CHALMERS

Automated IT Processes

Architectural design proposal for supporting automated IT processes in a Microsoft environment

Master of Science Thesis in the Programme Software Engineering and Technology

MARCUS MELBERG

MARKUS WESTERSTRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, July 2009

Automated IT Processes
Architectural design proposal for supporting automated IT processes

MARCUS MELBERG
MARKUS WESTERSTRÖM

Examiner: JOACHIM VON HACHT

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

Department of Computer Science and Engineering
Göteborg, Sweden, July 2009

Abstract

This master thesis defines and evaluates a system architecture proposal that will support automated IT processes systems in a Microsoft environment. The architecture is based upon programming models in the .NET framework and will enable systems to be integrated with Microsoft's Identity Lifecycle Manager (ILM). It is intended to be adaptable to changes and provide strength to the integration of the different programming models.

The architecture has been evaluated by a prototype that has been built using Windows Presentation Foundation and Windows Communication Foundation. Windows Workflow Foundation has also been used to define workflows that are hosted by ILM.

The result is an architectural design that provides scalability and a separation of concerns. The architecture can be seen as a structural framework that provides stability for further development. The prototype can be seen as the realization of this architecture, with features and functionalities that are common in applications for automated IT processes.

The most important question that this thesis aimed to answer; is it possible to use the .NET framework and ILM to build applications for automated IT processes, was answered by the result of the prototype. ILM constrains the infrastructure and the architecture but provides strength to important attributes such as maintainability and extendibility. The combination of ILM and the .NET programming models in this master thesis has been proven successful for automated IT processes.

Keywords: ILM, .NET, WPF, WCF, WF, Silverlight, System Architecture, C#, Automatization, Infrastructure.

Sammanfattning

Denna examensrapport definierar och evaluerar ett arkitekturförslag för automatiserade IT-processer i Windows miljö. Arkitekturen baseras på programmeringsmodeller i .NET ramverket och kommer möjligöra att system integreras med Microsoft's Identitiy Lifecycle Manager (ILM). Arkitekturen syftar till att göra det tänkta systemet anpassningsbart och underlätta integration mellan de olika programmeringsmodellerna.

Den framtagna arkitekturen har evaluerats med hjälp av en prototyp. Prototypen har utvecklats i Windows Presentation Foundation och Windows Communication Foundation. Windows Workflow Foundation har också använts för att utveckla arbetsflöden som har integrerats i ILM.

Resultat är en arkitektur som tillhandahåller skalbarhet och en tydlig separation mellan olika komponenter i systemet. Arkitekturen kan ses som ett strukturellt ramverk som underlättar för framtida utveckling. Prototypen som har utvecklats är en realisering av arkitekturen som innehåller funktionalitet som är vanligt förekommande i applikationer för automatiserade IT processer.

Den viktigaste frågan som detta examensarbete syftar till att svarar på; är det möjligt att använda .NET ramverket och ILM tillsammans för att bygga applikationer för automatiserade IT processer. Denna fråga besvarades med hjälp av implementationen av prototypen. ILM begränsar infrastruktursmöjligheterna och arkitekturen, men bidrar samtidigt till goda underhåll- och utvecklingsmöjligheter. Detta examensarbete har visat att ILM och de programmeringsmodeller som finns i .NET är lämpliga att använda för automatiserade IT processer.

Nyckelord: ILM, .NET, WPF, WCF, WF, Silverlight, Systemarkitektur, C#, Automatisering, Infrastruktur.

Preface

This is the report for the Master's Thesis done by Marcus Melberg and Markus Westerström at Chalmers University of Technology in the programme Software Engineering and Technology.

The Master Thesis subject was initialized and proposed by Atea/Spintop in Göteborg. Atea/Spintop is a company that is specialized at developing applications for automated IT processes. Joel Sanderi has been our technical supervisor at Atea/Spintop.

We would like to thank all the people at Atea/Spintop in Göteborg for all their help and support during the thesis work. We would especially like to thank Joel Sanderi for all the effort he has put into our work.

Many thanks to Joachim von Hacht, who has been our supervisor at Chalmers, for all the support and guidance during the thesis work and his feedback on the thesis report.

Table of Contents

Table of Figures

1 Introduction

This section will provide a background description of the problem domain that this thesis will investigate. The purpose of the thesis will be presented and the limitations that have been made, based on the purpose, will be explained.

1.1 Background

During the last decade many organizations have gone through an extensive reformation and automatization with help of *information technology* (IT) systems. An exception from this is the processes performed by companies IT divisions that may not have gone through such an extensive reformation or automatization. In this thesis report IT processes refer to the everyday task that the service desk department or IT divisions in a company performs, such as user registration, create mailbox account and distribute software.

The work performed by IT divisions is to a great degree about access management and identity management. The demands and the complexity of these management forms increase rapidly today. Examples of demands are:

- Financial benefits
- Security
- Information synchronization
- Efficiency
- Accessibility
- System integration

The IT infrastructure is an important aspect for any medium or large company to consider. It must be efficient, cost effective and secure. In many cases it does not deliver as much business benefits as it could. Inefficient identity management can have many impacts. Manual identity management becomes costly since users cannot manage their own identities. Instead these requests must be handled by a service desk or IT department which will lower the productivity for both the user and the IT department. [***1***]

Security is an important aspect that identity management must facilitate and enhance. For example, it must assure that only authorized users can access business resources to avoid intellectual property leakage. In the same way, identity management needs to have secure processes to grant access to users or employees that need permission to reach certain resources efficiently, in order to do their job and to avoid productivity loss. [***1***]

Within larger organizations information is often stored in many different data sources (email systems, directory databases etc.) and in many different formats. Keeping all these data sources synchronized with each other is a complex and time consuming task. Ignoring to handle these issues may result in inconsistent data when different systems will contain diverse information.

Automatization of an IT division's task often requires complex business scenarios which can have a lifetime that span over days and needs to be maintained in an automated way. An example is tasks that need approval and confirmation from different parties.

When developing applications for automated IT processes in companies there is often high demands on the *Graphical User Interface* (GUI). The GUI needs to display complex data and its relationship in a clear and intuitive way. A user with little or no technical skill should be able to access an application easily and use it in an efficient way.

Finally, all different systems that exist within an organization need a way to be integrated with each other. Problems quickly arise when these systems require different protocols that have diverse requirements when it comes to security, authentication and authorization.

Microsoft has developed different products to enhance this type of automatization in a Windows environment. An example is the *Identity Lifecycle Manager "2"* (ILM) that facilitates access management, identity management and provides a synchronization engine that can assure that changes made in one data source will be propagated to all the others.

Microsoft has also developed the extensive .NET framework, which supports building and running applications. This framework also contains different models and development technologies for various implementation tasks. Some of these models can facilitate the development of automated IT process.

Three important models are:

- *Windows Workflow Foundation* (WF)
- *Windows Presentation Foundation* (WPF)
- *Windows Communication Foundation* (WCF)

WF is a model in the .NET framework that can facilitate the implementation of business scenarios. WF provides a way to create and visualize complex processes in a lucidly way and maintain these processes during their lifetime.

WPF is a part of the .NET framework that can be useful to develop graphical user interfaces. It introduces a programming model with a clear separation between the graphical representation and the business logic behind. WPF provides features that make it possible to create intuitive and rich user interfaces.

WCF can be seen as the glue that can bind different communication protocols together, by introducing a common model for handling communication between systems (through services) in a secure and reliable way. By hiding much of the complex and detailed logic involved with communication the problems regarding different protocols can be addressed in a flexible and maintainable way.

ILM and the .NET framework will be explained more in section 3 Technologies.

1.2 Purpose

The thesis work aims to investigate and evaluate an architecture proposal for a system that will be developed with WPF, WCF and WF and integrated with ILM to develop applications for automated IT processes.

The purpose is to analyze and develop an architectural design, for automated IT processes, with the above mentioned technologies where ILM will be a central part.

Important parts of the architecture that needs to be investigated are:

- How the communication between WPF and WCF can be implemented.
- The most suitable way of using WPF with this type of communication.
- How ILM can be used as a synchronization service for different data sources and as a host process for WF.

The architecture will be analyzed regarding scalability, testability and if it can provide a distinct separation between the graphical user interface and the business logic.

1.3 Abbreviation/Definitions

Windows Presentation Foundation	WPF
Windows Communication Foundation	WCF
Windows Workflow Foundation	WF
Identity Lifecycle Manager	ILM
Extensible Markup Language	XML
Extensible Application Markup Language	XAML
Graphical User Interface	GUI
Data Access Layer	DAL
Active Directory	AD
Simple Object Access Protocol	SOAP
Software Requirement Specification	SRS
Application Programming Interface	API

1.4 Limitations/Delimitations

The testability evaluation will not take into consideration testability and usability of any graphical elements.

Evaluation regarding performance of the prototype will be limited, due to resource limitations.

The security of the prototype will not be verified to work in a production environment.

The architecture is based upon a specific infrastructure, and the thesis work will not investigate different infrastructure solutions.

2 Method

Many aspects of an architecture may not be fully evaluated without actually using it in a real context. For example, many strengths and weaknesses can be found when functionality is added or an external system appears that may require the architecture to adapt.

Therefore a prototype will be developed and used to empirically evaluate the new architecture proposal. The prototype will be a web application providing two major features. These features will provide meaningful insight on how functionality can be implemented using the new architecture and the related frameworks. The prototype will also provide input to an evaluation of the architecture regarding testability, scalability and if the user interface can be separated from the business logic.

