which are described in Chapter 3, are:

- Oracle ADF, which is the option promoted by Oracle. This was the technology had been selected by the company as the preliminary future technology to use.
- Microsoft’s .NET platform using Silverlight and WCF RIA Services
- Microsoft Lightswitch that is promoted by Microsoft as a fast way to create high-quality business applications targeting both the desktop as well as the cloud (2) regardless of developer skills.

This evaluation does not pretend to be a comparison between the .NET framework and the Java EE standard since both of these platforms are huge. Instead specific approaches using parts of the selected frameworks were compared.

Choosing database vendor and database design is outside the scope of this thesis. An Oracle 9.2i database was used when developing for ADF while Microsoft’s SQL Server 2008 R2 Express Edition was used when developing for the .NET framework.\(^1\)

1.4 Methodology
A practical approach was decided upon for evaluating the different platforms as it would give hands on experience that is difficult to get by other means. A number of scenarios reflecting real world scenarios were chosen for implementation together with an expert on the current system. Each of the implementation scenarios was specified in detail before starting the implementation to make sure that the same features were implemented for all frameworks.

When evaluating the alternatives their respective online documentation ((3),(4) and (5)) have been used as the primary source for help unless otherwise stated.

After some initial research with the aim to select candidates each of the candidates were used to implement a well specified simple application in order to get to know the different developer environments. The total time required for the implementation were recorded, including time spent debugging and searching for help. The purpose was to estimate how easy the platforms are to start using, to get an overview of how good their documentation are and how easy it is to get help for problems that can arise during development.

After the prototypes had been implemented the alternatives were reduced from three down to two.

\(^1\) Oracle’s latest .NET client at this time (ODAC Entity Framework and LINQ Beta 11.2.0.2.30) was not compatible with Visual Studio 2010 SP1, which is the reason for using different databases during the development.
Authorization and internationalization were then implemented into the Oracle ADF and .NET application to get insight into how much effort it takes to develop software with more complex features. These features were chosen as they were regarded as important features that had to be easily implemented and maintained.

1.5 Report structure

**Chapter 2** Contains some background theory regarding architecture and design of software applications.

**Chapter 3** Presents the different platforms evaluated.

**Chapter 4** Describes the different evaluation areas and corresponding application scenarios that were later implemented.

**Chapter 5** Describes the implementation of the evaluation scenarios from Chapter 4 using the different platforms.

**Chapter 6** Contains a discussion around the implementations for the different platforms where each implementation area is concluded with a recommended platform.

**Chapter 7** Presents a recommendation for the platform to use.

1.6 Glossary

**Annotation & Attributes**
Annotations (using Java vocabulary) or attributes (.NET vocabulary) provide a way to attach metadata to code. The metadata itself do not enforce any specific behavior but can be read by frameworks which may use the metadata. In C# attributes are attached to a function, method or class by writing “[AttributeName]” before its declaration.

**Authentication**
Authentication is the act of confirming the identity of a user. In this thesis authentication is based on the user supplying a username and password which the application then verify.

**Authorization**
Authorization can be said to answer the question “who can do X”, and thus is actually the process of assigning permissions to perform certain action. In this thesis authorization is at some places used with meaning of also implementing access control.

**Cascading Style Sheets (CSS)**
CSS is a language often used to define styles describing the visual appearance of web pages.

**Cohesion**
Cohesion is a measure of how related functionality in a software layer, module or class type is. High cohesion is generally desired within a module as it often makes it easier to
understand, reuse and maintain modules. While high cohesion does not automatically imply great code, low cohesion do most often imply “bad” code that is more difficult to understand and maintain. Examples of this is code mixing low level database access and downloading web pages with application logic with the result that changes to database access must be performed all over the program.

**Common Language Runtime (CLR)**
Common Language Runtime defines a runtime environment for executing .NET applications, it roughly compares to the Java Virtual Machine (JVM).

**Coupling**
Coupling between classes (or software modules) defines the degree to which they depend on each other. If module A access module B it means that module A is coupled to module B. The degree of coupling depends on how much module A dependent on the implementation of module B. Low coupling is a desired property between software modules as it makes it easier to perform changes inside them without affecting other modules.

**Dependency Injection**
Dependency Injection is a design pattern for reducing dependencies between different components. Instead of having a component configuring objects it requires, dependencies are set from outside the object. For more information refer to (6) or (7).

**Entity Framework (EF)**
Entity Framework is an ORM framework that is part of the .NET Framework.

**Expression Language (EL)**
Expression Language is a simple language that can be embedded into properties on JavaServer Faces pages causing the expressions to be executed each time the page is fetched. It supports binary operations such as addition, comparison operators as well as simple if-then-else support.(8)

**Groovy**
Groovy is a dynamically typed scripting language that runs on the Java virtual machine.

**Internationalization (i18n)**
Internationalization is the process of designing computer software for international deployment so that it can be adapted for use in different markets without altering the source code. The adaptations can include translating the application as well as modifying formatting of numbers and dates.

**JavaBean**
An ADF managed bean is a Java class that can be referenced and used from ADF web pages. Object instantiation is controlled by the web server with the lifetime of instances configured using xml.
Java EE
Java Enterprise Edition is built on top of Java Standard Edition ("normal desktop Java") and defines APIs and runtime environments intended for large scale networked application.

JavaServer Faces
Part of the Java EE standard. JavaServer faces defines a standard for building server side user interfaces that generate HTML so they can be accessed using a web browser.

JVM
Java Virtual Machine is the software that executes Java byte-code.

LINQ
Language INtegrated Query is a set of features in the .NET Frameworks for working with collections of data independent of what type of collection it is (in memory array, xml document, database table etc.). C# features built-in SQL like syntax for defining LINQ queries. More details on LINQ can be found in Chapter 3.2.3.

Locale
A locale is a collection of settings that differ between different cultures such as formatting of dates and numbers, how strings are sorted (important for strings including non-English characters such as the Swedish ‘å’, ’ä’ and ’ö’).

Localization
Localization is the act of adapting software for use in a specific region. This can include configuring formatting of numbers and dates as well as translation of application text.

Mono
Mono is an open source project, led by Novell, for building and running .NET applications on a wide variety of hardware and operating systems. Mono includes a CLR implementation, a C# compiler as well as an implementation of the majority if the .NET framework.

Moonlight
Moonlight is an open source implementation of Silverlight based on Mono.

MVC Pattern
Model-View-Controller is a design pattern for implementing graphical applications. See Chapter 2.2.2 for more details.

MVVM Pattern
Model-View-ViewModel is a design patterns for implementing graphical applications, specialized for applications using XAML to describe the graphical interface. See Chapter 2.2.1 for a more detailed explanation.
.NET Framework
The name of the standard library for programs written in C#. It is available to all languages running on the CLR runtime.

Observer Pattern
The observer pattern describes two types of objects where one of the objects, the “observer”, notifies the other that it wants to be notified to changes on the other, the “subject”. The pattern is sometimes called publisher-subscriber and is described in (9).

Oracle ADF
Oracle Application Development Framework is a framework for creating web based Java EE applications. See Chapter 3.1 for a more detailed explanation.

ORM, O/RM
Object-Relational Mapping means that relational data, such as tables and views, are mapped to classes as a mean to represent database objects in a programming language’s type system. ORM tools generally allow automatic conversion between the database representation and the in memory objects representation once a mapping from the database to object has been configured.

PL/SQL
PL/SQL or Procedural Language/Structured Query Language is Oracle’s procedural programming language extension for SQL with syntax similar to that of ADA. PL/SQL programs can be embedded in an Oracle database, run as queries against one or run in Oracle Forms runtime.

Property
In the context of .NET a property behaves like a public variable, but it is internally implemented as a pair of get and set methods. Properties in Java are similar but are made up of two separate functions named setXXX and getXXX where XXX is the name of the property.

PRISM
Being part of Microsoft’s Patterns & Practices, PRISM provides a software library as well as guidance for writing modular and maintainable WPF, Silverlight or Windows Phone applications. It was originally the codename for “Composite Application Guidance for WPF and Silverlight” which was later renamed to the shorter PRISM. (10)

RIA
RIA stands for Rich Internet Application and usually means software run in the web browser with features similar to those of desktop applications. Many RIA are built using browser plug-ins such as Adobe Flash, Java or Silverlight, but advances in web standards allows more and more features to be implemented using plain HTML and JavaScript.
**Selection**
A selection defines search criterions deciding which transactions to show.

**SQL Azure**
An online version of Microsoft’s SQL Server provided as a service.

**Transaction**
A transaction represents the act of selling or buying equities, currencies or other financial instruments.

**VDL**
View Declaration Language – Specifies a syntax that can be used to declare user interfaces for Java Server Faces servers, ex JSP, Facelets.

**Windows Communication Foundation (WCF)**
Windows-Communication-Foundation is a framework for building service oriented features and is part of the .NET Framework since version 3.5. It provides a unified interface for client-server communication using any of a number of different communication protocols such as SOAP, JSON, COM or binary (.NET specific).

**Windows Azure**
Azure is Microsoft’s cloud platform offering among other things virtual machines, and a runtime environment for cloud based applications.

**Windows Presentation Foundation (WPF)**
Windows Presentation Foundation is the desktop counterpart to Silverlight. It has been part of the .NET Framework since version 3.5 and can be viewed as a superset of Silverlight targeting desktop applications.

**XAML**
XAML (eXtensible Application Markup Language) is an xml based language used for describing the UI in Microsoft Silverlight and its desktop counterpart WPF. It is based on xml and allows objects to be defined and created as well as defines extensions for allowing custom logic to be run at parse time when the xml file is translated into an object graph.
1.7 Abbreviations

- **ADF** Application Development Framework
- **BC** Business Components (in the context of ADF)
- **CSS** Cascading Style Sheets
- **EF** Entity Framework
- **EL** Expression Language
- **IDE** Integrated Development Environment
- **JSF** Java Server Faces
- **ORM** Object Relational Mapping
- **PRISM** see PRISM in glossary
- **RC** Rich Client (in the context of ADF)
- **RIA** Rich Internet Application
- **UI** User interface
- **VB.NET** Visual Basic .NET (a programming language)
- **WCF** Windows Communication Foundation
- **WPF** Windows Presentation Foundation
- **XAML** eXtensible Application Markup Language: see glossary
- **XML** eXtensible Markup Language
2 Design theory

2.1 Architecture

2.1.1 Layering

In a layered architecture logically related components are grouped together into larger pieces called layers. The communication between layers is often restricted through defined interfaces, which describe the functionality exposed from a layer to another layer. The main reasons to divide a system into layers is to allow high level reasoning about the system as well as to reduce coupling within the system.

In a layered system each layer often builds on the layer below it, adding higher level functionality. Depending on how strict the layering is the requirements on how components in the different layers communicate will differ. When using very strict layering components in a layer should only interact with components in the layer directly below or above, while less strict layering allows a component to access all layers below it.

A RIA application can be divided into presentation layer, business layer and data access layer as in Figure 1. The presentation layer is responsible for interacting with the user. If the user wants to perform a business case, such as placing an order, the presentation layer will forward the request to the business layer. The business layer will in turn implement the logic required for placing an order.

Placing an order will probably involve doing some database operations, but instead of accessing the database from the business layer the business layer will ask the database layer to perform the required data accesses.

Some advantages with layering is that changes within a layer can be performed without affecting components in other layers, unless the interface is affected in which case at most one other layer should be affected. In the example RIA application above it could be possible to change database layout without affecting any of the other layers.

Another advantage of layering is that it promotes separation of concern, allowing the business layer code to care about the business layer functionality rather than about where and how to best access the data.
2.1.2 Client-Server, 3-tier and N-tier

Client-Server (2-tier), 3-tier and N-tier architectures describe how the software is deployed. Tiered architectures have some similarities with layering, but instead of focusing on dividing a system into logical groups of software components tiered architectures divide the software into different processes. These processes can then be deployed either on the same machine or on different machines (see Figure 2). The division into separate processes does not force layering, but it does facilitate layering as the different tiers are often given separate responsibilities.

**2-tier**

In traditional client-server architectures the system is deployed into 2-tiers, a client tier which the user interacts with and a server tier which the client communicates with. An example of a client-server scenario is when browsing the web, where the web browser (client) sends requests to download pages to the web server. Another is applications written in Oracle forms, where the client is the forms-application which sends requests to the database (server). Client server systems are often divided into systems with fat clients, where most of the functionality resides on the client, and thin clients, where the server implements most of the functionality.

**3-tier and N-tier**

In a 3-tier scenario the application is often divided into one (or many) presentation-tier(s), logic tier and data tier. The tiers can be deployed on different machines or on the same computer. N-tier architecture is a generalization of the 3-tier architecture that allows any number of tiers. It is however most often used to talk about architectures with more than 3 tiers. In the rest of this chapter N-tier is used to describe multi-tiered architectures with 3 or more tiers. Using an N-tier approach can facilitate the development of UI development for many different platforms by placing logic at a middle tier where it can be shared by all clients.
N-tier architectures can offer improved scalability and reliability compared to client-server architectures by allowing load balancing and redundancy to be configured at each tier. An application with much CPU intensive logic processing could deploy the business logic tier to multiple computers while using a single computer for the data access.

N-tier architectures also have advantages over 2-tier architectures when it comes to implementing security. It can enable clients outside a company network to access the content of a database through a middle tier while still preventing direct access to the database, which can be hidden behind a firewall\(^1\). Middle tiers running critical code can be deployed further inside the network on restricted machines to prevent tampering with the software.

2.2 Design

2.2.1 Model-View-ViewModel (MVVM)

MVVM is a design pattern that has gained popularity among Silverlight developers. Its use is promoted by Microsoft Patterns & Practices (10 p. 53) and all four sessions concerning general Silverlight development\(^2\) at Microsoft’s MIX\(^3\) web developer conference 12-14 April 2011 were tagged with MVVM.

![Figure 3 Overview of the MVVM pattern](image)

MVVM can be seen as a variation of the MVC pattern for Silverlight and WPF or as a layered design. It is heavily based on binding and commanding features of XAML, which is used to describe the graphical interface. Data bindings allow properties of the graphical interfaces to be “bound” to properties of other objects, like ViewModels. Changes to the bound object can be propagated to the View using notifications, with a design similar to the Observer pattern (9). Changes to the View can in turn update the

---

\(^1\) The middle tier should in scenarios like this implement some sanitation of data to prevent malicious data modifications.

\(^2\) The 24 other sessions tagged with Silverlight either focusing on specific features such as new features, media streaming, windows phone development etc.

\(^3\) See [http://live.visitmix.com/](http://live.visitmix.com/) for online material
bound object. Commanding is similar to the Command design pattern \((9)\) with the addition that the commands have a property describing if they can be executed or not. The View can bind UI buttons to commands so that if the ViewModel exposes commands with that name they are executed whenever the button is pressed.

The View, ViewModel and Model can be implemented almost completely independent of each other by defining interfaces between them. Apart from allowing different aspects of the application to be implemented in parallel the separation simplifies testing. Since the ViewModel do not contain any reference to the View, but rely on exposing data and commands for binding, it becomes much easier to test using automated testing than code which interacts with the UI. By using techniques such as Dependency Injection when creating the ViewModel the ViewModel and Model becomes loosely coupled thus further improving the testability of the ViewModel.

**The View**

The View defines the visual appearance such as what controls to display, what visual effects to use and when. To present data such as a numbers, dates or text the data is exposed as properties of the ViewModel and the proper visual control is created and bound to the exposed property.

The View generally consists of a XAML file describing the layout and a “code-behind” file that can implement logic related to the visual representation. Data bindings are used in XAML to allow the View to access and modify properties of the ViewModel.

```xml
<ComboBox ItemsSource="{Binding AvailibleCultures}" SelectedItem="{Binding SelectedCulture, Mode=TwoWay}"/>
```

Code example 1 Example XAML for a ComboBox allowing the user to choose between all items in the AvailibleCultures collection and updates SelectedItem property upon selection.

In the example above the ComboBox will use the AvailibleCultures property of the page’s data-context object (often the ViewModel) as its ItemSource (the items to display). It will set the selected item to the one returned by the SelectedCulture property and will update it whenever the ViewModel’s SelectedCulture changes. The code “Mode=TwoWay” tells the binding to work both ways making changes to the ComboBox’s SelectedItem property propagate to the ViewModel’s SelectedCulture whenever the user selects a new item.