The two major features that the prototype will contain are:

- *Register user*.
- *Distribute software*.

Register user aims to provide functionality to create a new user while *Distribute software* is intended to handle installation of new software to specific users or computers. Both of these features have been chosen because they reflect typical usage scenarios and are often requested when automating IT processes.

Register user exists as a similar service in the ASP.NET web portal that comes with ILM, but instead of using this service a new one will be created that takes advantage of WPF to enrich the graphical user interface. This way the application may provide better usability because of the many features that comes with WPF. This new service will not use any parts of the ILM ASP.NET web portal. The primary purpose of the *Register user* feature is to evaluate how the communication with ILM can be achieved in a satisfying way.

Distribute software is interesting since the technical process that runs in the background is rather advanced and especially since there cannot be any demands on the end users technical skills. The software distribution requires special configuration, the purpose is to investigate if the architecture can adapt to these changes. Further, the ILM-portal does not contain any similar service (compared to the *Register user*) which makes it possible to evaluate how completely new functionality can be added and supported.

Even though *Register user* will be implemented as a new service, the ASP.NET web portal with ILM has proven that this functionality can be added, which is not the case for *Distribute software*.

Further explanation of the functionality in the prototype can be found in the SRS (see Appendix A).

3 Technologies

This section contains information about the different technologies and frameworks that have been used in the project. The technologies are described to better understand the project and do not contain any extended information.

3.1 Microsoft .NET Framework

The .NET framework is a Microsoft Windows component that supports building and running applications and XML web services. The .NET framework is developed by Microsoft and the first version was released in the middle of 2000. The 3.5 version, which will be used in this project, was released in the end of 2007. There are two main components in the .NET framework; the common language runtime and the .NET framework base class library. [***2***]

The Common Language Runtime (CLR) is an implementation of the Common Language Infrastructure (CLI) standard done by Microsoft. CLR is a language independent environment for execution of program code, which also manage low-level details where memory management, thread execution and security checks are some examples. [***3***]

The .NET base class library is available for programming languages and encapsulates primitives, handling input/output (I/O), graphical rendering and takes care of interaction with different hardware devices. In addition to this it also enables a number of services, for example database access and handling of XML documents. [***4***]

All programming languages that are compatible with CLR can be used when developing applications in .NET. The most widely used languages in .NET are C# and Visual Basic but it also exist .NET compilers for Pascal, Smalltalk and a number of other programming languages. [***4***]

C# is a rather new programming language which was introduced by Microsoft with the first release of .NET. C# derives in some sense from many programming languages, high performance from C and object-oriented structure from C++. C# also contains garbage collection similar to Java and it has some influences from Visual Basic. C# provides support for structured, component-based and object-oriented programming which is naturally for a language that is based upon C++ and Java. [***5***] [***2***]

The .NET 3.5 framework contains different models, which are described in Figure 1 below. All of the development technologies in .NET, for example ASP.NET or WPF, can use C# for the programming logic. Both web applications and desktop applications can be based upon the C# language. [***4***]

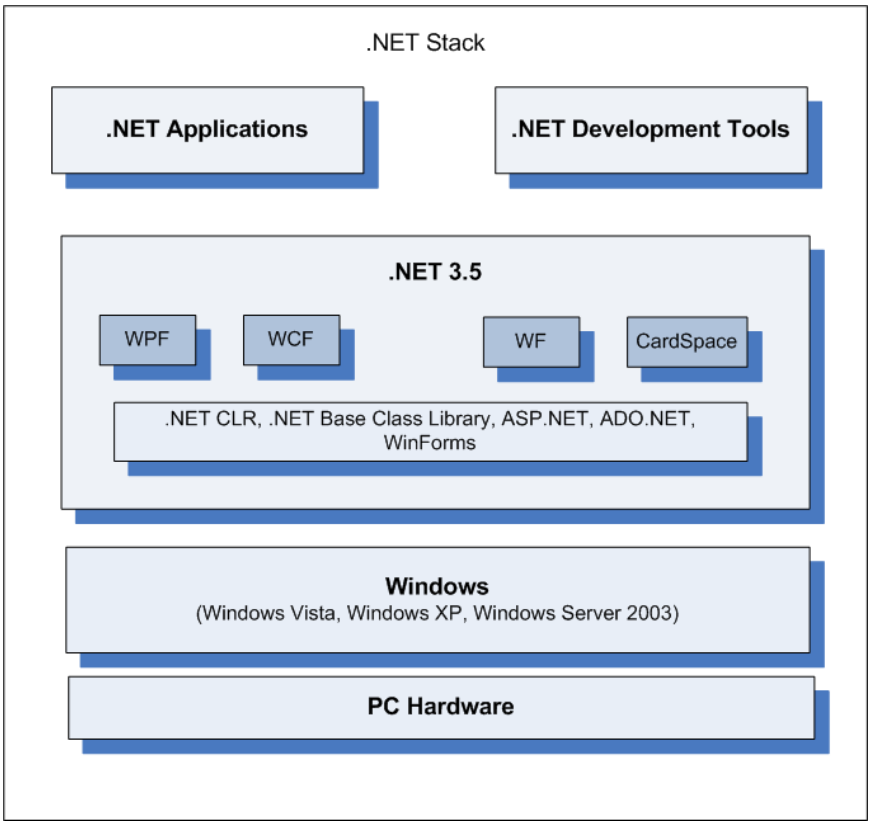

FIGURE 1: THE .NET STACK

3.1.1 Windows Communication Foundation

Windows Communication Foundation (WCF) combines and unifies all the communication programming models available in .NET 2.0 into a single model. It is based upon a service-oriented architecture in order to support distributed computing, where services are consumed by consumers. Before WCF was introduced a technology choice had to be made from the beginning since it could have an impact on the architecture for the application. [***6***]

ASP.NET Web Services (ASMX), Web *Service Enhancement* (WSE) extensions, the *Microsoft Message Queue* (MSMQ), the *Enterprise Services/COM+* runtime environment and *.NET Remoting* are technologies that are now collected in WCF. The result is that developers will not need to make such technology choices upfront and it enables the possibility to implement a solution that combines different requirements on a single technology platform. [***6***]

Three fundamental parts need to be specified in order to define a WCF service, called the ABC.

- Address
- Binding
- Contract

The address defines where it is available, the binding defines how the service can be accessed and the contract specifies what the service can do. The address, the binding and

the contract are encapsulated in an endpoint, which becomes a gateway that contains necessary information for clients that will use the service. [***7***]

It is possible to use three different contracts in WCF; the first type is the *Service contract* and can be seen in Code 1. The *Service contract* specifies what the service can do i.e. what functionality that is available. [***7***]

```
[ServiceContract]
public interface IUserService {

  [OperationContract]
  Person GetPerson(String objectID);

}
```

CODE 1: A SERVICE CONTRACT.

The second type is the *Data contract* which defines a data type that can be used with the service. An example of a *Data contract* can be seen in Code 2. [***7***]

```
[DataContract]
public interface Person {

  [DataMember]
  public String ObjectID { get; set; }

  [DataMember]
  public String DisplayName { get; set; }

}
```

CODE 2: A DATA CONTRACT.

The third type is the *Message contract* which allows the formatting of different messages to be controlled. An example could be to define how different data types should be serialized and inserted into a message. [***7***]

With the structure of WCF it is possible to separate the actual work the service performs from how the communication is controlled. This makes it possible to change how the service communicates without affecting what work it actually performs. [***7***]

The configuration of how a service should communicate with its clients (for example protocol and security) can either be done programmatically or by using a configuration file. [***7***]

3.1.2 Windows Workflow Foundation

Windows Workflow Foundation (WF) is used to create and define workflows in applications. A workflow defines all necessary steps that are needed to complete a certain task or work. Steps could for example be decision points. Many workflows have different branches which imply that the workflow will have alternative execution paths. [***8***]

One type of workflow is a sequential workflow. It is a workflow that can have branches but all operations will be performed in a sequential order. Sequential programming often refers to programming without branches but when related to workflows it means continuous flow. A sequential workflow should be used if the process is simple and there is no need for external inputs. Otherwise, one should consider using state-based workflows. A state-based workflow has a criterion for each step that must be fulfilled before the flow can continue to execute. Both workflow types are available in WF. [***8***]

WF defines a new kind of model that clearly separates what it will do and when it will be done. Business logic, what the workflow does, can be encapsulated inside discrete components or activities and the rules that control the flow of execution can be described declarative. With this clear separation it becomes easier to define when something will happen without affecting how it will be done. [***8***]

On top of this model, WF enables an editor to graphically display a workflow and make changes to it. The combination of the declarative rule model, business logic encapsulation and visualization capabilities enable workflows to be constructed and changed with little or no coding at all. [***9***]

3.1.3 Windows Presentation Foundation

One of the drawbacks with current technologies for creating Windows applications, such as Windows Forms, is that it is essentially based upon display technology that was introduced 15 years ago. Even though the technology has matured during the years it was not designed to provide many of the functionalities that are requested today where 3D rendering, animations and advanced layouts are some examples. Even though some of these functionalities can be provided it comes with a cost, both in developing time and complexity. [***10***]

Windows Presentation Foundation (WPF) aims to meet the new requirements of the next generations GUI's by providing its own graphical display system. This new display system is based around the DirectX libraries [***11***] (instead of GDI/GDI+ and User32), which have much better rendering capabilities, can provide more features and better performance. Older technologies get the appearance of standard interface elements, such as text boxes, check boxes etc. from the Windows API. This makes the elements difficult to customize but since WPF is based around its own display system and does not rely on the Windows API, this problem is solved. [***10***]

When developing windows applications using Windows Forms you create and design each page programmatically using either VB or C# [***12***]. In WPF a different approached is used which is based on a declarative XML markup language called *Extensible Application Markup Language* (XAML), see example in Code 3. A GUI in WPF is defined by a set of declarative commands (in XAML) that is later transformed by the WPF runtime engine into something that can be displayed to the user. [***13***]

```
<UserControl Width="500" Height="450">
  <Canvas>
    <TextBlock x:Name="ObjectIDText" Text="ObjectID" />
    <TextBox x:Name="addObjectID" />
  </Canvas>
</UserControl>
```

CODE 3: A XAML EXAMPLE.