**The ViewModel**

The ViewModel implements application logic and can contain conversion code for adapting data from the Model to better fit the View. ViewModels can be shared among different Views in the same application or can be reused when creating ports for other platforms such as mobile devices. The ViewModel can expose Model data such as collections of entities directly to the View allowing the View to bind directly to the
Model. This is illustrated by the dashed lines in Figure 3. When properly implemented the ViewModel can be easily tested using automated testing.

**Model**
The Model represents data or application state. In Code example 2 the Model consists of the CultureInfo objects. In a more realistic example the Model could consist of data entities exposed by a WCF RIA services, objects from web services, relational databases or just in-memory objects.

```csharp
public class SettingsViewModel ...
{
    /// Property exposing the Cultures the user can choose from
    public CultureInfo[] AvailableCultures { get; private set; }

    /// Property exposing the selected culture
    public CultureInfo SelectedCulture
    {
        public CultureInfo[] AvailableCultures
        {
            public CultureInfo SelectedCulture
            { get { return …; } set { //Called after combo box has changed
                //Update the selected culture, then notify all bound controls it has changed
                base.RaisePropertyChanged("SelectedCulture");
            }
        }
    }
}
```

Code example 2 Part of a ViewModel exposing two properties bound to by the ComboBox in Code example 1.

### 2.2.2 Model View Controller (MVC)
The MVC pattern is a design pattern for decoupling data from application logic. MVC was used for building graphical applications in Smalltalk-80(11) and has since then influenced many graphical frameworks. Some more recent examples of frameworks basing the design around MVC are Ruby on Rails(12), JavaServer Faces and ASP.NET MVC¹. There are many variations on the pattern with differences in actor coupling and interaction. The basics, visualized in Figure 4, are described here.

![Figure 4 Overview of the MVC design pattern](image_url)

¹ ASP.NET MVC is one of the available approaches for building web applications in the .NET Framework.
**View**
The View implements the graphical user interface. A View often displays as least some data from the Model and updates to the data in the Model are often propagated to the View via some kind of notifications using the observer pattern (9). When the user interacts with the View (like moving the mouse, clicking a button or field) it notifies its Controller about this.

**Model**
Just as in the MVVM design pattern the Model represents data and application state. It should encapsulate data manipulation and business logic. When changed it should notify attached views so they can be updated.

**Controller**
The Controller interacts with both the View and the Model. Requests from the View are sent to the Controller, so it can manipulate the View and Model as required. The communication with the View differs depending on the implementation and can consist of notifications of buttons being pressed as well as URIs navigated to. In the case of Oracle ADF the View sends named actions in the form of strings to the Controller.
3 Environments under evaluation

3.1 Oracle ADF

3.1.1 Overview
Oracle Application Development Framework is a development framework for building software using Oracles Fusion middleware which in turn is built on the Java EE platform. ADF is promoted by Oracle as a way to implement “rich Java EE applications, adhering to standard patterns and practices with greatly reduced effort” (13). The reduced effort comes from the fact that ADF development is supported by the Oracles Java IDE, JDeveloper 11g, which allow much of the development tasks to be performed using a declarative and visual approach.

ADF is not a single technology but, consist of a collection of technologies for implementing different parts of the application. Not all parts of the application must be developed using ADF libraries but other options are available as can be seen in Figure 5. In this thesis the approach recommended by (14 p. xxxiii) with ADF Faces RC (also just called ADF Faces in this thesis), ADF Task Flow and ADF BC is used.

A common Oracle ADF application is divided into two main parts, a business layer and an application layer. The business layer should implement business rules and data access. It can be implemented using many different technologies such as ADF BC, Enterprise Java Beans (EJB), Web Services etc.

The application layer implements the graphical interface and defines how the application should respond to user input. It can be implemented as a Java Swing application or as a web page using either standard Java web server technologies such as JavaServer Pages and JavaServer Faces or using ADF Faces RC, which provides a framework built on top of these web technologies. The view layer will follow the MVC
design pattern when building a web application with Oracle ADF, where the model defines the interface used to access the business layer, the view specifies the user interface and the controller handles user input and manages the application flow.

### 3.1.2 ADF BC the Business Service

The business service layer is where business rules and data access should be implemented. The idea is that it should provide a common place of implementation that can be shared among different graphical interfaces.

An ADF BC project contains a number of Application Modules which are used to group related functionality together just as database schemas. Each Application Module can contain a number of collections and related queries in the form of *entities* (which are an abstraction of database tables) and *view-objects* (an abstraction of SQL queries).

*Entities* in ADF BC can be created directly from database tables and can have validation and authorization rules attached to the entity as whole or to a specific property. These rules are enforced whenever the *entity* is used; regardless of if it is use in the view layer or business layer. Foreign keys are modeled as associations between *entities* and can provide help when creating *view-objects* for retrieving related data. All operations to add, update or delete data in the database are performed directly against *entities* in contrast to retrieving data from a database, which requires the use of a *view object*.

*View objects* represent SQL views and provide a mean to retrieve data from the database. They can be based on static lists, SQL queries, entities or collections created programmatically in Java. To create an updateable view it must be based upon *entities*. Basing it on entities has the additional advantage that naming and business rules specified for the *entities* come into effect for the *view-object*.

### 3.1.3 ADF View layer

**The ADF data model (ADFm)**

Oracle ADFm provides a binding API to access the business layer form the view and controller independently on the underlying technique used to implement the business layer. Mapping to ADF BC is performed automatically by JDeveloper, but for other business layers such as EJB or web services the developer must setup the mapping manually. ADFm provides an API to iterate over views as well as to call functions in the business layer.

Each ADF BC Application Modules is mapped to a *Data Control* which is a high level object that exposes collections and functions. All views added to an Application Module of an ADF BC project are exposed as *collections* that can be displayed on pages by using drag and drop when designing ADF Faces pages within JDeveloper. When dropping a *collection* on a page for the first time JDeveloper will create a page-definition file, which is an xml file describing the data bindings, for that page. ADF will then create an *Iterator*
that provides the functionality to interact with that collection. ADFm *Iterators* provide functionality to navigate and fetch the rows in the collections with functions such as Next, Previous, First, Last and functionality to go to a certain row based on its key. ADFm Iterators also provide additional functions for creating and deleting data. These functions can be either added to pages as buttons or to task-flows (see Figure 6) as intermediate states. The Data Control generated for the ADF application and its default generated operations can be found in Figure 30.

**ADF Taskflows - the Controller (ADFc)**

JDeveloper provides a visual approach to describe application navigation and some of its functionality using simple arrows and boxes diagrams to create ADF task flows. ADF task flows are similar to state machines with named state transitions. The states can be web pages, method calls or calls to other task flow, allowing nesting of tasks flows. Whenever a button or link is clicked by the user it can either generate an action directly or it can call a Java method which in turn returns a string that defines the action to perform. Using this approach a web page such as an edit screen can be used in different context without changes to the code as long as it is coded with care.

The task flow in Figure 6 will start by showing the *chooseSelection* pages as it is marked as the starting page, visualized by the green background. From this page the user can perform the commit and the open actions. If the user presses the button for saving changes the “commit” action will be issued to the controller resulting in the commit method, displayed at the top of the figure being executed. If the user instead presses a button with the action set to “open” the controller will go to the *ExecuteWithParams* state that will call the *ExecuteWithParams* function, which executes a parameterized query on a view objects. Once the method is executed the controller will move to the *viewSelection* state due to the default action (with the same name as the method) going
out from \textit{ExecuteWithParams}. The \textit{viewSelection} state will in turn present the user with a page making the controller wait for the user to perform the next action.

\textbf{ADF Faces Rich Client the View (ADFv)}

ADF Faces Rich Client (from here on just ADF Faces) consists of over 150 JavaServer Faces components\cite{ADFv} based on top of JSF. Some of the components are graphical controls while other are included with the purpose of converting values, validating inputs, reacting to user input, implementing drag and drop functionality etc.

ADF Faces views are built using xml where controls and simple behaviors are added via xml-tags just as in "normal" JavaServer Faces. The xml-tags can be added by selecting the corresponding tags from JDevelopers control toolbar and either dragging them to the visual designer or to just select them to have the control added inside the currently selected control. The controls are configured by setting their public properties such as the orientation property which is set to "vertical" for the panel splitter control in Code example 3. The general idea is that the developer should not need to know xml, JSF and ADF controls by heart instead much of the development can be performed using JDeveloper's visual designer.

The components available in ADF Faces sometimes overlap in functionality with standard JSF components. By using the ADF versions of the tags the components can hopefully be upgraded between releases to use newer technologies without requiring code changes\cite{code_changes} with the additional advantage that the ADF versions offer a larger set of properties that can be configured\cite{ADFv}.

\begin{verbatim}
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<%@ page contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://Java.sun.com/jsf/core" prefix="f"%>
<f:view>
  <af:document id="d1">
    <af:panelSplitter id="ps1" splitterPosition="204" orientation="vertical">
      <f:facet name="first">
        <af:outputText value="Some text " id="ot1"/>
      </f:facet>
      <f:facet name="second">
        <af:commandButton text="A button" id="cb2"/>
      </f:facet>
    </af:panelSplitter>
  </af:document>
</f:view>
\end{verbatim}

\textbf{Code example 3} An ADF Faces page displaying the text "Some text" and a button.

Just as standard JSF pages, pages with ADF Faces components are stored on a web server (sometimes called application server) and get executed whenever a user access them using a web browser. Being a web technology the execution of application logic is limited to browser requests. During each browser request a number of execution steps
take place such as recreating an in-memory representation of the components on the page, validating input, updating data, executing methods and finally rendering an HTML response to the client. ADF Faces RC components generate JavaScript and uses AJAX functionality to send requests behind the curtains and to enable parts of pages to be updated without the user having to reload the whole page (a feature called partial page rendering).

Adding logic and behaviors to a page it is performed differently depending on where the logic should be run and what the purpose it has. In short Java, EL, xml, and Groovy is used to add server side logic while JavaScript, JSF components and CSS is used to add behavior at the client (in the web browser).

Programming for the server
Adding logic to be executed on the web server is a three step process. First Java is used to code a Java class that adheres to some required conventions. The class must then be defined as a managed bean which can be done in the projects ADF configuration file named adf-config.xml, directly via text or visually via its properties pages. This configures the server’s runtime to create instances of the class, which can later be called from the page. EL code is used to access properties and methods of managed beans. This process can be somewhat simplified by using a wizard to attach a method to a button.

Control properties can contain embedded EL statement resulting in the EL statements being executed when the property is read, as when determining what text to show in a textbox, if a control should be visible or what action to perform when a button is clicked. EL also allows access to data bindings (which are exposed as JavaBeans by the ADF binding layer and allow both Java and Groovy code in the ADF BC model to be called). Some simple logic such as changing label on a button depending on if the user is logged or not can be performed using EL “alone” (by accessing the Java bean containing access to login status).

Programming for the client
ADF Faces allows JavaScript code to be embedded in the returned pages in order to add behavior on the client. This allows the page to respond to user actions such as hovering over an item, entering characters into an item field without requiring accessing the server. The scripts can contain functions that should be executed under certain conditions or just code to be executed when the page has been loaded. The scripts have access to ADF Faces JavaScript API that allows interactions with some of the ADF controls (as long as a client side class generation is enabled for the control).

The developer also has the possibility to attach CSS code to the web page, allowing the visual appearance to be modified as well as some simple visual behaviors to be defined.
3.1.4 Validation
Validation rules can be added for entity properties or for entities as a whole. JDeveloper will automatically create rules corresponding to database constraints for string length, NULL values and number precision. More rules can be added by choosing from a number of predefined rules or by writing validation rules in Groovy or Java. The built in rules allow simple comparisons, regular expressions or list membership (check if a value belong to a list of values or not) to be defined visually without writing code.

All rules are executed solely at the server, with the exception of two rules that are automatically propagated to the client. The rules that are propagated to the client are the maximum string length property (which is used to set the sizes of text boxes as well as the number of characters that can be entered) and the required property (which result in a “*” next to the name of the property and JavaScript code which check that the property is not left empty).

ADF’s validation rules are validated when a post-back occurs, which usually happen when data is saved or the user navigates away from the data. It is possible to manually force data to be submitted for validation when the user exit some controls by setting their autoSubmit-property to true. It is possible to save a recommended “autoSubmit” value for each property of an ADF BC entity, but these values are not automatically propagated to the UI making it less useful since manual modifications are required everywhere the behavior is desired.

3.1.5 Authentication and Authorization
JDeveloper allows a visual approach to configuring role based access control of ADF applications. ADF’s security model is centered on entities types, task-flows and web pages as subjects for which permissions can be specified. To view a specific page the user should belong to a role which either have permission to view the page directly (which only is configurable for pages which show data and therefore have a so called page definition file) or permission to view the task flow that is hosting the page.

ADF security allows assigning permissions allow different roles to read, delete and update (which can be configured down to property level) each entity type. The access permissions for entities are static in the sense that they do not allow per instance (row) rules to be configured. Instead they apply to all instances of a specific type. More detailed rules must be configured in the database or using entity validation.

3.1.6 Internationalization
The ADF approach to store locale specific strings uses a type of files called resource bundles, which are plain text files where each line follow the format “key = value”. JDeveloper does not provide any special interface for editing these files, but allows them to be edited as plain text files. JDeveloper does however store some strings in resources bundles automatically when working with ADF BC projects. Settings such as
display name, description and formatting for entities will automatically be stored in a resource bundle when the default values are changed. The same is true when writing validation rules, where the error message will be extracted to a resource bundle.

The resources bundles can be retrieved programmatically in Java code by referring to its name. Once retrieved the resource bundle is used as a dictionary, looking up values given their key as a string. This approach provides no help from the type system to prevent errors such as misspelled keys. No special tooling support is provided for working with resource bundles in Java code. Both adding strings to the resource bundles and referring to them is performed manually. Code example 13 (page 76) shows how a resource can be referenced from Java.

ADF Faces will automatically use the language and formatting of the browser (if it is supported by the application). It is also possible to setup the locale settings in an xml file where EL expressions can be used for setting the locale for formatting, decimal separator etc. in order to allow these settings to be configured independent of the language used. Numbers and dates will be automatically formatted according to selected locale as long as no custom formatting has been selected. Proper formatting of dates and numbers that do not use the default formatting is handled by storing the format strings in resource bundles. The format strings can then be translated just as any other resource string.
3.2 .NET Framework: Silverlight and WCF RIA Services

3.2.1 Overview
The .NET framework can roughly be said to be Microsoft’s object oriented development framework designed for writing windows programs. From a simplified point of view it can be compared to the Java and Java EE API. To access the .NET Framework a program must be compiled to CIL (Common Intermediate Language) which can be described as an object oriented assembly language, which apart from technical differences is similar to how Java programs are compiled to Java byte-code. The CIL is compiled to machine code by the Common Language Runtime (CLR) before it is run. One of the larger differences with Java is that the .NET platform was designed for being used by different programming languages. Support for C++, C#, F# and Visual Basic are shipped with Visual Studio and many other languages (including Ruby, Python, COBOL and Scheme) have compilers that can target the .NET platform.

The current version of the .NET platform (as of May 2011) is version 4. It is available from Microsoft for all Windows operating systems from Windows XP and Windows Server 2003 and newer. Alternative implementations exist of which the open-source implementation mono (16) is the most popular with versions for Linux, Mac, Windows as well as support for Android, iPhone and iPad.

For this thesis the combination Silverlight and WCF RIA Services with the Entity Framework was chosen. The high level architecture is a 3-tier design as given by Figure 7 where the first tier is the client written using Silverlight as presentation technique. The client communicates with a web service where the service is based on WCF RIA Services. The service in turn accesses a SQL database using Entity Framework as OR/M tool. The programming language used in the thesis is C# as it was developed specifically for the .NET platform. C# is present in most documentation for the platform and it is similar to C++ and Java which the author has previous experience with.
3.2.2 Silverlight – The Presentation Layer

Silverlight is a cross-platform\(^1\) and cross-browser programming environment supporting a subset of the .NET Framework targeting RIA and media experiences\(^{17}\). Silverlight applications have traditionally been hosted in web browsers through the use of browser plug-ins. Silverlight is no longer a browser only experience, but applications can be downloaded and installed locally since version 4. This allows an end-user experience similar to that of traditional desktop applications (without the browser UI) as well as integration with local applications such as Outlook and improved startup time. In fall 2010 Microsoft released the Windows Phone 7 operating system for mobile phones with Silverlight as the platform used for creating mobile applications.