3.2 Identity Lifecycle Manager "2"

Microsoft's *Identity Lifecycle Manager "2"* (ILM) is a solution for identity management and access management.

> *"The Microsoft strategy for identity management delivers a comprehensive solution to manage identities, credentials and identity-based access policies across heterogeneous environments".* [***1***]

ILM supports management of user identities from creation and throughout the whole lifecycle. It provides identity synchronization between different data sources, password management and user provisioning in an environment that works across Windows and heterogeneous systems. This provides possibilities for IT organizations to automate the processes for handling users and identities. [***1***]

ILM is based upon a common set of services, workflows, web service API's and logging which enable customization of the way of working with ILM and the ability for companies and vendors to extend the functionality in ILM. [***1***]

Figure 2 illustrates how ILM could be used in a company; it handles the synchronization between the directory database, the Human Resource (HR) database and e-mail systems. It enables the possibility to ease the work performed by service desks in organizations, for example provide functionality to end-users to reset their login password by themselves. [***1***]

ILM can be used to synchronize e-mail address lists that are maintained by different systems, for example Microsoft's *Exchange Server*, and also synchronize multiple directory databases into one address list. This facilitates identity management and result in better control of the IT structure in a company. [***1***]

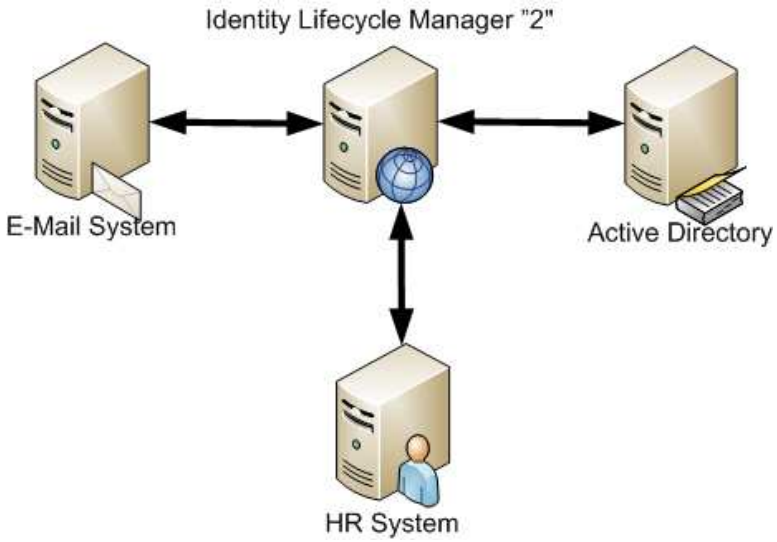

FIGURE 2: ILM SYNCHRONIZATION ENVIRONMENT

Another important feature in ILM is the user provisioning that provides functionality to define policies that automatically create user accounts, e-mail addresses and group permissions. In practice this means that a company only needs to register a new employee at a single system and it will provision the user into all other systems that are necessary for the employee to become productive immediately. [***1***]

Example: *If a new employee starts working at a company, he/she needs an account name to login to the network, an e-mail account and maybe become registered into the human resources database. To synchronize all databases one could use ILM.*

4 Tools

During the implementation of the prototype a number of different tools have been used to simplify development and collaboration within the project.

4.1 Development environment (IDE)

Visual Studio is a development environment that makes it easier to work with many of the different technologies and frameworks that are provided by Microsoft. Many useful tools to develop an application are supported in *Visual Studio* such as debugger and database visualization. [***14***] All implementation was made in *Visual Studio 2008* as well as the construction of workflows, defining web services and testing.

4.2 Collaboration

By using software that simplifies collaboration more time could be spent on solving implementation related problems instead of organizational.

4.2.1 Microsoft SharePoint

SharePoint from Microsoft has been used as a collaboration tool during the project. It is a browser-based tool which enables features like calendar for the project, discussion pane, bug tracking and task lists. It is also used to manage documents and collect all related files for the project at a single place. This collaboration tool has been used in order to work more efficient and maintain a structured way of working within the project.

Functional requirements were extracted from the *System Requirement Specification* (SRS), the SRS can be found in Appendix A, and inserted into a task list at the *SharePoint* site which provided the possibility to assign project members to different tasks. This made it easier to track which team member that had implemented a specific functionality in case any problem occurred in the code later on. It also provided a good overview of the project and a good insight about the progress of the project.

4.2.2 Version Control (Visual SourceSafe)

To make it possible to work many developers on the same project, a version control system is often used. The version control system keeps track of changes on files and helps solve conflicts when many people are working on the same file [***15***].

It exist many different version control software but the one that was used was *Visual SourceSafe* from Microsoft. Similar to many other version control software *SourceSafe* provides functionality such as backtracking of file changes and merging two versions of the same file [***16***].

5 System Architecture

In order to describe the architecture of the intended system, this section will first describe the infrastructure for the prototype. Due to the fact that the architecture design for ILM is fixed, a lot of emphasis will be put on the architecture for the prototype.

5.1 Infrastructure

A computer network at a company is built upon different physical components (nodes), which can be workstations, servers, printers and other types of resources. The network also has users (employees in the company). A logical grouping of these elements is called a domain in a Microsoft network. [***17***]

The domain could be seen as an administrative boundary and within this it is possible to control security for computers and users from a domain controller. A domain also shares a common directory database and in a Microsoft network this database is *Active Directory* (AD). This directory database contains security information and settings for the resources in the domain. AD also stores all user accounts and groups that are being used in the domain. [***17***]

The internal network in an organization is called intranet, which basically means that the network is only available for authorized users. An intranet at a company is often used to store and collect information for employees, for example at an internal website. An intranet is built upon common resources and databases for an organization.

ILM is developed to operate in a similar environment as the one described above and this type infrastructure is a prerequisite for ILM. [***1***]

5.2 Evaluation infrastructure

The infrastructure or the test environment, in which the prototype will be evaluated, is similar to the environment described in section 5.1. It will have two servers that run Microsoft Server 2008. One server (see server A in Figure 3) will act as the domain controller, and will therefore contain *Active Directory* (AD). The domain controller manages security aspects between users and domain interactions, i.e. centralizing administrations.

Server A in Figure 3 will host a *Microsoft Exchange Server* to provide all users with an e-mail account. The same server will act as file server to provide the ability to add functionality such as home directories for users and storage for applications that are distributed.

The key part in this infrastructure is ILM that will be hosted by server A. Figure 3 illustrates how the intended infrastructure will look, where the workstation is a client that is a member of the domain and can communicate with the prototype through a web browser.

Server B in Figure 3 will be used to host the prototype in its *Internet Information Services* (IIS) [***18***]. This server will also be a member of the same domain in order to use Windows

authentication for the prototype. Windows authentication means that the authentication to access the prototype can be based around the users in AD.

Example: *A user can login to the domain at a local workstation, and he/she will have access to the prototype if the workstation is a member of the same domain. If a user would like to reach the prototype from a computer outside the domain he/she will have to enter his/her credentials in the web browser, when trying to reach the prototype.*

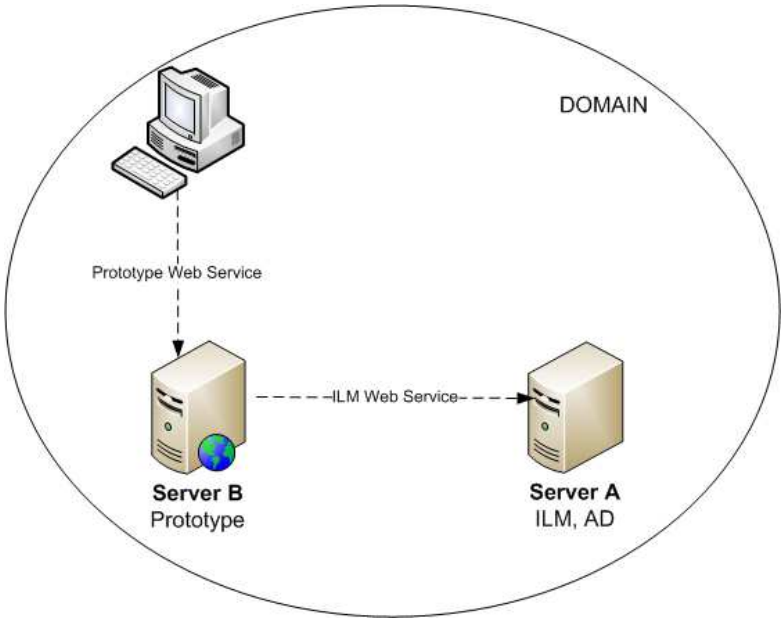

FIGURE 3: NETWORK INFRASTRACTURE

5.3 ILM Architecture

Figure 4 illustrates the high level architecture of ILM and all arrows indicate the data flow. The center of the ILM architecture is the synchronization engine; see *ILM Sync* in Figure 4. The synchronization engine is connected to different databases and directories through management agents. In Figure 4 these are called *Adapters*.