The graphical interface in Silverlight is built by adding controls, vector graphics, animations and some behavior using an xml based language called XAML. XAML and most of the controls are the same that is used when developing for WPF, Silverlight’s desktop-only counterpart, making it possible to use the same pages in both technologies. The code for a simple page is found below as Code example 4. This code will look like Figure 32 when displayed in the browser which is almost identical to how it looks like inside the visual studio designer (Figure 31).

```
<navigation:Page x:Class="LSOnline.Views.Page1" <!--Class for code-behind -->
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:nav="clr-namespace:System.Windows.Controls"
    Title="Page1 Page">
    <Grid>
    <!-- Create two equally sized rows -->
    <RowDefinition Height="*" /><!-- "*" ⇔ stretch -->
    <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <TextBox Text="Some Text" Grid.Row="0" />
    <Button Content="A button" Grid.Row="1" />
    </navigation:Page>
```

Code example 4 XAML for a Silverlight page displaying some text and a button

Out of the box 45+\(^2\) controls are available for Silverlight developers with 18+ more available as part of Microsoft’s open source Silverlight toolkit at codeplex\(^3\). Compared to ADF Faces this may sound like very few but while the Silverlight controls adds visual elements many ADF components describe non visual parts of a web page such as what format to display use in textboxes

---

\(^1\) While Microsoft only provides implementations of Silverlight for Mac, Windows, some Symbian phones and Windows phone 7 the mono project provides implementation for Linux, BSD and some other operating systems,

\(^2\) 60+ components according to (33), but it is unclear if these include the Silverlight toolkit controls since only 45 controls were found at “Controls by Function”, MSDN, [http://msdn.microsoft.com/en-us/library/cc645075(v=vs.95).aspx] 12 May 2011.

\(^3\) Codeplex ([http://silverlight.codeplex.com]) is a web page hosting open source projects.
The UI can be created by writing XAML directly, by visually creating the interface via dragging and dropping controls or by a combination of both inside Visual Studio. Microsoft also provides a separate program called Expression Blend for UI designers that focus on creating visual appearance including animations as well as navigation.

Each Silverlight page has an associated class, called its code behind class, defined by the “x:Class” property (see Code example 4). This class is the traditional location to add code for responding for button clicks and other events from the graphical interface. By double-clicking a button in the Visual Studio designer a new method is generated in the code behind class and the developer is navigated there. If an existing method has already been defined for the button, the user is navigated to it without creating a new method. Instead of coding in the code behind file alternative approaches such as the MVVM design pattern can be used to create code with lower coupling between UI and application logic. Independent of the method used all logic in a Silverlight application is run at the client and the coding is performed using any .NET languages. Thus knowledge of a single programming language and some understanding of XAML are required to create Silverlight applications.

Once compiled a Silverlight application is packaged into one or more xap files which contains the compiled code as well as other assets such as images. This file is downloaded and cached on the user’s machine when a Silverlight application is accessed with the browser.

3.2.3 Entity Framework – The Data Access framework

The Entity Framework (EF for short) is an O/RM framework that is part of the .NET Framework. The mapping between database and model (from which entities, classes, are generated) can be defined visually via Visual Studio’s built in designer, by directly creating xml files describing the mapping or by placing attributes on classes. When using the built in designer the model and mapping can be automatically generated from an existing database using a wizard to select tables, views and/or procedures to use as data sources. Queries can be performed against this model using Entity SQL which is very similar to SQL but with some enhancement such as using associations in the models instead of joins. Entity SQL queries are translated into SQL queries at runtime using the mapping.

When storing the mapping in xml files (the approach used by the wizard and declarative designer) the mapping is loaded at runtime. The runtime loading allows changes in mapping and switching of database provider to be done without recompiling the application.

The entity framework abstraction of database rows is called entities. They are .NET classes that among other things support change tracking. The abstractions of databases are object contexts which can contain any number of entities exposed in the form of
object sets (corresponds to database tables). The exposed object sets can be used to interact with entities in the database via operations for inserting, removing and iterating the entities. Changes to data including inserts and deletes are tracked by the object context allowing the changes to be submitted all at once.

There exists no abstraction for views, although database views these can be mapped to entities. This might at first seem strange to database oriented people but the reason for this is the integration of LINQ, a feature that distinguishes EF from many other O/RM frameworks. LINQ allows many different data sources (such as data in the database) to be queried in a type safe manner directly in code using SQL like syntax.

LINQ works by a set of methods such as Where, OrderBy and others that provide support for allowing typed queries containing filtering, projections, transformations, aggregations, sorting, grouping, joining of collections etc. When writing a LINQ query an expression tree representing the operations performed is generated. These expression trees can be passed around, combined and used as an abstraction of database views if desired. The query itself is not executed until the results are iterated, at which time it is translated into Entity-SQL and finally SQL (if the LINQ query were performed against Entity Framework entities). When using WCF RIA Services the expression trees created at the client are sent over the network to the domain service where they are merged with any server side expression before they are finally executed.

Code example 6 and Code example 7 shows C# build in syntax for using LINQ (notice the similarity to SQL queries) while Code example 5 show the “raw” code which Code example 6 will be translated to.

```
Return ObjectContext.Selections
  .Where(selection => selection.Owner == this.CurrentUser)
  .OrderBy(selection => selection.ID);
```

Code example 5 Alternative syntax for LINQ statement in Code example 7

For people new to C# the syntax in Code example 5 needs some explanation. The code “selection => selection.ID” generates a lambda function (unnamed function) which takes a selection (left side) as input argument and returns its ID (right side). By passing this function to OrderBy means that the id field will be used to order (sort) the results just as if one writes “order by selection.id” in a SQL query. The developer do not need to know all these details, it is enough to think that the function is applied to all elements that have passed the condition specified in the where clause.

3.2.4 WCF RIA Services – Hosting the business layer

WCF RIA Services (RIA Services or WCF RIA for short) is a framework built on top of WCF, .NETs client-server framework, and provides a convenient way to expose data from a server to Silverlight clients. RIA Services leverage the fact that both server and client run on the .NET platform. It allows code such as validation logic to be shared
between the application and presentation tiers as well as to automatically generate client side code for accessing the service from code or XAML. Visual Studio’s graphical designer allows data to be displayed in the form of tables, forms or individual properties by using drag and drop within the designer. Another feature of RIA services is that data exposed by the service can be queried by the client using XAML or LINQ (Chapter 3.2.3) with the result that the queries are sent the service where they are processed and possibly forwarded to a database in order to limit the amount of data sent over the network.

![Figure 8 Overview of WCF RIA Services](image)

Services built around WCF RIA Services are based around DomainServices, which are classes that expose collections of typed data as well as methods to client applications. The classes exposed can be decorated with metadata attributes to specify what properties should be available on the client. This makes it possible for properties such as passwords, permission data, creation date etc. to only exist and be accessible on the service.

The DomainServices can be seen as abstractions of databases (or database schemas) as they expose collections and methods. The collections exposed are the abstractions of tables and views. For each type of data returned from a DomainService four different types of special functions can be added, namely queries, updates, inserts and deletes. These functions can either be written using specific prefixes in their names (Get, Update, Insert or Delete) or with the help of attributes ([Query], [Update], [Delete] or [Insert]) attached to the function. See Code example 6 for an example of a query function where both the conventions with specific function naming (Get*) and an attributes ([Query]) have been used.

Apart from the default update function multiple named update functions (which can accept parameters) can be attached to a class. They allow the client to call complex logic that is later executed on the server. Named update functions can be used to allow resetting the password of a user despite that passwords are never exposed to the client.
When a WCF RIA application is compiled a “Domain Context” is generated for each domain service with the purpose of managing the communication with the domain service. The domain contexts expose queries and methods of the domain services except for update, delete and insert functions which are handled in a special way. Queries exposed by the domain service can be further filtered, sorted and paged (fetching a limited number of items such as items 1-10 or 11-20) using LINQ or XAML on the Silverlight client.

Code example 7 shows how the collection of selections exposed by the GetSelections function in Code example 6 can be filtered to include only selections with the property “TOMTRANSDATUM” ending with “11” (meaning the year xx11) at the client. The filtering request is sent to the service which in turn can translate it into SQL.

For each collection of a domain service the corresponding domain context will have an entity set generated. The entity set can be used to add or remove items from the service’s collection. To remove an item from the Selections table in Code example 6 one would call the “Remove” function of the domain contexts “Selections” entity set and pass the item as argument. Data fetched from a domain service is automatically tracked by the domain context. When a save call is performed on the domain context all entities that have been modified, inserted or deleted will result in a call to the corresponding update, insert and delete functions.

For “named” update functions methods are generated and attached to the class which they update. Calls to these methods are recorded and sent to be executed on the service together with any other changes when the client decides to commit changes.
See Code example 17 and Code example 16 in the Appendix for a definition of a named update function and code calling it.

### 3.2.5 Validation

Validation can be implemented by adding annotations to whole data entities or directly on properties. Validation is automatically checked at the client (to provide graphical response) as well as on the service (to provide integrity).

WCF RIA Services will automatically use validation attributes configured by the entity framework (which just as the other alternatives create rules for database constraints such as string length, NULL values and number precision). In addition to these rules some additional predefined attributes can be used for validation. These additional attributes provide support for declaratively adding validation based on regular expressions (useful for e-mail address etc.), allowed ranges of values and checking that a value belongs to set of valid values. If the built in annotations do not suffice more specific attributes can be created or the out of the box “custom validation” annotation can be used to call any C# method.

Microsoft also provides the open source Enterprise Library which among other things provides 18 additional validators and includes support for configuring validation rules using XAML based files (with support for using a graphical interface). (18)

The validation rules are propagated to the client allowing them to respond directly to data changes without any server communication providing immediate user feedback. It is possible to have different implementation of the validation rules on the client and server. This can be utilized to execute data intensive validation on the server which can be executed either asynchronously on data modifications or in a deferred fashion when the data is submitted.

### 3.2.6 Authentication and authorization

Authorization in WCF RIA Services is performed declaratively by annotating code, just as with validation. In the case of authorization the annotations are applied to methods of domain services. In order to limit who make modifications to an item the update, insert and delete functions of that item can be marked with authorization annotations such as the predefined RequiresRole annotation. Custom authorization annotations can easily be created and used if the built in support for role based access control does not cover all required scenarios. Code showing the usage of the RequiresRole attribute (Code example 17) as well as the implementation of a custom authorization attribute (Code example 19) can be found in Appendix C.

### 3.2.7 Internationalization

For Silverlight there exist two different types of files that can be used to store strings that are to be translated. The first approach store strings in the same type of XML based resource file, called “resx”, used by other .NET technologies. Visual Studio has built in
support for visually editing these files (Figure 9), which can contain other types of data such as images and icons. The runtime will look at the language specified for the current thread of execution when a resource string is accessed and return the string stored for that specific language.

The resource file generates a class with a property for each entry in the resource file when the application is compiled. This allows type-checked access to the resources as well as auto-complete functionality in Visual Studio. The strings can also be accessed from XAML by binding to properties of the generated resource class. In the project template the resource classes are accessed from XAML by going via an extra application-resources class that is accessible from all pages

The other approach is to use XAML based resource dictionaries. Resource dictionaries provide a general location for storing shared data including control look and feel. As such they can be used to store key-value pairs for a number of different data types including strings. The values can be accessed from XAML by referring to the key using syntax very similar to that used for data-bindings. The drawback with this way of internationalizing the application is that resource dictionaries are built into the application, with no support for switching at runtime. This means that the application must be compiled once for each targeted language.

Figure 9 Visual Studio’s resource editor

---

1 Ideally one would like to bind directly to the resource classes (which is done in WPF projects), but Silverlight 4 do not allow static (global) objects to be used as the target of a data binding sources. That is why an extra “wrapper” class with properties returning the static objects is required. This will no longer be a problem in Silverlight 5 which allows other types of “bindings” to be coded.
For entities and other data an annotation based approach to multilingual application can be used to specify strings centrally. By adding Display attributes to data properties it is possible to set their display name, description as well as some other properties, with support to fetch them from resource files. Silverlight controls such as the data-grid, data-form and labels will automatically detect and use these annotations.
3.3 Lightswitch

3.3.1 Overview
Microsoft Visual Studio Lightswitch or Lightswitch for short is a new addition to the Visual Studio product family. It is a developer environment optimized for so called data over forms (forms over data) also called CRUD (Create Read Update Delete) applications where most interactions from the user are intended to either view (read), add (create) or modify (update and/or delete) data. The general idea with Lightswitch is that the developer should be able to focus on the business logic and have the tasks such as database access, data export and UI creation automated as much as possible.

![Lightswitch architecture](image)

Lightswitch provides a high level approach to application development where a number of default screens (pages) types such as new, edit, search, and details screens are supported out of the box. UI design changes are performed using a visual-only approach where data can be dropped on the screen.

The compilation of a Lightswitch application is a two phase process where the first phase generates code for either a 2-tier or a 3-tier (Figure 10) application that is built using frameworks such as Silverlight, WCF RIA Services and Entity Framework described in Chapter 3.2. Once the code has been generated it is compiled in a second phase creating the final executable. This two phase compilation is almost invisible to the developer with the generated code hidden inside of Visual Studio. This allows Lightswitch applications to change the underlying platforms without affecting the programs themselves. Lightswitch currently supports generating a 2-tier application where presentation and logic are both deployed inside the client or a 3-tier application where logic and storage can be deployed either locally, to a supported web-server or to Windows Azure (see Figure 33, Appendix B).
3.3.2 Data Access

Data accesses in Lightswitch go through objects called data entities, which correspond to database tables (and views). Properties of the data entities are mapped to database columns by using types corresponding to standard database type as well as three more specialized types, e-mail, phone number and Money. These more special types are built in custom business types (more can be added) which control formatting as well as validation while still storing the value as a string or another data-type behind the scenes. The phone number type allows multiple allowed phone number formats to be specified (for example with or without country code) and it will automatically format the data according to the best matching format while typing.

Data entities created from inside the Lightswitch designer are added to a default data source, called ApplicationData, which is an “internal” database where database tables are created for the data entities and their associations running the application. When deploying the application the “internal” database is uploaded to a Microsoft SQL Server (either to a free express variant or any of the paid versions).

Lightswitch can as of Beta 1 get data from three different types of external data sources: .NET compliant SQL databases, SharePoint lists or WCF RIA Services (Chapter 3.2.3). Support for connecting to Access databases is planned for the final release (2). Lightswitch data entities can be created for tables, views or lists inside these sources, but no new tables or columns (apart from computed columns) can be added to these external data sources from within the Lightswitch designer. This is not a very large problem since if you should be allowed to add new tables or edit existing in one of these sources you will probably have other tools for doing that.

When a data entity has been defined some queries are automatically created for it. Two of them allow a single row to be selected based on the entities primary key. Additional queries containing filtering, sorting and grouping can be created for the data entity with the help of a query designer. Apart from adding data visually in the designer data can also be accessed programmatically via the data warehouse, an object accessible from all screens that expose all data sources with their data entities and queries. When working with the data warehouse LINQ can be used to query the data sources.

Validation rules can be added for whole entities or for validating specific properties upon change. When the validation fails the corresponding control is marked red and the user is presented with an error message.

Authorization of changes is performed in the same way as validation with specific functions that can be implemented to check if insert, update, delete or reads are allowed by the current user.

---

1 They differ in how they handle the case with no matching item
3.3.3 UI

A Lightswitch application’s graphical interface is made up of two main components, a shell and several views. The shell defines the applications look and feel and decides where to locate the main menu as well as where and how views are displayed once opened. The shell shipped with the Lightswitch beta shows the main toolbar using the ribbon look and feel at the top, just as in Microsoft Office. The standard shell displays the main navigation menu to the left as seen in Figure 12 and the rest of the area is used as workspace where screens can be opened in a tabbed fashion, just as in today’s web browsers.

![Figure 12 Lightswitch standard shell with 3 screens opened](image)

The Lightswitch UI is based on Silverlight but the designer is fundamentally different from both the standard Silverlight designer as well as JDevelopers designer for ADF Faces. The first difference is that when creating a new screen a list of templates (new-item, edit-item, list and details (the type opened in Figure 12), search-item and editable grid) are available. When selecting a template the screen is automatically generated for the selected data entity based on the template, allowing screens to be created with no or little manual intervention. Screens such as new, edit and details have special meaning for data entities and when the user clicks a button to add or edit a data entity that entity’s default screen for the specific action (or an automatically generated one) will be presented to the user.