ILM is a product that combines different management tools into one. ILM contains solutions for user-, group-, policy- and credential management. These are illustrated in Figure 4 in the *Solutions* tab. The intention is that ILM should work as a self-service for employees in a company, by using clients for these solutions. The portal is an administrative ASP.NET web portal that comes with ILM in which it is possible to manage users, groups etc. It is possible to develop customized clients for ILM that might contain additional functionality compared to the portal that comes with ILM.

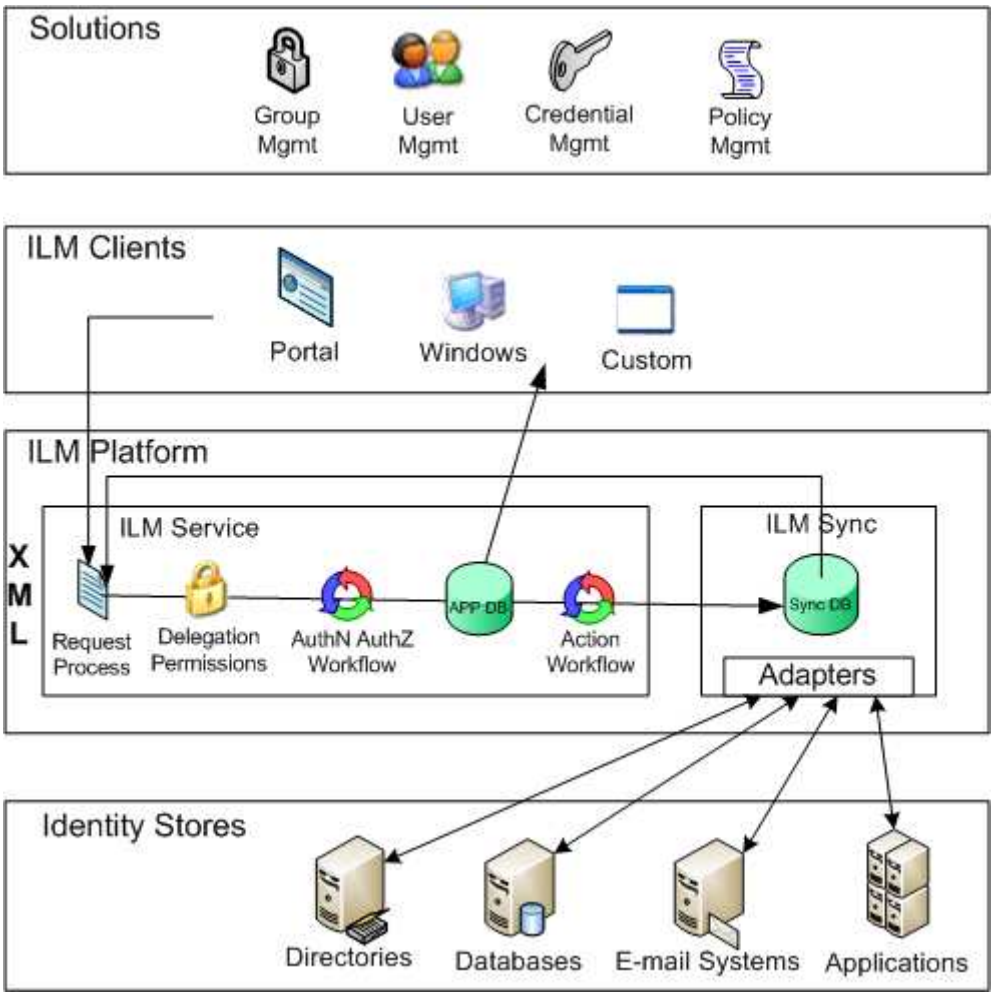

FIGURE 4: ARCHITECTURE OF ILM [*19*]

ILM uses an internal database, called *APP DB* inside *ILM Service* in Figure 4, to store all data. All ILM clients that need to access the database must go through the ILM service, even the synchronization engine needs to use the ILM service in order to write to the database. A result from a request will be returned to the client at the *APP DB*. Requests can also trigger workflows and clients can also store data which will be propagated to the *Sync DB*. If a request should trigger a workflow or not is defined by policies that states certain criteria. How and if data should be synchronized with ILM and other systems is defined by in- and outbound synchronization rules. [***19***]

ILM defines all stored data as different object types, for example *Persons* and *Groups*. It is also possible to create new object types in ILM, for example a new *Software* object. [***19***]

Example: *If a new user will be registered it will be done through one of the ILM clients. A XML request will be sent to ILM service from the client. The request will first go through the delegation permissions and then through the authentication and authorization workflows. If the request pass these steps the user till be stored in the APP DB, otherwise a fault message will be sent back to the client. When the new user is stored in the APP DB it might trigger a workflow, for example an approval workflow. This is defined by policies that describes when*

and how a workflow should be triggered. The new user could also be propagated to the Sync DB if the user fulfills certain criteria that are defined in a synchronization outbound rule. An outbound rule could be if the user is of employee type Contractor, then the user should be synchronized with the e-mail system.

The ILM service is a WCF service which exposes functionality that is available for object types such as put, get and enumerate. Exposed methods are called by creating *Simple Object Access Protocol* (SOAP) messages, also referred to as XML messages[1] [***20***]. These messages will contain details about what action should be performed and what data should be used. It is possible to choose which attributes one wish to get for a specific object type when requesting data from the *APP DB*. This is defined in the get-message. [***19***]

Code 4 illustrates a simplified SOAP message. This message will create a new *Person*, which is defined by the ObjectType attribute.

```
<Envelope>
    <Header>
<Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Create</Action>
    </Header>
    <Body>
        <AddRequest>
            <AttributeTypeAndValue>
                <AttributeType>ObjectType</AttributeType>
                <AttributeValue>
                    <ObjectType>Person</ObjectType>
                </AttributeValue>
            </AttributeTypeAndValue>
            <AttributeTypeAndValue>
                <AttributeType>DisplayName</AttributeType>
                <AttributeValue>
                    <DisplayName>Stefan</DisplayName>
                </AttributeValue>
            </AttributeTypeAndValue>
            <AttributeTypeAndValue>
                <AttributeType>Department</AttributeType>
                <AttributeValue>
                    <Department>Accounting</Department>
                </AttributeValue>
            </AttributeTypeAndValue>
        </AddRequest>
    </Body>
</Envelope>
```

CODE 4: SOAP MESSAGE

Object types can have two different types of attributes in ILM: single value and multi value. Single values represent a specific value such as *Boolean* or *String* but could also represent a reference to another object type. A multi value attribute is basically the same as a single valued but can hold many values or references, similar to an array. [***21***]

[1] No separation between SOAP messages and XML messages will be made in this report.

5.4 Prototype architecture

The main requirement for the prototype architecture was to separate all different components (layers, services) to achieve a clear separation of concerns. The separation will improve scalability since all components will be created with fewer dependencies and the possibilities to replace or extend a component will be greater.

ILM will probably be extended with new object types and attributes. It could be other changes as well but the prototype needs to be designed so it can adapt to these changes with minimum implementation effort. To accomplish this, the architecture should support auto generated classes that are based upon an XML schema from ILM, which defines the object types.

The architecture for the prototype will be influenced by a layered architecture and a service oriented architecture to enhance separation between components [***22***]. It will not be a pure service oriented architecture nor a pure layered architecture but a combination of them. The layered architecture part is illustrated in Figure 5.

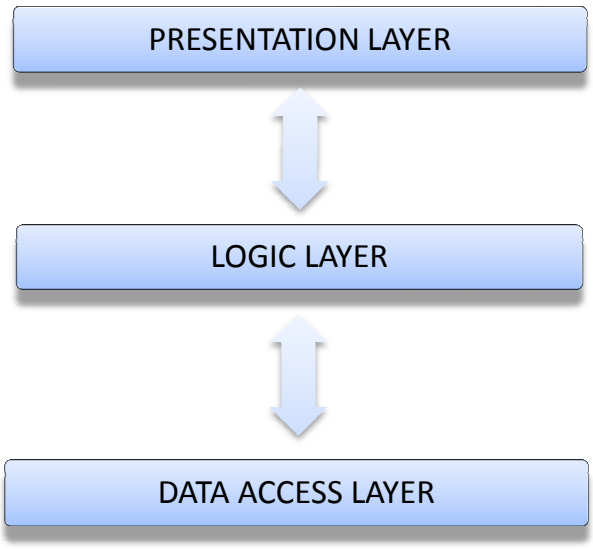

FIGURE 5: LAYERED ARCHITECTURE

The prototype will be dependent on the functionality in ILM, and because of this the communication between ILM and the prototype is important. This communication will be encapsulated in a *Data Access Layer* (DAL) that could be compared to the data layer in a layered architecture. This will provide ability to update or replace the DAL since requirements or functionality in ILM might change.

The messages sent through ILM web service are based upon XML and the prototype needs to transform data in these messages into an object oriented representation that can be used in C#. It will be much easier to work with objects within the prototype since the structure

and complexity of the XML requests and responses are hidden. This transformation should also be encapsulated in the DAL.

In order to provide flexibility to the architecture there exist a logic layer. This logic layer will contain all business logic that cannot be expressed in ILM. It will contain data validation of input from the presentation layer.

The graphical user interface will be encapsulated in a presentation layer. If there is not a clear separation between the presentation and the business logic it can cause problems when changes need to be made.