The second difference is that the Lightswitch designer is used to modify and interact with the control hierarchy of the screen (see Figure 13 for an example of the hierarchy). The screen layout is based on “groups” that can be used to divide the screen area into rows, columns and tabs. Each group controls the layout inside it allowing the content to be organized into rows or columns. The developer can select a group to add buttons,
data or other groups inside it. Inside the Lightswitch designer there is no preview of the final appearance nor is there a list of available controls to drag and drop from as in the other designers. Instead controls are added or modified directly in the control hierarchy.

The lack of visual preview can be a bit disturbing to developers used to work against one, but this is more than compensated for by one of Lightswitch unique features; the runtime designer. When running a Lightswitch application in debug mode the user can choose to edit an open screen from within the application. The user is then presented with the same hierarchal component tree as in the designer as well as a visual representation of how the screen will look. Once edited the screen changes can either be saved or canceled, all without having to recompile anything or leave the debugger.

From inside the screen designer the developer can choose to display data from one or more data sources (queries or entities) as well as to add local parameters by selecting “Add Data Item ..” from the toolbar. Available data is displayed to the left of the screen and can be added to the screen by using drag and drop. Tables and forms displaying all properties of a data entity can also be created by clicking on a node in the visual tree and selecting the appropriate option from a popup menu.

For each type of data there exists a default control, such as the label used to display read only strings on the screen and the textbox which is used for editable strings. For some data-types such as dates there are more options and the control used can be changed by clicking the small arrow next to the control’s icon in the designer.
3.3.4 Validation

Validation in Lightswitch can be configured by adding validation rules to screens, data entities or their properties. The validation methods are written using C# or VB.NET and from within these functions can errors, warnings and informational messages be added to the entity being validated or any of its properties. These messages will then be attached to either the control displaying a property (by a red border and a message) or to the whole screen so that the user can view them.

Validation for entity properties is run on the client, allowing any validation problems to be immediately visible to the user. When saving data and all properties are valid on the client the changes are sent to the data access layer. The data access layer will run the relevant entity level validation functions and only save the result if there are no errors.

Lightswitch will synchronize database constraints such as NOT NULL, string length and number precision between data entities and the database. When creating entities within Lightswitch it is possible to define additional validation properties such as minimum and maximum values. More advanced validation can be implemented by implementing any of the special validation functions that can be written for entity properties (one validate function per property) or whole entities (one per entity type).

Lightswitch comes with a number of business/domain data types such as phone-number and e-mail address that implement more advanced validation and formatting while still storing the data using normal database types. Additional such business data types can be created to allow specific validation logic to be centrally defined.

3.3.5 Authentication and authorization

A distinguishing feature of Lightswitch is that users logged in as application administrator get access to automatically generated administration screens once security has been enabled. These screens allow the administrator to create and edit users and roles. A user can belong to any number of roles and a role can be assigned any number of permissions.

Permissions are created within the designer and are made up of a permission name (for accessing it in code) a display name (shown in the administration screens) and a description (also available in the administration screens). The permissions can be checked in code by referring to its permission name using code such as “User.HasPermission( Permissions.CanViewCustomers )”. The creation and use of a global variable for each permission provides both auto-complete functionality and compile time checks.

The permissions are general by design and do not automatically apply to entities or navigation. Instead there are a number of methods for screens (CanRun which also hides screen from the navigation menu), queries (CanExecute), entities and entity properties (IsReadOnly) that are intended to be implemented in order to perform
access control. The application’s permissions are intended to be used when implementing these functions, but that is not a requirement. For entities this allows updates (CanUpdate), inserts (CanInsert), deletes (CanDelete) and reads (CanRead) to be configured for each entity type. There is also an additional function SaveChanges_CanExecute that can be used to look at the specific changes that have been performed in order to implement per instance access control.

3.3.6 Internationalization
Lightswitch does not contain built in support for building application with multiple supported UI languages. It is currently limited to use a single language. The language is configured at the application level and affects formatting as well as built in assets such as error messages. 41 different folders with localizations for different regions, including a Swedish translation, are shipped with Lightswitch beta 2.
4 Evaluation scenarios
This chapter describes the features and target scenarios that were later implemented in order to evaluate the development platforms. The scenarios were chosen with the help of an expert on the current system so that they would reflect likely real world scenarios.

Each of the implementation scenarios was specified in detail before starting the implementation in order to prevent “feature creep” (adding of additional requirements during the implementation) and to make sure that the same features were implemented for all frameworks.

4.1 The basic application
A simple application was developed in each of the development environments in order to see how easy it is to get started with the different environments, the quality of their documentation and how easy it is to get help for problems.

Before implementing the basic application a sketch/prototype was created as part of defining the requirements for the applications graphical interface and its navigation. This chapter will start by introducing the different screens and the navigation for the target system. This is followed by a description of design of the database, which the applications will access. The UI sketches for the different screens presented can be found in Appendix A.

4.1.1 The graphical interface
When starting the application the user will come to a start screen (Figure 24) where the user has the option to either go to the choose selection screen or to the settings screen.

Choose selection screen
This screen presents the user with a list of all selections in the database (Figure 25). When choosing a selection in the list some details about the selection are presented to the right. When the user is satisfied with the chosen selection he can continue to the transaction list screen.

Transaction list screen
The transaction list screen (Figure 26) displays a data-grid where the transactions in the selection are presented. The user should be able to reorder columns as well as sort displayed data based on any column.

Settings screen
On the settings screen (Figure 27) the user is presented with a number of drop-down boxes where different settings can be configured. All values are static (hard-coded) except the time-zone control, which is bound to a table in the database where all available time-zones are stored.
**Language**
The user can choose between “English” and “Swedish”, the value will later change the presentation of the application.

**Predefined Locale**
The user can choose Locale between “English (UK)”, “English (US)”, “Swedish”, “Finnish” and “Custom” each of the settings will set number format and locale to predefined settings. When setting it to custom, number format and time-zone can be changed independently.

**Number format**
The user can choose between “10,000.10” and “10 000,10” this setting was intended to affect formatting of numbers on the transaction list screen later on.

**Timezone**
The user can choose between timezone region names stored in the database, ex “Europe/Stockholm”. This setting should later in the project affect time displayed so all time is displayed in the local timezone.

**Navigation outline**
When starting the application the user is presented with the start screen. Figure 14 below visualize the main navigation scenarios where the user can then go either to the settings screen or decide to view transactions, by first going to the choose selection screen.

![Figure 14 Navigation flow of the basic applications](image)

The user can at any point in time go directly to the start screen or the settings screen by using any of the global buttons at the top of the application as can be seen in Figure 24.

**4.1.2 The Database**
A database scheme with populated tables was created for the sample application. The tables and some of their used properties can be seen in Figure 15.

**The Settings table** allows the application to store the selected settings at the settings page. It contains the UI Language; the locale used for formatting; a string that can store information about custom number formatting and the selected timezone that is a reference the timezone table setup with foreign keys.
The Timezone table contains a list of all supported timezone regions by name such as ‘Europe/Stockholm’

The Transaction table represents a table used to store financial transactions. A table with many more columns was used in practice, but properties that were not fetched from the database are excluded.

The Selection table contains search criteria for selecting transaction from the transaction table. For example the FOMxxx, and TOMxxx columns gives upper and lower bounds for some of the transactions properties. Some identification information such as the owner (creator) of the selection and a name (PORTFOELJ) is also stored.

The selection table was created using a subset of a view where transactions and selections counterparts from a real world test setup were joined. To reflect the fact that the table should represent a view used for querying no foreign keys were added between the Selection and the Transaction tables. Instead transactions were to be fetched by a SQL query like “SELECT * FROM Transaction WHERE LISTPARAM_ID = X” where X is the ID of the Selection to be shown.

As with the Transaction table many additional properties were stored in the database, but these are the ones used and fetched by the different prototype implementations.

4.1.3 Measurements

The time required for the implementation were recorded, including time spent debugging and searching for help. The time measured were divided into the time reading online documentation before starting to code, the time used to implement the desired navigation and look and feel and the time required for setting up data mappings and displaying data on the applications different “pages”.

In addition the number of search results when searching for the different technologies (using Google as search engine) was recorded in order to get a course estimate of their popularity.
4.2 Deciding which techniques to continue with
Before starting to implement more advanced functionality the different alternatives were to be reevaluated and the alternatives were to be reduced down to two. The alternatives chosen to continue with were Oracle ADF and Silverlight + WCF RIA Services. Refer to Chapter 5.3 for the rationale for this.

4.3 Authentication and authorization
In order to evaluate the support for role based authorization the applications were configured to enable authentication before roles and users were created. The applications were configured to enable users to login using a form based approach, meaning that the user would be able to press a login button and be presented with a request for password and username. Four different users were added to the application MrAdmin, BoUser, Simple and NonPriv. They were assigned different roles according to Table 1. The time required for all this was recorded and the result is presented in Figure 22.

<table>
<thead>
<tr>
<th>User \ Roles</th>
<th>Admins</th>
<th>Backoffice</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>MrAdmin</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>BoUser</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Simple</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>NonPriv</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4.3.1 Authorization scenarios
In order the get an overview of how authorization works in the different development environments as well as to see how they handle more complex authorization rules a number of different requirement, which were thought to represent the most common scenarios, where implemented. The time required for all this was recorded and the result is presented in Figure 22.

Restricting user permissions
Each of the groups should be assigned the permissions according to Table 2. If a user belongs to multiple roles, he should be allowed to perform tasks allowed by any of the roles.

<table>
<thead>
<tr>
<th>Action \ Roles</th>
<th>Admins</th>
<th>Backoffice</th>
<th>User</th>
<th>Anonymous</th>
</tr>
</thead>
<tbody>
<tr>
<td>View screens</td>
<td>-</td>
<td>-</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Create/Delete Selections</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Edit selections</td>
<td>All visible</td>
<td>-</td>
<td>Owned</td>
<td>-</td>
</tr>
<tr>
<td>Edit transactions</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Confirm transactions</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Limit visible rows based on user
In addition to the permission in the table, access to the Selection table was limited to only those Selections where the selections Owner property equals the user’s name or where the Owner property had been assigned to a special purpose account.

Allow specific updates: The confirm function
Although not all users should be assigned full access to edit data it might be necessary to allow them to edit data in specific ways. In this scenario Backoffice users should be able to mark transactions as confirmed changing the status property from ‘new’ to ‘confirmed’, but should not be allowed to do any other modifications. The users should be able to select multiple transactions on the transactions screen and then press a button in order to mark the entities as confirmed.

The confirm functionality itself should be implemented as part of the business layer in order to see how the frameworks support separating application logic from the presentation layer and how authorization is handled.

Hide elements for unauthorized actions
To secure an application and prevent unauthorized users from performing an action it can be enough from a security standpoint to prevent it at the data access layer. It is however not user friendly to allow the user to perform an action in the first hand just to present them with an error message telling them they are not allowed to performed the action. To evaluate the support for hiding unauthorized functionality three different buttons calling the confirm function were added to the graphical interfaces. One button would always be visible. One would be disabled if the user was not authorized to perform the action and the last one that would be hidden if the user was not authorized to perform the action.

4.4 Internationalization and localization
In this report the term localization is used in the meaning of adapting software for use in countries or regions with different languages and formatting. Examples of formatting that differ is the decimal separator for numbers, which can be a dot or a comma, and formatting of dates where both the separator between as well as order of year, month and day may differ meaning that 4/5/2011 could mean either 4 May 2011 or 5 April 2011.

It was a must have requirement for a future system to support both different UI languages as well as supporting different formatting of dates, numbers and currencies so the applications were updated to expose the available regions on the settings to allow the user to change the selected locale and language. By default the language set in the web browser should be used for to determine the language to display and the formatting to use.
Translation
The environments were first prepared for localization. Since they had not been developed with internationalization in mind they both contained a large number of hard coded strings. The first effort was to move these hard coded strings to separate files, called resource files for Silverlight and resource bundles for ADF. During this preparation any tools provided by the IDE was to be used. The resource files were copied and renamed by adding a locale specification such as ‘sv-SE’ to indicate that the copies should be used for web browsers using Swedish as language. The environments were then configured to use the new files for web browser that indicated that they accepted ‘sv-SE’ as primary language instead of ‘en-US’ (English-United States). The time was captured and is presented in Figure 23, where the time required for the translation is also present. Both these timings exclude time to verify correct behavior.

Formatting dates and numbers
The applications should use the web browsers language in order to adjust the formatting of numbers and dates unless the user enters the settings screen and manually changes the formatting. For dates 4 may 2005 should be formatted as “2005-05-04” when using Swedish formatting and as “5/4/2005” when US formatting is applied.

Adding timezone awareness
In order to allow people working in different countries to work together the application should adjust displayed times according to the timezone of the user. The timezone to use should be set according to the language settings of the browser. In order to implement this some database columns has to be changed from date types to a type storing both time and date such as Oracle’s Timestamp type.
5 Implementation

This chapter describes the implementation of the different scenarios described in Chapter 4. It starts with a description of the hardware and software used for the evaluation and then continues with by describing the implementations in the order the features were implemented. Each scenario builds on the application developed for the previous scenarios starting with the basic application.

Chapter 5.3 then describes which applications were further evaluated by implementing the authentication and authorization (Chapter 5.4) as well as the internationalization (Chapter 5.5) scenarios.

5.1 Software and Hardware

Initially the Oracle ADF version bundled with JDeveloper 11.1.1.2 was used. It was upgraded together with the IDE to version 11.1.1.4 before starting evaluation of internationalization properties (Chapter 4.4). When evaluating Visual Studio Lightswitch, the beta 1 release of the software was used. Evaluation of Silverlight and WCF RIA Services was performed using Visual Studio 2010 SP1 with the bundled versions of Silverlight (4) and WCF RIA Services (SP1).

Development and testing took place on a virtual machine with Windows XP SP3 installed. It was assigned two cores of an Intel Xeon X5450 CPU (3GHz) and 2GB of RAM. After the initial evaluation (Chapter 4.1) the amount of RAM was increased to 4GB, but it was still limited to 2.97GB due to 32bit memory limits.

5.2 The basic application

5.2.1 ADF Implementation

Introduction videos and documentation (mainly from oracles website) was studied four around 3 hours prior to the implementation. The implementation then begun with creating a shell where pages could be hosted via task flows by following the step by step guide available at (20). The result can be found in Figure 16. Most of the visual appearance came from the template used, but labels for choosing between the applications main areas was added as well as some menu buttons. No manual coding took place. Instead the example code for navigation and managing the tabbed interfaces were copied directly from the web page. The green color at the bottom was added by editing the documents “inline style” which basically allows the developer to attach CSS code to page elements. Initially the purpose was to change the blue background but there were no simple option for changing it. Some people suggested modifying the template used for creating the page which was well beyond the scope of this initial application.
Figure 16 Initial look of ADF program

After the visual appearance had been setup and some simple navigation was working the task of creating the data mapping was next. The mapping of database tables to entities was performed using a wizard where the database tables were selected.

Unfortunately the wizard failed with creating a connection to the database. After searching the internet a workaround was found. The connection settings could be entered from the project properties pages before running the wizard.

Entities were initially created for the Selection, Transaction and Timezone tables. The Settings table was added at a later time, resulting in the entity diagram in Figure 17. It was selected that all entities would get updateable view objects generated for them in the wizard. Once the entities had been created, columns that would not be used were removed in order to keep the property count down\(^1\). Even though a foreign key relation had been created from the settings table to the timezone table it was not picked up automatically, not even after when the “synchronize with database” function was run. Since this association was of no importance at the time, it was not added manually.

When implementing the navigation between the Selection view and the Transaction view the id of the selected selection must be passed between the pages. This was implemented by attaching an ADF actionListener component to a button. The component copied the id of the current selection to a task-flow “global” memory space from where it could be read when loading the transactions (the ExecuteWithParams state in Figure 6). This worked until the button was changed to a link with a

\(^1\) An alternative approach would have been to create a view exposing just the required columns and mapping directly to the view.
“convert to ...” option from inside JDeveloper. JDeveloper warned that some properties and the attached component were unsupported for links and that they would be removed. The component was not removed leading to runtime errors when showing the page. Large stack traces were presented, but with no indication of which control in the control hierarchy caused the error. This led to some frustrated debugging of a newly added “open” button for which a new actionListener had been added to implement the functionality instead.