Example: If t*he presentation layer (client) needs to keep track of the order in which business logic should be executed and if the system is using different types of clients, then each client needs to be updated if there is a change in which order business logic needs to execute in.*

The DAL and the logic layer will be the server side component and the presentation layer will be the client side component of the prototype. The intended architecture for the prototype is illustrated in Figure 6. In Figure 6 the *Prototype WCF service* and the *web server* is the server component and the *WPF Client* is the client component of the prototype.

The prototype should support or enhance implementation of different types of clients, for example a web application and a desktop application. The client should also be as thin as possible which is supported by the layered architecture [***23***]. To achieve this, the logic layer and the presentation layer will be separated. The presentation layer will therefore only communicate with the logic layer through a WCF service, this could be compared to a service-oriented architecture [***24***]. This service will be referred to as the *Prototype WCF service*.

Other reasons for creating a WCF service between the client and the server of the prototype are to hide complexity and prevent direct communication between ILM and clients. All the complex transformation logic and message creation between ILM and the prototype will be handled on the server side, which will simplify creation of clients.

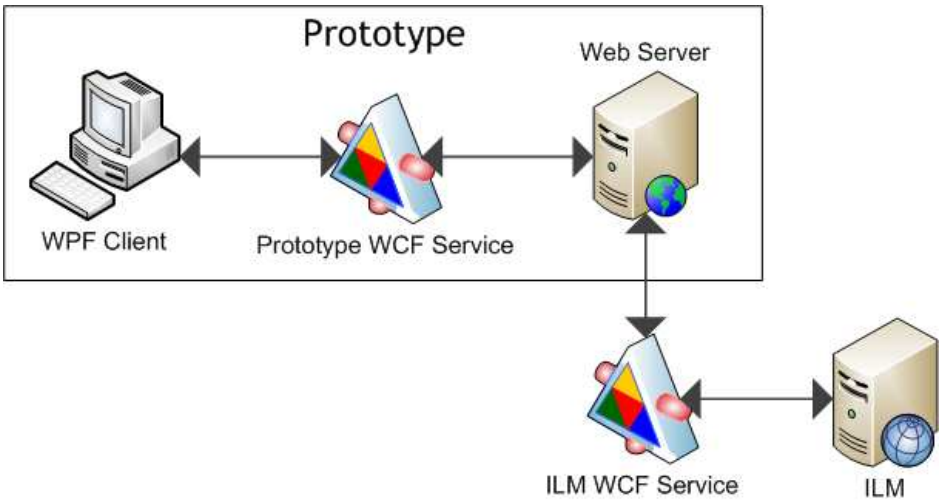

FIGURE 6: ARCHITECTURE OVERVIEW

The *Prototype WCF service*, see Figure 6, could be hosted on the same web server as the prototype to reduce complexity in the infrastructure. This service should contain more intuitive methods than ILM's put, get and enumerate (such as register user, delete user) and also use more convenient data types as parameters.

With the proposed architecture a distinct separation between all the components in the system can be made. All the components will be independent from each other and they will have the ability to easily be replaced or extended. In order to easily replace clients, they should only need to make one call to the server and the server should handle the process of triggering the right actions in the correct order.

6 System Design

The system design will be based on the prototype architecture and will investigate how the architecture can be implemented. The system design will be affected by the technology choices and the requirements from the architecture. This section will describe these requirements and design solutions for them.

6.1 WPF Technology Analysis

Choice of WPF technology has not been explicitly done (the architecture will support any technology). Therefore an analysis will be presented regarding which type that will be most appropriate to use.

Two different approaches can be used to create a web application in WPF; Silverlight or *XAML Browser Application* (XBAP). This analysis contains a brief overview of the two technologies and is based upon a comparison that was made in the beginning of the thesis work. The purpose of the comparison was to investigate which one that was most appropriate to use in the development of the prototype.

6.1.1 Silverlight

Silverlight applications are compiled and packaged for the web. There are two different options for hosting a Silverlight application, from an ordinary HTML page or through an ASP.NET web form. The result will be the same and the Silverlight code will run at the local client computer and not on the web server. [***25***]

Even though the different hosting approaches produce the same result, the ASP.NET hosting has some advantages. One advantage is that it enables websites to contain both ASP.NET and Silverlight content which gives developers better control of when and how to initialize the Silverlight content. Another advantage is that ASP.NET hosting can minimize the occurrence of cross-domain related problems when hosting both a Silverlight application and a web service. Cross-domain problems could occur if the web service is not hosted by the same server that distributes the Silverlight application. If no server-side code is needed there is less reasons to use an ASP.NET website as hosting approach. [***25***]

Silverlight requires that a plug-in is installed into the web browser and does not support older operating systems such as Windows 98 and Windows ME. Except that, Silverlight is supported by multiple browsers and runs for example in Internet Explorer 6 or higher, Mozilla Firefox and Google Chrome. It is platform independent and runs on Windows and Mac OS X. It will also run on Linux due to the *Moonlight* project that is currently under development. *Moonlight* is an open-source implementation of Silverlight for Linux [***26***]. [***27***]

Silverlight does not contain all the functionality that WPF provides in ordinary standalone Windows applications. Silverlight could be seen as a subset of all the functionality in WPF. For example is 3D modeling not supported in version two of Silverlight. [***27***]

6.1.2 XAML Browser Application

XAML Browser Application (XBAP) provides all functionality that is available in WPF. The prerequisites to run a XBAP as a web application are higher than Silverlight. It requires that the client computer has Windows with .NET Framework 3.0 installed and it can only run in Internet Explorer and Mozilla Firefox. XBAP applications also run in a secure sandbox in order to prevent access to local system resources e.g. access to local files. [***28***]

Two advantages with XBAP are: the possibility to make the web application look similar to a desktop application and it is easy to convert an XBAP application into a standalone WPF Windows application. [***29***]

XBAP is suited to use for internal solutions, where the number of users are restricted and where the environment could be controlled. In such environment it is also easier to ensure that all machines have .NET 3.0 installed.

6.1.3 WPF Technology Choice

Both XBAP and Silverlight were tested in a minor scale and after some consideration the choice became Silverlight. The major reasons for this were the multiple browser support and cross-platform support. Another reason was the perceived understanding that more people are using Silverlight which implies that more information is available about it.

One major reason to use XBAP instead of Silverlight would have been additional functionality but the prototype did not require anything that was not possible to implement using Silverlight.

6.2 Entities

An *entity object* is an object that represents an identity within a system. All objects within a system have a unique identity by its memory reference but unlike regular objects an *entity object* should be able to keep its identity even if the in-memory reference is removed. [***30***]

Example: *A Person object maintains an identity in the form of a personal security number. Even though the person is stored and loaded again and thus getting different in-memory representation it would still be the same person. If many copies of the same Person exist within a system they should be seen as the same object since they will all share the same identity.* [***30***]

Entity objects were introduced to hide much of the data structure complexity and provide an object oriented representation of the information in ILM (i.e the object types).

The C# classes, which describe the *entity objects*, will be generated by using the *Service Model Metadata Utility Tool* (Svcutil) that is provided by Microsoft**.** *Svcutil* can connect to an exposed web service (the *ILM WCF service* in this case) and generate classes based on the data structure that is available in a XML schema via the web service. In this thesis report these generated classes will be referred to as *entity classes* and instances of these classes

will be called *entity objects.* Examples of *entity classes* can be *Person* (that represents a user) and *Software*. [***31***]

All *entity classes* should define a *WCF Data contract* (see section 3.1.1) to make it possible to use the *entity classes* as data type in *WCF Service contracts*. Each *entity class* will implement a basic interface, *IEntity,* which can be seen in Code 5. The reason for this is to provide better support for transformation of *entity objects* to XML requests, this is described later in section 6.5.1. Since the *entity classes* that are generated by the *Svcutil* do not implement the *IEntity* interface it will be added manually.

Partial classes can be used to prevent manually changes done to *entity classes* from being overwritten when they are regenerated. Partial classes will split a class definition into different files which makes it possible to separate a generated part of a class from the part that was written manually. [***5***]

```
Public interface IEntity<T> where T : new() {

  // The id of the entity
  string ObjectID { get; set; }

  // Returns the changed attributes for this object
  ICollection<String> GetChangedAttributes();

  //Clear the changedAttribute list
  void ClearChanges();
}
```

CODE 5: THE IENTITY INTERFACE

Using *entity objects* will give benefits to the system. They make it possible to abstract the complex data structure they represent. They can also be used as parameters when constructing interfaces (for classes) and service contracts (for web services) so they become more descriptive and intuitive to use for developers. An example of how entities can be used in a service contract can be seen in Code 6.

```
[ServiceContract]
public interface ISoftwareService {

  [OperationContract]
  List<Software> ListAllSoftwares();


  [OperationContract]
  List<Software> SearchSoftware(Software template);

}
```

CODE 6: A SERVICE CONTRACT FOR THE PROTOTYPE WCF SERVICE.

6.2.1 Tracking changes

To make parsing of *entity objects* into XML messages (see section 6.2.1) easier it has to be possible to track changes made to *entity objects* attributes. Otherwise all data in an entity object has to be sent to ILM even though it has not been updated. This can lead to serious implications since ILM may trigger certain workflows or policies in external systems based on these updates. It is therefore essential that only data that has been changed is updated.