The work then continued with some minor issues (minor problems but some took significant time) such as misspelled data bindings. Some binding errors resulted in no data being displayed while other crashed the current user’s session and printed a stack trace or displayed a message box with a short error code. Common for the majority of errors which took time to resolve was that the information reported on where and when they occurred was sparse or absent. When error information in the form of large stack traces were available the stack traces most often showed the call stacks of the ADF framework and the JSF environment, with no code belonging to the applications itself, and no information of what control or binding expression had been executed.

The time used to implement the application was somewhat increased by the fact that JDeveloper crashed around 8 times during a 2 day period with each crash requiring at least 5 minutes to get started again. While most of the time waiting was spent searching for information and documentation regarding ADF, the crashes made it somewhat difficult to stay focused at the task at hand. The reason for the most frequent crashed was found to be caused by references to a class which had been removed (and JDeveloper had validated that the class was not referenced). After removing all references manually (by searching through all the files of the project) the rate of IDE crashes went down considerably.

The time spent debugging was further increased by the fact that starting the application in debug mode took at least 2-3 minutes if the integrated web server was not started, and around a minute to deploy updates and open the application in the web browser if the web server was already started. These unwanted breaks accumulated to a nontrivial amount of time during the course of the implementation.

5.2.2 Lightswitch implementation
Before starting to create the Lightswitch application the introduction to Visual Studio Lightswitch video and the relevant “how do I” step by steps videos available online were studied (5).

With a default UI and shell already created by the project setup the development began with adding the already existing Oracle database as an (external) data source. The application was connected to the data source using oracles .NET provider (ODAC Entity Framework and LINQ Beta 11.2.0.2.30) which had been installed prior to the
implementation. Just as with ADF a wizard presented the developer with available tables and views to map against. All views and tables were presented in a hierarchical fashion giving longer loading times than ADFs wizard, which allowed tables to be searched by name instead.

Once the entities had been created most of the Selection’s properties were hidden. This took quite some time since Lightswitch beta 1 does not support batch editing of entity parameters. This might not be a problem when using Lightswitch from the start of a project as this is configured when the parameters are added, but it was annoying to perform it manually in this project.

Initially the creation of the settings screen was attempted using a control first strategy. This wasted some time before realizing that Lightswitch has almost no support for displaying static items not bound to any data. Instead the screen was created by dragging data from the settings table. To allow combo-boxes to be used for choosing language and number-format the corresponding entity properties were assigned static lists of possible values.

The choose selection screen was almost feature complete by just creating the standard list-and details screen. The only action required was to change the summary field of the selection entity so it displayed the name of the selection. Instead of adding open and close buttons, the name displayed in the list was configured as a link (by changing a property) and the shell’s tab control was used to close the screen. The results can be seen in Figure 12.

One of the data entities was renamed when the implementation was almost finished. The rename was not properly implemented in Lightswitch beta 1 and resulted in some compilation errors which had to be manually fixed.

The other major nuisance during the implementation was when trying to debug some code. Quite some time was spent trying to figure out why a debugger breakpoint was never reached only to find out that the debugger did not work properly.
5.2.3 Silverlight Implementation

Before starting the .NET / Silverlight development the documentation at (3) and (17) were consulted and Visual Studio was updated to Service Pack 1, which had recently been released. The time spent on documentation was chosen to be roughly equal to that used for reading ADF documentation. About half an hour of that time was spent looking at third party controls.

![Figure 18 Silverlight business project template](image)

Once the initial application had been created about 40 min was spent trying to setup a nice looking 3rd party menu with the same appearance as the menu used in ADF. A similar menu was created but with slightly different visual behavior. The menu was then removed, with 40-60 wasted minutes, in favor of following the style set by the project template with links at the top.

No tab control was added to the application so it had only one active screen at a time as in Figure 18. The rationale for this was that the solution should follow the style used by any tutorial or project template used. It was also of interest to see how Silverlight’s browser integrations worked for the application. Silverlight’s navigation framework integrates with the browsers’ back/forward buttons as well as the browsers’ URI text field. The integration allows navigation between Silverlight pages to work in the same way as when navigating between web pages. The integration with the browsers’ URI-field allows bookmarking of pages as well as manual navigation between them.

When pages (views) for home, choose selection and settings had been created and the navigation had been configured it was time to setup the data mapping. To setup the data mapping an entity framework model using Oracle’s .NET data provider was tried, but after some IDE freezes and some searching around the web it was clear that Oracle’s beta provider was not compatible with Visual Studio SP1.

An instance of SQL Server Express 2008 R2 was configured and the data used by the application was transferred to the newly created database. The installation and migration of data was not recorded as part of the time to create the application.

Once the new SQL server was up and running the entity framework model was created and exposed through two domain services, one for selections and transactions and one for settings and time zones.
An issue that occurred with generating the domain services was that the entities had been renamed and saved, but not compiled, resulted in that the generated domain services used the old names. This was fixed by recreating the domain services after the model had been compiled for one of the services, and by using Visual Studio’s build in refactoring support to rename the other.

When data services had been created the data could be added to the views by dragging and dropping data as grids, forms or individual fields. In contrast to the other two alternatives two pages were created for showing transactions. One used the data-grid built into Silverlight and the other used a data-grid from a third party manufacturer\(^1\). The rationale for this was to see what kinds of features are possible in Silverlight.

When the application had been implemented and fulfilled the specification the settings screen as well as the transaction screen was re-implemented using the MVVM approach. This did not change the behavior of the application, but just introduced more code for this simple example.

5.3 Deciding which techniques to continue with
Once the implementation (Chapter 5.2) and evaluation (Chapter 6.2) of the environments in the context of the basic application had been performed it was time to choose which environments to continue the evaluation with. The results from the implementation and evaluation were presented to the company’s chief developer. The two environments which interested the company the most were selected for further evaluation after discussing the results and the environments in general.

It had already been decided by the company that evaluation of Oracle ADF were to continue so the choice was between continuing with Lightswitch or Silverlight + WCF RIA Services. It was decided to continue with Silverlight + WCF RIA Services as it was believed to better fit the company. The discontinuation of Lightswitch’s evaluation was further motivated that the product was still in an early beta stage with no announced release date.

5.4 Authentication and authorization
Before starting the implementation of the authentication and authorization scenarios described in Chapter 4.3 the documentation regarding authentication and authorization for the development environments were read through until the general idea and the steps required were well understood. This involved reading both the official online documentation as well as spending some time to look at what people have written about it to see alternative implementations as well as workarounds to common problems.

---
\(^1\) The manufacturer is Developer Express which they abbreviate as devexpress.
5.4.1 ADF

Authentication

The ADF application had ADF authentication and authorization enabled by using a wizard built into JDeveloper. The wizard allowed both roles and users to be specified and took care of modifying all required xml configuration-files to enable security. The wizard did check that passwords conformed to some complexity requirements. A custom logon button as well as a label displaying the username was added using the steps described in the online documentation (21).

The application was run to see if a user could login once the users and roles had been configured. When trying to access the application the user was redirected to a login form instead of seeing the welcome screen. When entering username and password the user was presented with a HTTP 403 “not allowed”-page independent of if the correct username and password were entered or not.

ADF allows access rights to be configured for task flows using a strategy where roles must be assigned access rights to pages and task-flows before they should be able to view it. If this is a positive or negative feature will depend on the security requirements for the application as it introduces some extra steps to be performed for each page and task-flow.

Access was granted to everyone including the anonymous (not-logged-in) user to execute all task-flows and view all pages for the application. Once access had been granted did the navigation through the pages work, but login still resulted in a not allowed page. After searching through log files on the web server it was concluded that the webserver denied login with a message of invalid password/username. After some more debugging it was noted that there were warnings in the deployment logs, which JDeveloper had been silent about, saying that the passwords were not accepted by the integrated web server (even though they were validated by JDeveloper). With new longer passwords the authentication finally worked.

Authorization

Restricting user permission

Restricting users’ access to the different entities was performed by first enabling security using the entities’ properties view (Figure 34) for all relevant entities. Authorization could then be configured by opening the correct entity and right clicking on it in the “structure” -windows and choosing to configure authorization. The screen presented allows permissions for reading, updating and deleting entities to be configured for different roles. The edit permissions will automatically propagate too many of the controls used to edit data, making the control read-only if the current user does not have permission to update its data.
Authorization for entity changes can be configured down to the property level, which requires a similar workflow as when working with whole entities. First security is enabled for the property and then authorization is configured using a separate window as shown in Figure 19. The whole process is completed by 5 + 7 clicks inside JDeveloper without writing any code.

![Image: JDeveloper dialog for configuring update permissions](image)

Figure 19 JDeveloper dialog for configuring update permissions

Nearly all permissions in Table 2 could be configured by using the authorization menu at the entity level. The exception was the edit selection permission and the confirm transaction permission (discussed further below). To restrict modifications to selections owned by the current user the built in authorization was not enough and the ability to write code for custom validation rules was (mis)used. A validation rule checking that the user either belonged to the admin role or that the owner property corresponded to the current user was written for the selection entity. An authorization rule, allowing only administrators to change the owner property, was added to the owner property in order to prevent users to circumvent the security check by changing the owner to their own name.

Limit visible results based on user
The ADF BC project had automatic access to some global variables, such as the “securityContext” which can be accessed using EL expressions. The security context contains information about the current user and the user’s assigned roles. To limit which rows could be read the WHERE statement in Code example 8 was added to SQL statement used for fetching selections. A SQL bind variable named username was used to allow parameters to be passed to the SQL query. The parameter was then bound to

---

1 Misused since Validation rules were used for a different purpose than what they are designed for. The outcome of the validation can depend on if other validation rules are run before it, modifying the entity or performing some other action with side-effect.
an EL expression which fetched the current username, allowing the query to work without any explicit arguments being passed.

```
WHERE Listparm.OWNER IN('SYSREP',UPPER(:USERNAME))
```

**Code example 8** Where-statement limiting visible selections

**Hide elements for unauthorized actions**

Two buttons were configured to depend on the current user’s permissions; one was disabled while the other was hidden if the user did not belong to the “backoffice” role using code similar to that in Code example 9 below. To disable or hide components EL expressions were used to see if the current user belonged to a role or not and set appropriate values depending on the outcome. Read only properties for controls were automatically set depending on the current user’s edit permissions. This prevented changes to transactions for everyone except administrators, but allowed everyone to try and modify all selections since the requirements to only allow the current users selection had been implemented using a validation rule.

```
rendered=#{securityContext.userInRole["backoffice"]}
```

**Code example 9** EL expression setting visibility for control depending on role membership

**Allow specific updates: The confirm function**

A button and some code was added to the transactions screen in order to allow the users to select multiple rows in the table and then call the confirm function in the business layer with the selected rows.

The Java code for the presentation layer can be found in Appendix C as Code example 16. The code extracts the currently selected rows and then calls the function in the ADF BC projects Application Module. The code presented does not work flawlessly. When not logged in the code works as intended and allows multiple transactions to be confirmed at once, but if the user is logged in only a single transaction is retrieved. In order to prevent the issue from having significant effect on the implementation time for ADF the efforts to make it work as intended were abandoned once this strange behavior was discovered.

The permission check for the confirm transactions scenario had to be implemented in code since no built in feature to configure authorization for methods attached to ADF BC Application Module was found (Code example 14). Since the function requires the status property to be modified the “backoffice” role was granted update permission to the status property of transactions. This had the undesired side-effect of making it editable everywhere it was displayed in the application.

To secure the implementation additional validation logic would have been required to validate that only allowed changes to the status field is made by user in the “backoffice” role, but it was not implemented. Possible workarounds are to either call a database
function to which the user has access or to execute the code as admin using impersonation. No documentation was found on how impersonation could be implemented for the confirm function so the functionality to implement it might not exist.

5.4.2 Silverlight

Authentication
The template used to generate the Silverlight application had already implemented most of the features required for authentications such as login/logout buttons displaying username as well as a form for logging in. What was left was to setup usernames, roles and authentication mode of choice. Adding users and roles was done from within the ASP.NET configuration utility accessible from within Visual Studio (the WCF RIA Service is hosted as an ASP.NET web application). When running the WCF RIA Services with forms authentication as done here the user and role information is saved in database tables against which the users are authenticated.

It was decided to improve the user experience by reacting to user login and logout actions. When a user logged out he was navigated to the start screen in order to prevent any possibly sensitive data to be displayed. Upon login the current page was refreshed in order to force it to update the graphical interface based on the new user’s identity. This could be implemented in the application shell by reacting to users logging in and out.

Authorization
Authorization in WCF RIA Services is configured by declaratively adding annotations to domain services. Annotations can be added to either to specific methods or the domain services as a whole, affecting all its methods. When working with WCF RIA Services two authorization attributes are available out of the box. These are “RequiresAuthentication”, which limit access to logged-in users, and “RequiresRole” (see Code example 17), which restrict access to users belonging to a certain role. Custom authorization annotations can be created to support scenarios other than role based authentication.

Restricting user permissions
To restrict access to Selections and Transactions their corresponding Update, Insert and Delete functions was annotated with the RequiresRole property.

To implement the more advanced rule, that only the current owner or an administrator can edit a selection, a custom authorization attribute was written (see Code example 19). To make it impossible to forge the owner of the selection the owner is fetched from the database using a LINQ expression. While the logic could have been added inside the UpdateSelection function the annotation based approach used by WCF RIA Services was used since it allows reuse and separation of concerns.
Limit visible results based on user
Limiting the visible Selections based on the current logged-in user was implemented by modifying the LINQ query used to get all transactions by adding the appropriate filtering (see Code example 10 below). Since the query is defined and executed on the service it is not possible to circumvent the filtering or fake the username.

```csharp
public IQueryable<Selection> GetSelections()
{
    string UserName = ServiceContext.User.Identity.Name;
    return from s in ObjectContext.Selections
           where s.OWNER == "SYSREP" || s.OWNER == UserName
          orderby s.ID
           select s;
}
```

Code example 10 WCF RIA Services query method with code to limit the selections returned (yellow background)

Hide elements for unauthorized actions
Silverlight do not come with any built in support for hiding or disabling components based on the current user’s permissions. This mainly comes from the fact that Silverlight by default do not provide information about the current user, instead the user information is provided by WCF RIA Services. When using the WCF RIA Services it is possible to disable components directly from XAML depending on if the user is authenticated (logged in) or not. It is not possible to change the behavior of component based on assigned roles directly in XAML out of the box, but several different suggestions on how to implement this was found.

Two buttons were added and configured to depend on the current user’s permissions; one was disabled while the other was hidden if the user did not belong to the “backoffice” role. Their visibility was updated in the code behind file when loading the page and when a user logged in. This approach is simple to implement and was powerful enough for the application.

```javascript
var user = WebContext.Current.User;
confirmButton.IsEnabled = user.IsInRole( Roles.backoffice );
```

Code example 11 Code used to enable button based on role membership

While the approach used (Code example 11) was simple and supported without any initial coding some other approaches worth mentioning were found:

- Expose visibility information via MVVM.
  Add properties such as IsAdmin, or CanConfirmTransaction to the ViewModel which in turn check the access permissions.

- Create an authorization class with “attached property” (a property that can be attached to any control) to implement the authorization checks. This could enable writing code like it in Code example 12 below, where the role names could be type checked at compile time to prevent hard to find runtime-bugs due to misspelled role names.
Allow specific updates: The confirm function
The confirm functionality was added to the domain service as a named update function for the Transaction class. On the client a button was added with some code calling the service side “confirm”-function for all selected transactions once the button was clicked. The implementation can be found in Appendix C (Code example 16 and Code example 17).

Since WCF RIA Services authorization is based on securing service side functions the authorization implementation was quite straightforward. A single RequiresRole annotation to the new function was all that was required to secure it. However for custom update function WCF RIA Services also validates that the user has permissions to call the normal update for the entity type. This undocumented feature gave some trouble for this specific scenario. The normal update function was changed to allow the “backoffice” role to perform updates while still preventing non-administrators from changing the transactions (see Code example 18 for the implementation). The choice of using named update in the WCF RIA Services is questionable since it required authorization code to be written inside the function. The functionality might better be implemented as a freestanding method on the domain service taking a list of IDs.

5.5 Internationalization
Before starting internationalizing the applications the documentation for the corresponding development environments were read through including the relevant chapters on localization for the different platforms in the books Building Rich Internet Applications with Oracle ADF Business Components and Oracle ADF Faces (22) and Professional C# 4 and .NET 4 (23).