Many properties in an *entity object* have primitive types which will have default values in C#. If the changed properties are not tracked, the default initialization of primitive types can be interpreted as changes made to the *entity object* and will be sent as updates to ILM. This can lead to the problems described above but can also lead to inconsistent data. A possible workaround would have been the nullable type feature, which is provided since .NET 2.0, that allow primitive types to be nullable (assigned a null value) [***32***]. All attributes in such a solution either have a specific value or null. It should therefore be sufficient to do a direct comparison between the *entity object*, that should be updated, and the data that is available in ILM.

The primarily reason that the nullable type cannot be used is because it is not possible to generate entities with this feature using the *Svcutil*. Another minor problem is that nullable types are a .NET feature which some languages do not support. This can affect the WCF services so they cannot be fully used by other systems. With the current infrastructure this is not a problem but it can affect further integration with other systems.

Since the nullable feature is not available a direct comparison between all attributes in an *entity object* is not possible. A different strategy is to use a changed attribute array that will be added to each *entity object*. When a client make changes to a property, the name of the property will be added to the changed attribute array. This solution allows changed attributes to be tracked so the correct values can be updated in the ILM data storage.

The *Svcutil* program can be configured so that property listener functionality is automatically added to the generated entities. This will make it possible to attach a listener to the *entity object* and receive notification when the value of a property has been changed. The listener can be attached to the *entity object* in its partial class (see Code 7). When the value of an attribute is changed it will notify the attached listener and put the name of the attribute an array, called *changedAttributes* in Code 7.

```
public partial class Person : IEntity<Person>
{
  private ICollection<String> changedAttributes;

  public Person()
  {
    this.PropertyChanged += new PropertyChangedEventHandler(
Resource_PropertyChanged );
    changedAttributes = new HashSet<String>();
  }

  private void Resource_PropertyChanged(object sender,
PropertyChangedEventArgs e)
  {
       changedAttributes.Add(e.PropertyName);
  }
}
```

CODE 7: A PROPERTY LISTENER IS ATTACHED TO THE ENTITY OBJECT

Multi values (see section 3.2) will be generated as an array and the array is exposed as a property in an *entity object*. When content is added to the array it will not trigger a notification to the listener. The reason for this is that a notification is only triggered if the attribute itself is changed (i.e. the reference to the array) and not the content of it. To be able to track changes for these arrays, the content of the updated array will be compared to the stored array in ILM to find differences between them.

The array content comparison can be costly when the size of the arrays increases. A solution can be to add two additional arrays for each multi value in the ILM. One array will contain all the values that should be added and the other array will contain all the values that should be removed. This solution would avoid a comparison between all attributes in an array.

Adding two additional arrays for each multi value attribute will require lot of configuration in ILM. The configuration will take too much time from the development of other parts in the prototype. Therefore this solution will not be used. Instead the solution that does a direct comparison between the content of the arrays will be implemented.

Serialization of the different *entity objects*, when they are sent to different clients, is another issue that has to be solved. Since the implementation (i.e. the actual code) of an *entity object* is not preserved when deserialized by a client, the code that handles the change tracking will be lost. Even though the data is preserved the same tracking logic that was implemented on the server will be required to be implemented on the clients. This will not be an ideal solution but given the architectural requirements no other solution could be found to solve this problem.

6.3 The Silverlight client

The *Silverlight client* will handle the interaction with users, and transform input into appropriate system calls. Even though Silverlight provide many features to ease development of *Graphical User Interfaces* (GUI) some additional functionality had to be implemented.

Navigation and loading of different XAML pages in the GUI should be easy to handle programmatically. Therefore a master page will be introduced which maintains the basic page structure in the Silverlight client. The master page will contain content that is common for all pages, such as the header and the menu (see Figure 7). It will also handle the loading of each page so the loaded page gets inserted into the master page content container. The content container is the portion of the GUI that will change when the user navigate to a new page.

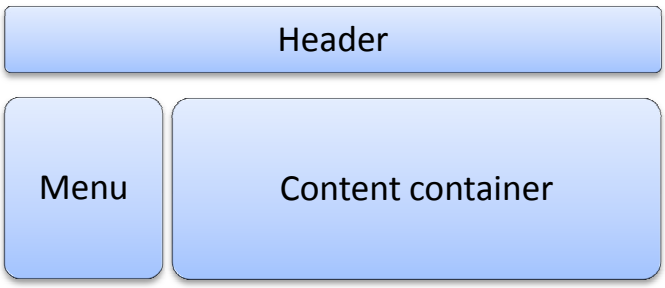

FIGURE 7: EXAMPLE OF A MASTER PAGE STRUCTURE

The master page will subscribe to a *change page event* on each page to handle navigation. An event will be triggered when a page detects an action from the user that requires navigation to a different page. When the master page retrieves the *change page event* it can replace the old content with the new one.

A Silverlight application maintains its state similar to a standalone application which can be used to keep the number of created pages at a minimum. The Silverlight client will provide a page factory that caches previously constructed pages. By using this factory the system will maintain a single page instance for each type of page. One benefit of keeping the amount of instantiated pages small is to prevent unnecessary communication with the *Prototype WCF service, s*ince each page usually contains web service requests to retrieve data.

Based on the requirements of the changed attribute tracking the Silverlight client will implement a separate service layer. The layer will handle the communication with the *Prototype WCF service* and for each *entity object* that is received, the required listener will be attached. This will make it possible to track changes to entities properties within the Silverlight client similar to what was described in the entity section.

By introducing a single place where the listeners are attached to all retrieved entities, this tedious and error prone work can be reduced. The service layer also reduces the impact that

the changed attribute tracking has on the application which will make it easier to replace or remove the functionality if a better solution is found.

6.4 The Prototype WCF Service

The *Prototype WCF service* will be used to hide the logic involved when communicating with the *ILM WCF service*. Since most of the business logic will be handled by ILM, the *Prototype WCF service* will only call the appropriate function in the data access layer (see section 6.5). If additional business logic is required, which ILM cannot provide, it can be placed inside the logic layer.

If all available functionality in the system is placed inside one *WCF service* the service can be hard to maintain and understand. Therefore functionality that logically belongs together should be grouped together and placed inside separate services. An example of this grouping is to create a software service that handles all functionality that is associated with *Software* and a different service that handle *Persons*. All services that should be available in the prototype can be found in Figure 8.

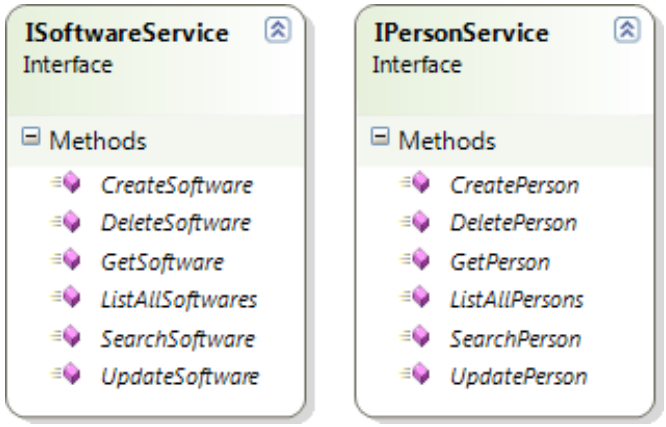

FIGURE 8: THE AVAILABLE SERVICES IN THE PROTOTYPE

6.5 Data Access Layer

The *Data Access Layer* (DAL) will contain all logic that is involved when communicating with the *ILM WCF service* and is equivalent to the Communication layer that was described in section 5.4. To support communication with ILM the DAL should be able to create request messages and parse responses into appropriate *entity objects*. Example of a request could be to update data in ILM based on the information in an *entity object*.

6.5.1 Parsing

A way to transform *entity objects* from its object oriented representation into XML has to be possible. It is equally important to be able to transform XML messages into *entity objects*.

The parsing functionality should be implemented in a generic way to be able to handle different and new types of entities without additional code. Failing to achieve this will make it tedious and error prone to adapt to changes made in ILM.

A possible solution can be to use a client library, which was developed by Microsoft, to help with parsing [***33***]. It can handle most of the parsing regarding XML messages from ILM but also provide classes to handle the communication with the *ILM WCF service*. By using the client some of the complexity of parsing can be removed and development time can be reduced.

The client library will parse responses, from the *ILM WCF service*, into special *ResourceManagement* objects. Simplified, these objects binds attributes to specific values that were retrieved from ILM data storage. If a request needs to be made, a *ResourceManagement* object will have to be constructed. It should contain the attribute and value mappings which will be transformed into a correct XML message. Because of this new class the parsing functionality will be slightly changed. *Entity objects* will be parsed to *ResourceManagement* objects instead of parsing *entity objects* directly to XML messages. An example of this process can be seen in Figure 9. The process is, as mentioned above, bidirectional since parsing has to be made in both directions.

FIGURE 9: THE PARSING PROCESS

To make the implementation of the parser generic the *Reflection library* will be used. That library is part of the .NET framework and provides functionality to handle properties, methods and constructors for different objects more dynamically [***34***]. By using reflection a parser can be created to handle different kinds of *entity objects* instead of implementing a parser for each of them.

The *Reflection library* can be used in conjunction with the client library solution. The client library solution has the advantage that the mapping from attribute to value is suited to work together with the *Reflection library*. Since each attribute in a *ResourceManagement* object has the same name as a property in an *entity object*, parsing can be done without much additional code.

Example: *A specific Person is requested from the ILM by sending a XML message. ILM sends back a XML message that contains the data for that person. The data is made up by a number of attribute names and their corresponding values that belong to the retrieved Person. The client library parses the data in the XML message into a ResourceManagent object. For each attribute name and its associated value, stored in the ResourceManagement object, the parser sets an attribute with the same name in the created Person entity object.*

In order for the client library to be used some changes will have to be added to provide better support for additional data types.