As with authentication and authorization some time was spent looking at what people have written about localization for the environments to find alternative implementations as well as workarounds to common problems.

The implementation was divided into the areas of translation, adapting formatting of date and numbers and adding time-zone awareness. Each area was implemented completely in turn for both environments before starting to implement the next, in contrast to previous scenarios where implementation was performed for one development environment at a time. It was decided to start the development with Silverlight instead of Oracle ADF. The reason for this was that if the implementation

---

1 While a normal method could have been added to the domain service, the named update function was used since its purpose is to support more complex update scenarios.
takes less time during the second implementation (caused by already having implemented the features once before) it should not affect ADF in a negative way.

5.5.1 Silverlight
This Chapter describes how the Silverlight application was internationalized and localized. It assumes that the reader has read Chapter 3.2.7 which describes the basics of Silverlight localization.

Extracting hardcoded strings
Visual Studio is not shipped with support to extract hard-coded strings from XAML to resource files so the hard coded strings were located and manually entered into a resource. Binding to the strings from XAML could be performed using the build in visual data-binding tooling, but most of them were entered directly in XAML with some cut and paste since this approach felt faster. When working directly in XAML code there were no auto-complete functionality for specifying the names of the strings, making manual edits prone to misspellings not detected by the compiler. Fortunately the design window picked up the changes while coding, allowing direct visual feedback regarding if the names were correct or not without starting the debugger.

When around half of the application had been prepared for translation the type of resource file used was switched in order to check the tooling for the other resource type, XAML resource dictionaries. Almost any XAML property can be extracted to a resource dictionary from the property windows in Visual Studio. Extracting the strings to a resource and updating the XAML to use the string from the dictionary is performed by right clicking the property and choosing extract to resource from its popup menu. This approach was faster and more convenient than the previous attempt. Visual Studio also provides support for showing the values in the resource dictionaries with search support by choosing “Apply Resource” in the property window’s context menu.

While visual studio did not provide built in support for extracting XAML strings to resx files two promising plug-ins where found, one of them was available via Visual Studio’s built in extensions manager and the other was hosted at the codeplex.

Translating the application
Translation of the application strings was quite straightforward and can be summarized by three main steps.

- Create copies of the resource files and translate the copies.

---

1 “Resource Refactoring Tools 2010” (35) allowed hardcoded strings to be extracted to “resx” files by adding extra alternatives to the right click menu inside the code editor. When extracting the addin presents a screen where similar already extracted strings can be chosen from or a new resource entry can be created. The addin did not work with Silverlight 4 XAML out of the box, but generated code XAML that was only compatible with WPF. Since the source code is available it could easily be modified to also work for Silverlight.

2 “WPF Localization Addin” (34) worked with Silverlight and allowed detection and extraction of hardcoded strings for whole files at once.
• Rename the copies so they have the same name as the original resource, but with region code added. Ex rename EntityStrings.resx to EntityStrings.sv-SE.resx.
• Edit the project settings and add “sv-SE” to the list of supported regions.

For the resource dictionary two version of the application were built, one against the original version and one version against the translated strings.

**Formatting of numbers and dates**

.NET allows each thread of execution to define separate locales used for the graphical interface and for formatting data. In a Silverlight application these locales are automatically set to that of the browser meaning that strings will use the formatting the user is accustomed to, unless otherwise changed.

The settings screen was updated so that it generated a list of available cultures to select for program language and formatting. The controls defined in XAML still used “en-US” as locale even after having changed the .NET locale. The solution was to set the Language property of the Silverlight top level window. The documentation had to be consulted again before understanding that this was a documented feature aiming at compatibility with the “xml:lang” property used to localize XML files.

Once the functionality to change language had been implemented the settings screen was converted to follow the MVVM design pattern. This change allowed the implementation to become cleaner and somewhat easier to understand.

**Adding timezone support**

Initially no documentation on how to handle time-zones was found in the WCF RIA documentation so the “normal” .NET classes for dealing with times and time-zones were tried. After searching the WCF RIA Services forums and Google some suggestions were found.

In .NET there exists two different data types for storing a date and time, they are the DateTime and the DateTimeOffset. The former contains information about a point in time and whether the time is specified in UTC or the local timezone while DateTimeOffset stores the timezone as well as point in time. The data in the database was converted to the DateTimeOffset data type and the corresponding entity was updated and regenerated to match the database. This failed with a message when that the DateTimeOffset is not supported by WCF RIA Services when trying to compile the client.¹

Before starting a second attempt the database and application were changed back to use the DateTime type again. The second attempt was to follow a guide for the old beta version of WCF RIA Services, where a property was modified to change if DateTime properties stored in the database should be considered to represent UTC time, or local

¹ It is however supported in the preview of WCF RIA SP2, which has since then been released.
time on the service. This attempt failed since the property mentioned had been removed before the initial release of WCF RIA Services.

Finally custom “calculated” properties that translated the time retrieved from the database from UTC to local time were added to the transaction entity. These new properties were then used instead of the old properties when displaying the values in the graphical interface. This solution, which is found in Code example 21, did not provide completely automatic translation between time-zones as some manual coding was required for each relevant property. It was however easy and quick to implement.

5.5.2 ADF

Preparing for translation

While JDeveloper support the use of resource bundles for allowing translation of entities none of the required strings were defined in the resource bundles automatically. By default the labels for data is the same as the column names in the database, but the label names (and any other localizable settings) must be defined manually for each property before they are specified in a resource bundle. Setting up the display parameters for all properties was simple but took some time since every single property had to be selected and then configured in separate windows. While this procedure could have been somewhat more streamlined it was considered a minor nuisance since these settings should be set up when creating the properties in the first hand.

JDeveloper provides support for automatically creating resource strings when configuring the UI visually by using the properties window. This approach is only recommended for small projects, since it very easy to create inconsistent strings and duplicates (22). For the ADF application the recommended approach described below was used instead.

When working with ADF Faces RC web pages strings other than labels for data entity properties were moved to resource bundles. JDeveloper provides graphical support for performing parts of this task. After a JSF component has been select in the JDeveloper’s visual designer the “select text resource” window (Figure 35) can be opened for string properties by using the right click menu of the property in JDeveloper’s “Property Inspector”-window. This window allows the developer to search among the defined text strings as well as to create new entries in the resource bundle (if the desired text is not present in the resource bundle). This part of the preparation took some time since already specified values were ignored, meaning that the strings had to be entered into the window for each string that had been hard coded in the UI. While the “select text resource” window was very handy for internationalizing the application it was not bug free. It automatically added code for accessing the correct resource bundle, but used the wrong name for the resource bundle (despite having configured the project to use a
specific bundle). This resulted in somewhat inconsistent errors (from http error 404, page cannot be found, to a message box saying that the resource bundle could not be found) when trying to navigate to any of the pages.

**Translating the application**
The translation was straightforward and follows the same steps as that for the .NET application:

- Create copies of the resource files and translate the copies.
- Rename the copies so they have the same name as the original resource, but with region code added. Ex rename bundle1.properties to bundle1_sv_SE.properties.
- Edit the project settings and add “sv-SE” to the list of supported regions.

**Formatting**
The settings screen was updated to allow the language to be changed during runtime. This was accomplished by using the code available in (22). An attempt was made to allow the locale and language to be changed independently, but this failed. The documentation (24) said that the locale could be set with EL to the language code to use, but in reality it threw an exception (preventing pages from loading) if setting it to a string. After spending some time trying to resolve the issue it was decided not to implement it.

**Timezone support**
Despite the fact that ADF should have support for automatically converting date and times to the timezone of the user all attempts to use this feature failed. Around 8 hours were spent trying to get the feature working before abandoning it. Below are some of the attempts described.

Before attempting to test the support for automatic timezone conversion some of the data-types used for storing entity properties were changed from date time the Oracle’s timestamp type (which stores date + time). First the changes were performed in the database (the time to do this was not included in the total time). Then the entities in the ADF BC projects where updated using JDeveloper’s built in support for updating entities based on database changes.

When it had been verified that the changes had propagated all the way to the view objects, the application was run. However the changes had not been enough, an exception was thrown with a message of an invalid conversion being done. The problem was found after some time troubleshooting. The data-types had not been changed for the auto-generated Java classes representing the entity, but the classes had to be manually edited when the entities were changed.

Some further attempts to get the timezone support working included:
- The “time-zone” property in the global configuration file “Trinidad-config.xml” was changed while using the timestamp data-type to represent times. Different values such as UTC and PDC were set but the same time was displayed. Trying to set other values or using an EL expression was an easy way to get the web requests to crash.

- The data-type was changed to “timestamp with timezone”, which store timezone information with the date and time information in the database. This resulted in errors when fetching data from the database complaining about the formatting of data. It is possible that this was cause to a mismatch in formatting between this version of the database and newer releases.

- The data type was changed to “timestamp with local timezone” data-type that assumes the timezone of the web server is the timezone used in the database.
  - When accessing the application from another machine with another timezone such as UTC time or Finnish time (UTC+2), the requests crashed.
  - When changing the timezone of the server to another timezone such as UTC time or Finnish time (UTC+2) within the operating system or by sending a parameter to the Java virtual machine, the requests crashed.
6 Discussion and measurements
This Chapter aims at providing both a quantitative and qualitative comparison of the frameworks based the time required for the implementation described in Chapter 4 as well as the experience gained from the implementation.

The Chapter is divided into a subchapter for each of the major implementation scenarios. Inside each subchapter the different frameworks are discussed and measurements from the implementation are presented. Each subchapter is concluded with a presentation of the framework recommended for that specific area.

Some of the content of this chapter will require reading the implementation first to fully understand it, but the comparison should be informative enough to allow them to be read without first consulting the implementation.

While trying to be objective it is impossible to be completely objective and the pros and cons as well as the discussion will reflect the opinions of the author.

6.1 Previous experience
As any previous experience with the development environments will most likely influence their rating here some of the previous programming experience is presented here.

Lightswitch was completely new to the author who had no previous experience with any similar approach to development. Visual Studio had been used earlier, mainly for programming C++, but had also been used to create a simple C# web site back in 2005 using ASP.NET. The author had used Java including Enterprise JavaBeans (EJB 3) in a number of university courses as well as some very small projects. Previous Java development had been performed using NetBeans and Eclipse as IDE with glassfish as application server. While no JSF development had been performed the previous experience with HTML, CSS and ASP.NET (which uses an approach with many similarities to JSF) was very helpful.

Overall the skills in Java were considered better than those in C#.
6.2 The basic application

This chapter is divided into a couple of subchapters with pros and cons for the different environments based on the implementation experiences. This is intended to give a quick summarization of the implementation experience. The pros and cons are followed by measurements from the implementation and a comparison of some key areas.

6.2.1 Oracle ADF

Pros
- JDeveloper provides a single place to specify parameter names, UI hints and validation for entities
- Reusability of Views (web pages), Queries (ViewObjects) and Model (entities)
- Built in Query designer control which support queries to be saved
- Good drag and drop support for displaying data in the application:
  There are many ways data can be visualized and the different controls (tables, forms, list boxes etc.) are grouped depending on purpose (such as selecting a single item, selecting multiple items etc.)
- Much can be achieved without coding Java
- Found good tutorial for creating both the shell UI as well as setting up navigation between pages.
- Support on all major operating systems with any of the most popular browsers

Cons
- Unstable development environment
  - Connection could not be created during “create project wizard”,
    - Could add it in other ways, but did still get error even though it succeeded
  - Many crashes, each taking at least 5 minutes to get started again.
  - An entity reported as unused by the IDE was removed, resulting in compile and runtime errors until some uses had been removed after finding them with notepad++
  - When converting a button to a link JDeveloper warned that some unsupported properties would be removed, but one was left leading to runtime errors. With no source row specified the troubleshooting was done in wrong place.
- No very user-friendly debugging
  - Little compile time checks
    - Errors with data-bindings, navigation or components placed wrong are not detected until run-time.
  - Hard to understand error messages like “ADFC-06002” and “AD_FACS-06074”
  - Long stack traces, but no visible calls to user controls or methods
- Not easy to track down data binding errors, only get “null pointer exception”
- Not straightforward to navigate from a button to the code executed when it is clicked
- Slow application/debugger startup
  - Up to a minute to show effects of changes
  - Above 2 minutes if server is not started. Since the screen layout in JDeveloper is different when server is active it is easily stopped when editing the application.

- Maintainability
  - Encourages drag and drop of data, which generates code which need manual changes all around the application once properties are added, removed or changed. This applies to all types of controls except the read only dynamic tables which can generate the columns displayed at runtime.
  - Changing data entities in the IDE requires the automatically generated Java classes to be edited manually since changes are not propagated to them. This led to exceptions at run time (no compile time errors were created during the development but it might happen in some cases).

- 2GB of RAM is not always enough when developing even a simple application
- When using the link control the user is presented with something that looks just as a standard web hyperlink but without a right click menu which can be confusing.

6.2.2 Lightswitch beta 1
Lightswitch beta 2 was released shortly after the evaluation of beta 1. The application was later implemented in Lightswitch beta 2 since Lightswitch beta 1 applications could not run in Visual Studio 2010 with Service pack 1 installed and Lightswitch’s project files could not be upgraded from beta 1 to beta 2. The evaluation of Lightswitch was not affected by the reimplementation which took a mere hour, but some of the comments on Lightswitch beta 1 have had footnotes attached to notify the reader of features that had changed in beta 2.

**Pros**
- Very fast UI development.
  - Much of the applications navigation is wired up automatically
  - Many screens (such as edit, details, new) are created automatically without the developer having to add them.
  - Screen templates allow the creation of screens depending on purpose with good default generated appearance to start from.
  - Runtime UI designer allows screens visual appearance to be modified with instant visual feedback while debugging the application.
- Best documentation for getting stared at the webpage.
- A number of different data access methods (such as SQL Server, WCF RIA Services, SharePoint and Access in the future) could be used in parallel.
- Data can automatically be exported to excel when run on the desktop.

**Cons**
- Limited control of placements and appearance of UI controls.
- Mapped to whole database tables, not parts of them. Although columns could be hidden they would still be fetched. Could have created views in database to handle this, but then business rules may have had to be defined in multiple places, once for each view for the data.
- The built in refactoring support (such as rename) was lacking.\(^1\)
- Debugging didn’t work\(^2\).
- No batch-edit of data entities column properties.
  This is probably not a problem when using it while designing from the start, but it was not fun when using with a legacy database where lots of columns needed changing.
- No design support for multilingual applications inside the designer, but a single language and settings are chosen.

### 6.2.3 Silverlight

**Pros**
- Best end user experience
  - Especially impressed by 3rd party data-grid
  - Integration with the browsers navigation features
  - Responsive and fast feedback on actions
- Many controls when combined with the free toolkit, even more when considering 3rd party controls.
- Support for Drag ‘n drop of both visual components as well as data entities
- Simple conceptual idea with domain context and domain services makes it easy to understand the code, debug it and to start writing own code.
- The visual designer executes the page and its binding allowing bindings as well as some startup code to be verified from inside the designer without having to start debugging the application.
- Much documentation with especially good “how do I” samples for WCF RIA Services.
- Maintainable code

---

\(^1\) This seems fixed in the Beta 2 release of Lightswitch.
\(^2\) This was later verified to be fixed in the Beta 2 release of Lightswitch.
Data-grids and forms can automatically generate columns, controls to display data entities with support for customizing the appearance by adding annotation to an entity.

- Writing code using MVVM pattern facilitates testing and makes the code easier to update
  - Many features can be implemented using different techniques and free libraries.

**Cons**

- WCF RIA services is not supposed to work on mono until version 4. (25) This currently prevents the client from running on all major operating systems.
- More manual work required than when using Lightswitch
- More options for different aspects of the application can sometimes make it difficult to know what is best suited for a given purpose.

**MVVM**

Silverlight might be the framework where the developer has the most freedom in choosing how to implement the application. This can both be positive as it allows the development to be tailored to fit the application being developed as well as the development team, but it still required some choices to be made. One of these choices is regarding how the graphical interface will be linked to the data model. The development can follow the drag and drop approach (data over forms) as used when developing for ADF and Lightswitch, or the MVVM pattern.

Using the MVVM approach for the transaction screen when it only displayed data felt like over-engineering the application and wasting time. However did it actually made the implementation easier with cleaner code as a result when used for the settings screen. Developing using MVVM felt much like when starting to write units test for code: there is some extra coding required that initially takes some time to get used to, but the return of the investment is probably worth the effort with more robust and maintainable code as a result.