6.6 Handling workflows

The ILM architecture does not impose any specific constraints on how a workflow needs to be constructed. Most of the functionality available in the WF model can be used. It exist a number of ILM activities that can be added to a workflow. These activities provide specific information from ILM to the workflow.

When a workflow needs to be hosted by ILM, it should be added through the web portal that is provided by ILM. The web portal provides a convenient way to register workflows. The registration can be adjusted so workflows can be executed based on different events such as creation, deletion or updates of specific object types in ILM.

A workflow will be created which will handle the creation of a home directory for a registered person. The workflow should be triggered when a person has been created in AD since the unique identifier (called *ObjectSID*) for a person is needed (which is provided by AD). When ILM executes the workflow the home directory will be created and permissions will be assigned to it. These permissions will limit the access so only the registered person can use the directory. The workflow will be structured as described in Figure 10 and should contain two ILM activities (*currentRequestActivity* and *UpdateResourceActivity*) and a custom activity (*CreateHomeDir*).

The *CurrentRequestActivity* handle the retrieving of data that is needed to create a home directory, (*the ObjectSID*). The *CreateHomeDir* activity contains logic that creates the home directory based on the data that was received. Lastly the *UpdateResourceActivity* makes the appropriate updates in ILM so the Person will be assigned the newly created home directory.

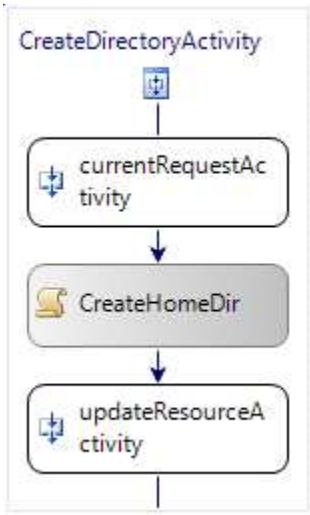

FIGURE 10: WORKFLOW FOR CREATING A HOME DIRECTORY

7 Working Process

This section describes the working process and how the work was divided. The project included many uncertainties and therefore the prototype was developed using an iterative development process.

7.1 Development Process Description

The functional requirements of the prototype were divided into separate iterations which made it possible to divide problems into smaller pieces.

In the end of each iteration the work was evaluated and demonstrated to the technical supervisor. The development of the prototype has been incremental since each iteration has introduced new learning in the form of possibilities and problems. These new learning have been used as input to next iterations and to further refine the design of the prototype.

The first iteration required that a good system design, which was scalable and flexible enough to support the functional and technical requirements, could be established. Even though the goal was clear from the start it was not obvious how the goal could or should be achieved.

Each iteration has contained the following phases: Identification and revision of requirements, system design, implementation, testing and evaluation. Each of these phases can be seen in the Figure 11, where the iteration starts with *Planning/requirement* and ends with *Evaluation* of the iteration. [***35***]

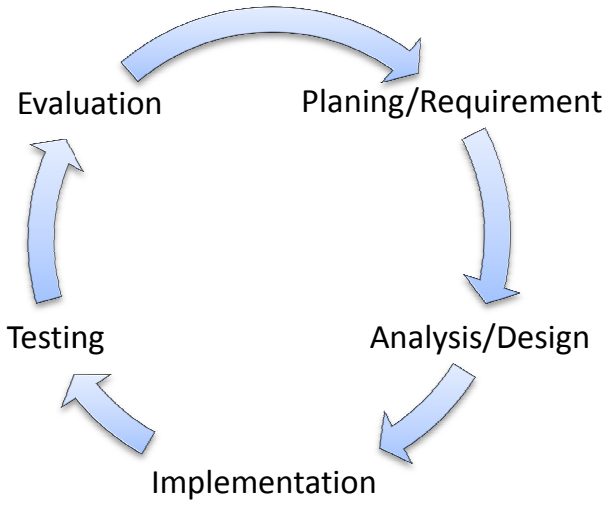

FIGURE 11: THE ITERATION PROCESS.

7.2 Iterations

The process to develop the prototype was divided into three separate iterations and assigned project time was based on the estimated complexity. Each iteration had a specific goal and provided input to the next iteration. The last iteration (*Iteration 3*) was assigned additional time to provide a buffer for unforeseen problems.

7.2.1 Iteration 1

The architecture relies on many different frameworks. These frameworks had to be tested and evaluated to find possibilities they provide but also restriction they impose. Many of the used frameworks were large and complex so the first iteration was dedicated to learn these frameworks. Because of the limited time and complexity of each framework only the basics were considered.

To make the framework evaluation more interesting, small prototypes were developed. Each prototype used the frameworks in different ways and made it possible to explore and solve many possible problems related to the system design. The small prototypes made it easier to stay focused and only evaluate parts of the frameworks that were required to fulfill the requirements.

The output from the first iteration was an evaluation of the available frameworks that provided good feedback to the system design.

7.2.2 Iteration 2

This iteration focused on the communication between all the components. Without a proper communication structure the functional requirements would have been more difficult to implement.

When the communication between each component was defined the *Register user* functionality could be implemented. During this implementation the look and feel of the GUI could also be established.

At the end of *Iteration 2* the following output was produced:

- The foundation of the system design was implemented.
- The parsing of the request and responses from the ILM was handled.
- The client interaction to ILM was solved.
- The look and feel of the GUI.

7.2.3 Iteration 3

The functionality to handle software distribution is similar to *Register user* in many parts of the system, except ILM. Because of this most of the work within this iteration was done within ILM. With the introduction of a new object type, *Software*, configuration had to be done so that ILM could handle it. It had to be possible to assign software to a user and the ILM had to synchronize with the correct databases.

The AD had to be set up so it could handle the actual software distribution (installation) when a user was assigned specific software.

At the end of *Iteration 3* the following output was produced:

- A new object type, *Software*, was introduced and successfully implemented.
- The AD and ILM were configured to handle software distribution.

8 Testing

Tests will primarily be used to evaluate the final testability of the architecture. The prototype is very likely to change over time when new requirements or solutions appear. Therefore it will be essential to react to these changes without breaking previously verified functionality. Regression testing will be done on important parts of the prototype.

The requirements of the testing framework were the following:

1. A test should be able to be reproduced so any team member can run the same test at any time.
2. It should be clear from the running test which class and methods it evaluates.
3. The result of a test should be viewable in a GUI, preferable in the *Integrated Development Enviroment* (IDE) itself, which should give clear response if the test failed or succeeded.
4. It should be possible to write the tests in C#.

A fitting methodology to use, based on the above requirements, is to construct unit tests to verify parts of the prototype.

8.1 Unit testing

A unit test is a way to verify that a part of the code that has been produced is working. A unit test is usually a method that invokes another method and check the result of that invocation against pre defined assumptions. If the result from the tested method does not fit with the assumptions, the test fails. [***36***]

Unit tests should be able to be automated and repeatable. If a unit test is run with the same code, it should produce the same result each time. Further it should be possible to run a unit test with just a single command and it should not require any external configuration [***36***]. A unit test should also produce some kind of result that indicates if the unit test failed or not.

A unit test can run inside an engine that calls all the unit test methods. This engine is usually called test driver. To provide a way to evaluate the result from the unit tests (if it is a failure or a success) a test oracle is needed [***37***]. The test driver and the oracle can be provided by using a testing framework.

8.2 Testing framework

It exist many different testing frameworks that help with creation and running of unit test. The first unit test framework that was evaluated was *NUnit* [***38***]. It is a well used and mature framework. It ships with a GUI for monitoring results of tests and functionality to support creating and running of unit tests [***38***]. Even though a GUI is provided one major drawback with *NUnit* is the lack of integration with VS2008. External add-ons exist, such as *Resharper* [***39***] and *Testdriven.NET* [***40***] to provide this integration, but they all require license costs.

Another unit test framework is the one that is included in VS2008 itself. It provides functionality very similar to *NUnit* but also provides a GUI to monitor the result of the tests within VS2008. [***41***]

Based on the testing process requirements the testing framework that fulfills most of the requirements is the unit test framework that comes with VS2008.

8.3 Testing structure

The number of unit tests for an application quickly becomes extensive and a clear structure of the unit tests should be maintained. Without a clear structure it becomes difficult to keep track on which unit test that is testing which class.

To make it easier to track which unit test that belongs to which class, all unit tests should be structured in the same way as the classes in the prototype solution.

Unit tests do not provide any functionality to the application itself and they should not be compiled together with the application assembly. Otherwise it can make the assembly unnecessarily large and more difficult for developers to use, since unit test classes will be mixed with the application classes.

A solution can be to create a separate testing project which will be compiled separately. It will prevent the testing classes to be compiled into the prototype assembly.

8.4 Testing of the prototype

When unit tests were produced for the prototype, focus was not to produce complete unit test coverage of the whole code base, but rather identify critical parts of the prototype that could affect testability.

8.4.1 Testing of the Data Access Layer

In the architecture the *Data Access Layer* (DAL) contains both the classes for communicating with the *ILM WCF service* and the parsers that handle the transformation between *entity objects* and XML. The communication with ILM is handled by the DAL and is a critical part to test. If the architecture cannot support a way to test this communication, testability will decrease.