The recommendation if staring to develop for Silverlight is to use MVVM for screens containing additional logical processing of the displayed data. The drag and drop approach to data over forms application can be used for screens just showing data, but an MVVM approach coupled with a separate data access layer within the client should be considered as it would allow the graphical interface to be completely decoupled from the business layer.
6.2.4 Measurements

**Time to implement**

Looking at the time used to implement the basic application with the three different platforms (Figure 20) one can see that Lightswitch (5 hours 37 minutes) was the framework requiring the least amount of time to implement the application. 9 hours and 40 minutes was spent to implementation the Silverlight application which is 72% more time, while 21h hours and 12 minutes was spend on Oracle ADF which is around 277% more time. While the comparison cannot be said to be completely fair\(^1\), the timings provide a clear indication about how much effort was required to start using the different frameworks.

![Figure 20 Time used to implement basic application](image)

The fast implementation using Lightswitch comes from the good online documentation, which covered all the scenarios for the simple application, as well as the basic application being suitable for forms over data application development, the type of application targeted by Lightswitch.

The implementation of the basic application took the most time when using JDeveloper and Oracle ADF. Some of the additional time can be accounted for by being the first application created, but that do only account for a small part of the time used. A big

---

\(^1\) The Lightswitch application was the only application that did not at any point change background color based on the screen displayed, instead it showed the selection chosen in tab header. For Silverlight some extra 3rd party controls were added. One of them (a datagrid) was added on an extra screen not present in the other implementations.

\(^*\) Deployment of the ADF application was abandoned after having spent one hour trying to deploy it to a freestanding application server instead of running it on the weblogic server bundled with JDeveloper.
part of time used to implement the ADF application was spend debugging the application, which to some degree reflect that the visual development used by JDeveloper requires some more study before being usable, but it also reflect a slower development environment with less friendly debugging facilities than the alternatives.

As for the Silverlight application it required around 2 hours more for the actual implementation than the Lightswitch application, but it did on the other hand use an extra screen with 3rd party controls such as the extra data grids that were responsible for part of the extra time required, but not all. It only require around one third the time for the actual implementation compared to the ADF implementation so it can still be considered quite fast.

**Online references**

A number of different keywords were searched for on Google’s Swedish web page in order to get an estimate of the amount of material on the different frameworks, their popularity and number of users. The number of reported hits can be found in Figure 21 below.

\[\text{Figure 21 Number of search results for the different technologies}\]

From the results of the searches it seems like Silverlight as technology has significant more material about it online than any of the two other frameworks. One reason for this might well be that Silverlight as a UI technology is a more general technology than Lightswitch (which is built in Silverlight) ADF Faces (which are components built on top of JSF as UI technology).

While not scientifically proved it felt like there was much more information written about Silverlight by enthusiasts using it outside of work than for any of the other

---

* WCF RIA Services was previously called .NET RIA Services, “RIA Services” was chosen as it matches both the names.
frameworks. This gave the impression of a much larger user community. The reason for this is probably related to the licensing differences between Oracle ADF and Silverlight. Silverlight and .NET is completely free to use and develop for with a number of free IDE’s, both special editions of Visual Studio as well as open source alternatives.

When developing for Oracle ADF the JDeveloper IDE is provided free of charge, but licenses are required for deploying ADF applications. The license is included in licenses for Oracle’s application server, but must be purchased separately if using an application server from another vendor, which “only” cost thousands of dollars. (26)

The relatively high numbers of hits for Lightswitch were somewhat surprising since it is the youngest technology as it was announced at 3rd august 2010 (27). It seems that people are starting to follow the development of Lightswitch. With vendors of 3rd party Silverlight components starting to prepare these for Lightswitch the development around Lightswitch will be interesting to follow.

6.2.5 Comparison

End user experience

When evaluating the end user experience a number of different factors were taken into account. Some of these factors were responsiveness of the graphical interface, how the application behaved according to user expectations and interoperability with other applications such as Excel.

The Silverlight and Lightswitch applications were somewhat similar when it comes to responsiveness of the graphical interface. Their responsiveness of both validation and navigation felt significantly better than ADF, even when running over a fast local network. When navigating between pages they both respond immediately by opening the graphical interface while data is fetched in the background. The data is then presented when it arrive, which can take some extra time. For the ADF application no navigation occurs before the data has been fetched and processed by the application server. While all data fetches might take the same time in all technologies it often feel likes it takes twice the time for the ADF application.

Since all applications were run in the browser the end user will expect them to provide a user experience similar to that of normal web pages. While it was believed ADF would be the best alternative in this aspect it was not. The browsers’ right click menu was disabled for many controls including links. Any use of the browsers’ URL field or navigation buttons messed up the application completely. The Silverlight application did however work as expected with respect to the browsers’ navigation buttons and URL field. Lightswitch did not integrate with the browser, but new content was always opened in new tabs so this was not as disturbing as no navigation took place in the application.
One of the strength of the ADF platform is the built in control for entering and storing queries. To have similar features in Silverlight or Lightswitch 3rd party controls needs to be used, but then the controls used less screen real estate and looked better.

When it comes to integration with other applications Silverlight was the only technology which by default provided copy-paste support for data in data-grids. Lightswitch has an almost as useful feature that allows the content displayed in data grids and lists to be opened directly in Microsoft Excel. It was disappointing to see that Oracle have put effort into making copy-paste unavailable for most ADF components, since this feature is otherwise automatically provided by the web browser.

All in total it was considered that Silverlight had the best end user experience.

**Developer experience**

IDE – General Impression
After the initial evaluation of the environments, Visual Studio (targeting Silverlight and WCF RIA Services) was considered to provide the best IDE experience.

Visual Studio provided the most responsive IDE with fastest startup time, most modest memory requirements and fastest compilation times. Loading the Lightswitch project with Visual Studio provided the next fastest startup with ADF (which could take minutes to show up) as the slowest IDE. Compilation times were longest for the Lightswitch project with ADF at second place.¹

All environments featured some kind of auto-complete functionality. The auto complete functionality provided faster and more accurate results in Visual Studio than in JDeveloper. JDeveloper did however provide better support for resolving names that belonged to namespaces not opened (imported). It offers a menu with alternatives to choose from when using a type which was not defined in any of the already imported packages. To use the same feature in Visual Studio the keyword not found had to be right clicked and the option “resolve” chosen from the context menu (an approach also supported by JDeveloper).

During development in JDeveloper it was required to navigate between many different windows, while much of the same functionality was performed using much less navigation in Visual Studio that allowed faster development.

The number of available extensions for the different IDEs was also examined. 57 extensions targeting JDeveloper were found at oracles website (28) and 1712 extensions targeting Visual Studio 2010 were found on Visual Studio’s website (29) giving an indication that the latter is more popular.

¹ Lightswitch beta 2 did feature somewhat faster compilation speed than that of ADF.
Debugging experience
The debugging experience for the different environments varied more than expected. Lightswitch did on one hand allow the UI to be modified while debugging, a much useful feature for interface related issues, but the ability to debug code was the worst since the execution did not stop when reaching breakpoints\(^1\).

JDeveloper and ADF provided good support for debugging the Java code. Unfortunately the experience was not as good for debugging database problems, the UI or non-Java code (like EL expressions or Groovy). The developer was often faced with short cryptic error codes or long stack traces without information of what graphical component or expression did cause errors when debugging these parts of the application. The lack of compile time type checking and error checking for many parts of the application lead to that many of the issues with JDeveloper and ADF did not manifest until runtime, requiring more time spent on testing. This in combination with slow application startup time contributed much to the time used for the implementation.

The Silverlight debugging experience was the smoothest of them all. Breakpoints could be set in both the Silverlight client as well as in the WCF RIA Service. Debugging invalid behavior was further simplified compared to the ADF experience since the conceptual model behind the frameworks was simpler, making the flow of execution easier to follow. Visual Studio did also provide some useful features which helped creating the UI. The first and maybe most impressive detail was the fact that the pages actually was executed and run in the designer while they are created. This allowed immediate and accurate feedback on the UI appearance, startup logic as well as data-binding behavior without even starting the application.

If there are problems with invalid (misspelled) data-binding when debugging a Silverlight application the application will by default appear to function as if the binding did not exists. However the application will log information about the binding to the debug-log. The log contained detailed information such as class and id (name) of the target component as well as source making the binding errors easy to locate. While it would be even more convenient to be able to break and inspect the binding itself the behavior works well\(^2\).

Documentation
While the numbers of search hits found in Figure 21 may not be 100% accurate it is closely related to how easy it was to find good material online for the different technologies.

Lightswitch was the technology which did provide the least amount of material, but the material found on the official home page was excellent for the application developed,

---
\(^1\) Debugging worked in beta 2
\(^2\) Setting breakpoints for data-bindings is supported for Silverlight 5 development
giving a better introduction to Lightswitch than for any of the other environments in half the time. It is however unclear how much high quality documentation exits for more advanced scenarios, but since Lightswitch applications run on Silverlight and .NET the documentation for these would be useful to a Lightswitch developer.

Documentation was somewhat easier to find for ADF than it was for Lightswitch. Many of the pages found contained material more advanced than what the Lightswitch related pages had contained. It was still not always easy to find good information about error messages or specific feature such as the automatic timezone adjustment. Much more documentation was found for Silverlight and WCF RIA Services than for any of the other alternatives. The online documentation for the WCF RIA Services provided a very good starting point for building WCF RIA Services applications, but did also some contain some general Silverlight material. The official online documentation held high quality overall and had integrated functionality to comment on the documentation to share knowledge with other developers, which provided some insights beyond the documentation.

Ease of use, Productivity and Maintainability

Lightswitch provided the simplest and most productive IDE where it was easy to know where and how to implement the features for the basic application after only watching some limited online documentation. However the simplicity comes at a price where the development and graphical interface is limited to forms over data applications. Renaming and changing some entities required both manual code edits (since the rename functionality was broken)\(^1\) as well as parts of the UI to be modified. Lightswitch was not the only development IDE requiring changes in code when modifying entities, but the Java classes generated for the ADF entities also had to be manually edited since JDeveloper’s visual editor only updated the entities xml-files for many changes.

Creating the visual appearances in ADF and Silverlight was somewhat similar both in effort to get started as well as how controls and layout components could be added in an xml like format using drag and drop. Both frameworks allowed graphical interfaces for data over forms application to be developed using approximately the same effort. JDeveloper has an advantage over Visual Studio for development of simple applications as it presents a larger selection of controls to be chosen when dropping data on a page.

The frameworks differ in how well the graphical interface can handle changes to entities. Both the Silverlight data-grid and data-form could generate the visual appearance at run-time based on the data being displayed. ADF and Lightswitch had all the properties and controls specified at development time (except for ADF’s dynamic read only table \(^\) ) requiring changes everywhere an entity is displayed if a property is added, deleted, renamed or modified.

\(^1\) This seems to work in Lightswitch beta 2
Defining the data and settings up the business service was not that very different in WCF RIA Services and ADF BC, both approaches allowed database tables to be selected and used as entities. The large difference is when it came to extending the application and understanding the application flow. WCF RIA Services has a conceptually simpler approach with a service class that seems to be accessed directly on the client. The Silverlight programming felt easier than that of ADF Faces especially when interacting with the service. The difference may be most easily visualized by comparing the code required to access the confirm function (Appendix C - Code example 15 and Code example 16).

While Lightswitch allowed the fastest implementation it did not handle entity changes without effort, an area where Silverlight allowed far more changes without modifications all over the application. The initial effort to create an application was lower for Lightswitch than for ADF and they both required a similar amount of manual changes for entity modifications. What makes Lightswitch better than ADF for simple pages are the built in page templates.

**Recommended development environment**

For each of the areas discussed the different development environments were graded according to how well they performed. A 1 means that the development environment was considered the best within that specific area with 2 meaning it was the next best environment. The results can be found in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>JDeveloper; Oracle ADF</th>
<th>Visual Studio; Silverlight &amp; WCF RIA</th>
<th>Lightswitch beta 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Debugging</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Documentation</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Ease of use</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Productivity*</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Maintainability*</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1st places</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2nd places</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3rd places</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

* Productivity represents how quick simple forms-over-data (CRUD) applications can be created using the environment while Maintainability represents the estimated effort to create and maintain larger more complex applications.
6.3 Authentication and Authorization

6.3.1 Measurements
The time used to implement the authorization scenarios can be found in Figure 22. The implementation was faster in Silverlight than when using ADF. This gives an indication that authorization, especially for nontrivial scenarios is easier to implement when using WCF RIA Services. For trivial authorization scenarios it likely that it takes roughly the same time to implement entity based authorization in both development environments. For a WCF RIA Service a single file must be modified with a maximum of four annotations to be written. In JDeveloper the configuration is purely visual but slightly more steps are required; first the entity must be opened and configured for security in the main edit entity window. Then the “structure” window should be opened and its context menu should be brought up (by right clicking) so the developer can choose to open the authorization window for that entity.

![Figure 22: Time required for adding authentication and authorization](image)

It is also worth noting the difference in amount of code required for the confirm scenario and the associated time required for it. The difference in time required for implementing the confirm scenario was not considered when comparing the frameworks authentication and authorization support, but it is taken into account for the final recommendation.

*: The confirm scenario is not regarded as 100% successful for the ADF implementation as the status field had to be editable everywhere for allowing the server side implementation to change the value.
### 6.3.2 Comparison

Both frameworks implement role-base authorization, but they authorize different types of actions.

ADF BC authorization is built around entities and it is possible to authorize reads, updates and deletes, where updates can be authorized down to the property (column) level. Silverlight on the other hand place the authorization at the method level, which allow entity oriented authorization for read, updates, deletes but also allows inserts to be treated separately from updates. By basing the authorization around service methods the same authorization method is used to authorize access to all method exposed via domain services and not only functions for working with entities.

When it comes to expressiveness of the authorization frameworks the approach used by WCF RIA Services is the most extensive, allowing any type of custom authorization rules to be implemented. ADF security is limited to the predefined types requiring validation rules to be misused as authorization rules for more advanced scenarios.

On the client side the situation is somewhat different. Both application frameworks have access to some kind of object with security information, but only ADF Faces can check role permissions directly in the UI out of the box. ADF provides a graphical interface to configure who can open which screens from within the designer. While Silverlight could implement the same functionality in code behind or using MVVM some custom extensions must be coded to get the same support directly in XAML.

#### 6.3.3 Recommended development environment

Silverlight + WCF RIA Services is deemed as the better environment for authentication and authorization as it allows a wider range of authorization scenarios to be implemented. It was also the only environment where all the security requirements for this prototype application could be implemented as desired and without unwanted side effects.

ADF did provide somewhat better support for adapting the UI based in user permissions, but Silverlight can with minor effort be extended to provide similar or even more powerful support. The approach used in the implementation with permission checking in the code behind was not much worse than that used by the ADF application. It did not take more time to implement than the ADF counterpart and it did also have the advantage of being type checked by the compiler, preventing run-time bugs.

### 6.4 Internationalization and localization

This chapter contains a discussion around the internationalization and localization support in the ADF- and .NET platforms based on the internationalization and localization of the applications previously developed (Chapter 5.5). The chapter also present the time used for the implementations and it ends with a presentation of the
development environment which was perceived to have the best tooling for internationalization and localization.

6.4.1 Measurements
The time required for the different internationalization scenarios can be found in Figure 23 below. The internationalization and localization of the ADF application took around 2.5 times more time than for the Silverlight implementation, when excluding the time for time-zone adjustment.

![Figure 23 Time used for implementing localization](image)

The most important aspects here are the time used to prepare the application for localization where hard coded strings were moved to resource files, the time used for the actual translation and the time required for testing these as these best represents the time requirements in the long run.

The “Testing” measurement include the time used to run (and start) the application in debug mode to verify the translation of strings as well the localization of formatting. JDeveloper with ADF did perform worse in this aspect compared to Silverlight. Apart from much slower application startup and shutdown the invalid code created when JDeveloper was used to internationalize hard coded strings also contributed to this measure.

* The time-zone support was never made to work for ADF. It was abandoned after almost 8 hours. This measurement includes all the time used for implementing the time-zone support including debugging and searching for other solutions (in order for it not to affect the other scenarios)
While there was a difference in time used to allow the locale used for formatting to be changed it is regarded as having little importance as this should be a one-time investment.

6.4.2 Comparison

Both development environments base their support for multilingual applications on similar technical solutions, where strings are stored centrally in special files instead of being hardcoded into code or UI. The .NET approach is slightly more technically powerful as it allows not only strings to be stored in these files, but also other resources such as images.

JDeveloper’s visual approach for configuring error messages and entity properties allows it to store these types of strings in resource files automatically. The Silverlight and WCF RIA approach instead rely on writing annotations in code, which do not allow the same kind of built in support. While the different annotations for display information, validation and authorization do support the use of resource files the strings are manually added to the resource files and then referenced by the annotation. This approach using annotations for internationalizing entities was unfortunately not described in the book or the localization overview page online, so it was not used during the localization process which could have speed up the process. However this approach is used by the project-template (see Code example 20) and it seems to be the most promising approach when it comes to internationalizing data displayed in Silverlight.

The frameworks offer a slightly different approach for accessing the resources from code, which can be seen in Code example 13. The .NET resource files have classes generated which allow typed-checked access to their content. To access a resource in ADF one must first get a reference to correct bundle (identified by a string) using the correct locale and then get the value by supplying the key (as a string), which takes more time and is more error prone than the .NET approach.

```
string Text = ErrorResources.LoadError;
Locale locale = ADFContext.getCurrent().getLocale();
ResourceBundle bundle = ResourceBundle.getBundle("view.ViewControllerBundle",locale);
String Text = bundle.getString("LoadError");
```

Code example 13 Code for accessing a resource from C# (Top) and Java (Bottom)

Visual Studio does support hard coded strings in XAML to be extracted to and to search in XAML resource dictionaries, but not resX resource files. JDeveloper does not allow strings to be extracted, which would have made the tooling more powerful, but will
automatically search among all previously defined resources when entering a new text using the select text resource window.

The time required for the ADF implementation could have been significantly reduced if JDeveloper had copied the already entered string to the “select text resource” window. The effect this has on normal development is questionable since the window should be used from the start when developing an application with localization in mind.

### 6.4.3 Recommended development environment

JDeveloper with ADF is deemed the development platform with the best built-in tooling for internationalization and localization.

This is an area where JDeveloper’s visual approach to development pays off as it allows much of the interaction with resource bundles behind the scenes. The internationalization could be performed more transparent in JDeveloper than in Visual Studio, where the strings were added to the resources manually. While the .NET approach allowed more types of resources as well as much more convenient access in code the better tooling in JDeveloper gives it this recommendation.

It is worth nothing that some free plug-ins improving the support for working with resource files in Visual Studio found. These were not considered during this evaluation and could tip the bowl in favor of Visual Studio. Similar plug-ins for JDeveloper were searched for, but none were found.
7 Recommendation

This chapter will start by looking back at the initial requirements for the future platforms and how the alternatives relate to the major requirements. This is followed by summarizing the results from the implementations and finally a development platform is recommended.

7.1 Looking back at the initial requirements

Future prof

Both Java and C# belong to the top-5 most popular programming languages as of May 2011 according to TIOBE (30). This makes it unlikely that it should be hard to find people proficient with these languages in the foreseeable future. While Java is the language with most online activity according to the list is it worth noting that developing with ADF requires skills with JDeveloper, JSF, the ADF Framework itself as well as EL, Groovy, CSS and HTML. Groovy is not found until somewhere around 50 and 100 in the TIOBE list. When using Silverlight and WCF RIA Services the developer should have some knowledge of XAML and the entity framework, but is quite likely that .NET developers have experience with the technologies as they are major parts of the .NET platform.

Lightswitch is still in beta and while it looks promising it is unclear how the adoption will look like and how the product will evolve making this the most risky approach.

Easy to get started

Lightswitch provide the absolutely easiest platform to start using, based on the implementation of basic applications, with Silverlight somewhat more complex to use followed by JDeveloper and ADF.

Performance

All frameworks provided reasonable performance for the implementations. The startup times for the Silverlight applications was both somewhat slower than that of ADF (when it was downloading a new version) and faster (when the current version of the application had already been downloaded). All implementations provided reasonable performance, but the ADF application felt somewhat sluggish at times.

Initially some concerns had been raised around the performance of the 3-tier approach used by the .Net platform, but this was not noticed at all. It is actually not that strange since the ADF application is actually a N-tier architecture where the web browser hosts the presentation tier which interacts with the web/application-server, middle tier(s). While no performance measurements were captured it is much possible that the Silverlight application is somewhat faster and uses less bandwidth than the ADF application. Only the data (which can be sent binary) is sent between the client and the WCF RIA Service in contrast to the ADF approach where both UI and data are sent as text.
**Easy to debug**
This was considered important for the choice of framework, but only one the Silverlight + WCF RIA Services approach fit this description. It is however likely that Lightswitch debugging works at least almost as well when it is released. JDeveloper and ADF suffered from a variety of issues as described in chapters 5 and 6.

**Maintainable and testable**
These implementations do not give experience for a well informed and completely fair comparison, but it seems that Silverlight + WCF RIA Services is the most promising here. It was the only framework where neither UI nor some generated code had to be modified when making small changes to the entities. The ability to have Silverlight controls auto generate their appearance depending on the content looks very promising. The ADF project required manual modification to auto-generated classes as well at the UI. Lightswitch beta 1 did not update naming everywhere when changing entities which made manual modifications to the code required for it as well.

**Testability**
This was not evaluated as part of this thesis, but all platforms support some kind of testing. For testing an ADF application one can install a JUnit extension to allow writing unit tests in Java for exercising the code part. To test the graphical interface a web testing framework such as selenium (31) could be used. Visual Studio comes with a build in test framework. No special framework was found for testing Lightswitch applications, but many of those that can be used for testing .NET and Silverlight should be possible to use for the projects generated by Lightswitch.
7.2 Looking back at the implementation

In this chapter the rating of the different development platforms produced in Chapter 6 is summarized together with the time used for the different implementations. Table 4 contains the rating based on the experience with the basic application while Table 5 is concerned with authorization and internationalization. The tables contains both the time required for different activities where a 1 means that framework was the fastest while a 2 means it was the next fastest and so on. As for the rating of the functionality a 1 means that the framework was considered the best in that specific area.

**Table 4 Rating of the development environments based on experience with the basic application**

<table>
<thead>
<tr>
<th></th>
<th>ADF</th>
<th>Silverlight</th>
<th>Lightswitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time required</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>End user experience</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Developer experience</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDE</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Debugging</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Documentation</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Ease of use</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Simple applications (productivity in Chapter 6.2)</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Larger applications (maintainability in Chapter 6.2)</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; place</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; place</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt; place</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 5 Rating of the development environments in the areas of authorization and internationalization**

<table>
<thead>
<tr>
<th></th>
<th>ADF</th>
<th>Silverlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication and Authorization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Time Required</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>- Rating</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Internationalization and localization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Time Required</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>- Rating</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; place</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; place</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

- 80 -
7.3 Final recommendation
The recommendation is to use Silverlight and WCF RIA Services as the foundation for future development. It did not only perform best in the areas compared (Table 4 and Table 5) but does also best fit the initial requirements. The other environments do not fulfill the wish for ease of debugging and fails with either ease of use or internationalization support.

Silverlight + WCF RIA Services is the alternative that provides the best end user experience, allowing responsive application that behaves much like desktop application, as well as the best developer experience. It has a solid user base and a powerful platform that was the only one where all the desired features could easily be implemented.

It is believed that adhering to MVVM while developing the application will lead to the most easily tested and maintainable application possible with any of the frameworks.

7.4 Some notes on the recommendation
While the evaluation and recommendation is quite general, it does target CRM Treasury Systems. The recommendation should however apply to any similar company, which develops somewhat larger applications which go beyond very basic CRUD applications and with no major investment in either Java or the .NET platform.

If the evaluation had targeted a somewhat simpler application where multilingual interfaces were not a major requirement it might very well have been Lightswitch that would have been recommended. An eye should probably be kept on Lightswitch, as it did provide very fast development and integrated with a number of data sources including WCF RIA Services.

If a company already has large investments and expertise in the area of Java EE and JSF, but none in .NET, then Oracle’s ADF framework will most probably be the most compelling alternative for that company as it would probably allow easier integration with existing systems as well as allow parts of existing expertise to be utilized.
8 Conclusion

During the work with this thesis a small number of alternatives were selected for evaluation. These environments were then successfully used to create the basic application in each of them.

After the initial application the evaluation continued with the two largest frameworks and a number of authorization and internationalization scenarios were successfully implemented with both frameworks, except with some minor features missing.

Based on the experience from the implementations both quantitative as well as qualitative comparisons could be performed between the development platforms. Based on the results of the comparisons the Silverlight + WCF RIA Services platform is recommended as the best option for future development.
Bibliography


http://download.oracle.com/docs/cd/E12839_01/web.1111/b31973/af_global.htm#BJECDDDE.


Appendix A – Basic application, initial sketch

Figure 24 Startup screen

Figure 25 Choose selection screen
Figure 26 Transaction screen

Figure 27 Settings screen
Figure 28 Text and a button in Code example 3 as shown by the JDeveloper designer.
Figure 29 The page in Figure 28 as displayed in a web browser
Figure 30 DataControls with generated collections and operations for the ADF application
Figure 31 Text and a button in Code example 4 as displayed by the Visual Studio designer

Figure 32 Text and a button in Code example 4 as displayed when running the application
Choose the client and server topology for your application:

**Client**
- Desktop
  - This application runs on the user's desktop and can access other programs on the user's computer.
- Web
  - This application runs in the user's browser and will not be able to access other programs on the user's computer.

**Application Server**
- Run application services on the end user's machine.
- Host application services on an Internet Information Services (IIS) Server. A separate server is required.
- Host application services using Windows Azure.

Learn more about hosting using Windows Azure

Figure 33 Deployment alternatives for Lightswitch application
Figure 34 JDeveloper’s entity UI where authorization can be enabled for the entity
Figure 35 JDevelopers's "Select Text Resource" window

<table>
<thead>
<tr>
<th>Display Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Su</td>
<td>SJ</td>
</tr>
<tr>
<td>Save</td>
<td>SAVE_BUTTON</td>
</tr>
<tr>
<td>Selection</td>
<td>SELECTION_LABEL</td>
</tr>
<tr>
<td>Settings</td>
<td>SETTINGS_LABEL</td>
</tr>
<tr>
<td>Summary text</td>
<td>SUMMARY_TEXT</td>
</tr>
<tr>
<td>Submit</td>
<td>SUBMIT_BUTTON</td>
</tr>
</tbody>
</table>
Appendix C – Source Code

```java
/**
 * Set transaction status to 'confirmed'
 * @param transactions is list with id of the transactions to “confirm”
 */
public void confirmTransactions(List transactions) {

    //Check permissions here since no way of configuring it visually was found
    if (!ADFContext.getCurrent().getSecurityContext().isUserInRole("backoffice"))
        throw new AccessControlException("Not allowed");

    //Get view and create a new iterator to not affect the clients iterator
    RowSetIterator iter = getTransactionView1().createRowSetIterator(null);

    //Select one transaction at a time and get it
    for (int i = 0; i < transactions.size(); ++i) {
        int index = (Integer) transactions.get(i);
        iter.setCurrentRowAtRangeIndex(index);
        //Must cast to get row of correct type
        TransactionViewRowImpl row = (TransactionViewRowImpl) iter.getCurrentRow();

        //update the row
        row.setStatus("CONFIRMED");
    }
}
```

Code example 14 Server side function placed on ADF BC tier for marking transactions as allowed
public class transactionBean {
    private RichTable _transTable;
    private RowKeySet _rowKeySet;

    //Exposé TransTable property so that the table can bind to it,
    //allowing us to access below
    public void setTransTable(RichTable table) {
        this._transTable = table;
    }
    public RichTable getTransTable() {
        return this._transTable;
    }

    //Exposé Selection property so that the table can bind the currently selected
    //rows to it

    //Get the list of selected transactions in the table
    private List getSelectedTransactions() {
        List ids = new ArrayList();
        RichTable table = getTransTable();
        BindingContext ctx = BindingContext.getCurrent();

        //Get a reference to an iterator, so we can access the selected data
        Iterator selectedTransIt = table.getSelectedRowKeys().iterator();
        DCBindingContainer bindings =
            (DCBindingContainer)ctx.getCurrentBindingsEntry();
        DCIteratorBinding transIter =
            bindings.findIteratorBinding("TransactionView1Iterator");

        //Add the row indices of all selected transactions to the list
        while (selectedTransIt.hasNext()) {
            List keys = (List)selectedTransIt.next();
            Key key = (Key)keys.get(0);

            transIter.setCurrentRowWithKey(key.toStringFormat(true));
            ids.add(transIter.getCurrentRowIndexInRange());
        }
        return ids;
    }

    //Function called when confirm button is pressed, get selected transactions
    //and pass their row numbers to server side confirm function
    public String confirmAction() {
        List ids = getSelectedTransactions();

        //Get iterator so we can get application module
        DCBindingContainer bindings = (DCBindingContainer)BindingContext.getCurrent().
            getCurrentBindingsEntry();
        DCIteratorBinding transIter =
            bindings.findIteratorBinding("TransactionView1Iterator");
        DCDataControl control = transIter.getDataControl();
        ApplicationModule am = (ApplicationModule)control.getDataProvider();

        //Cast to the applications specific type defined in the ADF BC project which
        //allow us to call the confirmTransactions function
        AppModule app = (AppModule)am;
        app.confirmTransactions(ids);

        return null; //No action specified for the taskflow
    }
}
Code example 16 Client side code calling custom update function “ConfirmTransaction” for all selected items of a data-grid in Silverlight

```csharp
/// Mark transactions as confirmed
private void confirmTransactions_Click(object sender, RoutedEventArgs e)
{
    foreach (Web.Transaction trans in transactionDataGrid.SelectedItems)
    {
        trans.ConfirmTransaction();
    }
    this.transactionDomainDataSource.SubmitChanges();
}
```

Code example 17 A WCF RIA Services “custom update” method for marking transactions as confirmed in the Silverlight prototype

```csharp
[Update(UsingCustomMethod=true)]
[RequiresRole("backoffice")]  //Limits access to backoffice members
public void ConfirmTransaction(Transaction currentTransaction)
{
    if (currentTransaction EntityState == EntityState.Detached)
    {
        this.ObjectContext.Transactions.Attach(currentTransaction);
    }
    currentTransaction.STATUS = "Confirmed";
}
```

Code example 18 WCF RIA code for updating transactions with authorization checks added

```csharp
[RequiresRole("admin", "backoffice")]
public void UpdateTransactions(Transaction currentTransactions)
{
    //Only allow admins to update
    if (!ServiceContext.User.IsInRole("admin")
    return;

    //Auto generated code:
    this.ObjectContext.Transactions.AttachAsModified(currentTransactions,
                                                       this.ChangeSet.GetOriginal(currentTransactions));
}
```
public class MustBeOwnerAttribute : AuthorizationAttribute
{
    // Called whenever a function marked with [MustBeOwner] is called
    protected override AuthorizationResult IsAuthorized(IPrincipal principal, AuthorizationContext authorizationContext)
    {
        // The entity being validated (update, insert or delete)
        Selection argument = (Selection)authorizationContext.Instance;

        // Allow admins to change Selection without being the owner
        if (principal.IsInRole(Roles.Admin))
            return AuthorizationResult.Allowed;

        // Get owner information from the database and verify the user owns it
        using (var db = new LSSonline.Web.Silverlight2Entities())
        {
            string owner = (from Selection s in db.Selections
                            where argument.ID == s.ID
                            select s.OWNER).FirstOrDefault();

            if (owner == principal.Identity.Name)
                return AuthorizationResult.Allowed;
            else
                return new AuthorizationResult("Can’t modify others Selections.");
        }
    }
}

Code example 19 Implementation of a custom authorization annotation for use with WCF RIA Services

... // The Name displayed will be the string defined with the key “PasswordLabel” // in the registration-data resource file.
[Display(Name = "PasswordLabel", ResourceType = typeof(RegistrationDataResources))]
public string PasswordConfirmation { ... }
...

Code example 20 Use of the Display attribute to configure the label for a property

public DateTime? DealDate
{
    get
    {
        // Check if we had a value stored in the database
        if (DEAL_DATE.HasValue) // Create a new date time with same time set to UTC
            return new DateTime(DEAL_DATE.Value.Ticks, DateTimeKind.Utc).ToLocalTime();
        else
            return null; // No value in database so continue to propagate null
    }
}

Code example 21 Code used to convert unspecified DateTime from UTC to local time

- xiii -