Unit tests for the DAL could be created but there were still some issues that decreased the benefits. The data that the unit tests were based on was difficult to define. To be able to insert and remove data into ILM we used custom written *Insert* and *Delete* methods (from the DAL itself) in these test cases. If the *Delete* and *Insert* methods stopped working, many test cases would fail at the same time even though they were not directly related. This made it harder to find the bug that caused the failure. An example of this dependency can be seen in Code 8.

```
[TestMethod]
public void TestCreateAndDelete() {
  PersonDataAccess access = new PersonDataAccess();
  Person person = new Person();
  person.Domain = "liteware";
  person.DisplayName = "Created User";
  // The person is  created
  Person newperson = access.Create(person);

  Assert.IsNotNull(newperson.ObjectID);
  Assert.IsNotNull(newperson.DisplayName);
  // The person needs to be removed to clean up in ILM
  access.Delete(newperson.ObjectID);
}
```

CODE 8: DEPENDENCY ISSUE IN A UNIT TEST.

To solve this dependency problem a more direct communication with *ILM WCF service* could have been used. This would have required construction of XML messages from scratch which would have been both error prone and time consuming.

Another problem was that if the *Insert* method succeeded but the *Delete* method failed, data had been inserted into ILM which was not deleted afterwards. This left data in ILM that further corrupted tests and made it even harder to track the cause of failures. To fix this a developer had to manually delete the data to get the test cases to work again. This is an unacceptable scenario in a large project.

A solution would have been to move the insertion and deletion of test data into the *ClassInitialize()* and *ClassCleanup()* methods which run before and after a test case has been executed. A problem that can arise with this solution is if test data is changed. It may break tests that relied on the previous data that was defined in the *ClassInitialize()* method. This can make it tedious to manage data for a test case since it can be hard to predict if changes to data will break previously constructed tests.

8.4.2 Testing of workflows

A problem was encountered which prevented testing workflows that contained specific ILM activities. A workflow containing ILM activities has dependencies on an object of type *RequestType*. *RequestType* objects contains information about the request that triggered the workflow. These kinds of workflows will often need information in this object to deliver its functionality. Therefore the object needs to be provided to properly test the workflow outside of ILM.

Each time a *RequestType* was tried to be instantiated an exception was thrown. Because of limited documentation no information was found that could explained how the instantiation could be done in a successful way. Without a way to manually insert *RequestType* objects, workflows that use this kind of object cannot be unit tested.

9 Result

This section will present the result of the thesis. The first part will describe how the prototype works and what kind of functionality that it provides. The second part describes the architecture of the prototype and what kind of advantages and disadvantages this architecture gives.

Apart from the work with the prototype and the architecture, a lot of work as been spent on configuring ILM, in order to synchronize and provision all the desired data. We have managed to customize ILM in different ways, creating new object types, host and trigger workflow based upon policies that we have defined.

9.1 The Prototype

The prototype was implemented according to the SRS (see Appendix A), architecture and system design. The graphical user interface was implemented in WPF/Silverlight and has been clearly separated from the business logic with help of the *Prototype WCF service*. The *Prototype WCF service* provides the possibility to extend the prototype with different types of clients. The prototype contains two major features: *Register user* and *Distribute software*.

Figure 12 illustrates a part of the graphical user interface. In this view it is possible to manage users.

FIGURE 12: GRAPHICAL USER INTERFACE

Almost all the functional requirements listed in the SRS (see Appendix A) were implemented according to the specification. Except from the validation of input, some client side

validation exists but a stable version would require both server side validation and better general error handling.

The login and the logout features were not implemented according to the SRS. Instead these features were handled by Windows authentication and did not demand any implementation. It requires that the user have an account in AD. If the user can login to the intranet he/she will have access to the prototype, otherwise the user will have to enter login information in the web browser. The prototype does not provide any ability to logout. The reason for this is that the prototype is primarily intended to work as an internal application for companies and should be based on the same authentication as the internal network.

9.1.1 Register user

Register user makes it possible to create a new user with different attributes. The graphical user interface contains functionality to create, delete, update and search for users. As shown in Figure 12, it gives an overview of all the users in the system. This overview is important if the prototype should be used as a tool for identity management.

When a new user is registered a *Person* object is created at the client side with all necessary information. The object is sent through the *Prototype WCF service* (in XML) to the server side of the prototype. This object is then parsed into a *ResourceManagement* object. A XML message is created based upon the information that the *ResourceManagement* object contains. This message is sent to ILM through the *ILM WCF service*.

When ILM receives the message a user is created in the ILM database and provisioned to *Active Directory* (AD) with help of ILM synchronization engine. In ILM we can decide if all users should be provisioned to AD or we can specify certain criteria, for example that a user should be a specific employee type in order to exist in AD.

When the user has been synchronized to AD, ILM trigger a workflow that creates a home directory for the newly registered user. This way the triggering feature that ILM provides could be tested and evaluated based upon how it could be used to support the architecture.

The home directory is created with security permissions that only allow the created user to see or use the content of that folder. This is access management in an automated process. To be able to set permissions on a specific folder in a Windows environment, the user needs to exist in the Active Directory for the intended domain. The reason for this is that permissions are set by using the unique identifier for the user, which is generated when the user is created in AD.

The unique identifier in AD is called *ObjectSID.* By using the *ObjectSID* for a user from AD it is possible to set the correct permissions for a folder. That is why it is important that the workflow that creates the home directory is triggered after the user has been provisioned to AD and assigned a unique identifier. The home directory is created in a network share at a file server where the workflow has permission to create a new folder. WF has been proven

to work for this kind of automatization since ILM can host the workflow and trigger it based upon specified policies.

9.1.2 Distribute software

The *Distribute software* feature enables the possibility to distribute software to an existing user. When software is distributed to a user it is automatically installed on the computer that the user login to if the software is not already installed. The software will be available no matter which computer the user will use. Other users that will use the same computer will not have access to that software. It is possible to distribute software to specific computers instead of to specific users, this is however not fully supported by the prototype.

The interface lists all software that is available for distribution. It is then possible to search for a user and choose desired software in the list.

ILM did not contain any software object from the beginning so this had to be created with all the necessary attributes. This new object type resulted in a new XML schema that could be used to generate a *Software entity object* in the prototype.

The software object has a multi valued attribute in which the *ObjectID* for the associated users are stored. The user's *ObjectID* is then added to this attribute and the software object is transferred to the server side through the *Prototype WCF service* and then propagated to ILM through the *ILM WCF service*.

The distribution of software to a computer is handled by *Active Directory* (AD) and therefore any assignment of software to users has to be synchronized with the AD. A software object does not exist in AD so each software is modeled as a group in AD.

Each user that is assigned specific software is added as a member of the group that represents that software. The software distribution is performed by using group policies that tells AD to push out a specific *Microsoft Installer* (MSI) file to each member of a group. The group policy defines how the software should be distributed, if it should be automatically installed or if a user should be able to manually choose to install it. The group policy also contains information about the MSI-file and the physical path to it.

9.2 Architecture

Many aspects were discovered when the architecture was realized with the prototype. These aspects concerned attributes such as testability, separation of concern and scalability in both negative and positive ways.

9.2.1 Testability

Problems arose when test cases were created to verify that the *Data Access Layer* (DAL) was working as expected. It became hard to handle data that a test case should use and make sure that no data was left behind that could affect other test cases. No satisfying solution could be found that made it possible to clear ILM completely of data after each test case. In

some cases this left the ILM data storage in an inconsistent state which had to be solved manually.

It was possible to host a workflow within a test case which made unit testing easy. However, if the workflow had a dependency with the *RequestType* object, the *RequestType* object could not be provided since it was not possible to create an instance of it. This made it impossible to create unit tests for such workflows.

Most of the test issues were related to ILM in some way. Other parts of the system that did not have any direct relation to ILM could be tested successfully. Because of the layered architecture good separation could be upheld within the system. This separation made it easier to test each layer in isolation.

9.2.2 Separation of GUI and business logic

Most of the business logic in the prototype is maintained by ILM. Logic can be added by creating a workflow that will be hosted inside ILM. Additional business logic can easily be implemented inside the different *WCF services* or in the logic layer.

All business logic is hidden behind different *WCF services* and the *Silverlight client* could be completely separated from it. No business state had to be maintained within the *Silverlight client* which made it easier to implement.

9.2.3 Scalability

When a new object type is defined in ILM the creation of a corresponding *entity class* could be done automatically by using the *Svcutil* program. Although the *entity class* could be generated in an easy way some additional work had to be done in order for the *entity class* to be used in a proper way. First of all a partial class had to be implemented for the *entity class* to make it possible to track changes made to *entity objects*. Secondly, additional logic also had to be added to the *Silverlight client* to provide the same functionality.

The architecture can adapt to changes made to an object type in ILM (for example additional attributes), that is already generated as an *entity class*. This adaption can be made without any additional code. This made it very easy to make changes to object types in ILM that already had a generated *entity class* since all that had to be done was to run the Svcutil program again.

The *Data Access Layer* (DAL) provides base functionality such as *selectAll*, *select*, *search*, *delete* and *update* for any kind of *entity class*. This functionality is encapsulated in the abstract class *AbstractDataAccess*. If a new *entity class* is generated, little new code has to be written to provide these basic functionalities. The only code that has to be added is a new *DataAccess* class (for the newly generated *entity class*) that inherits from the *AbstractDataAccess* class.

A parser was created that made it possible to transform any kind of *entity object* into a *ResourceManagement* object and the other way around. If a new *entity class* is added or changed no updates needs to be made to the parser.

Because of the functionality to track changes made to *entity objects*, the parsing of entities to *ResourceManagement* objects was efficient. Only updated data will be handled which will make XML messages smaller.

Tracking of changes made to multi-valued attributes was not possible to implement with the current solution. Because of this all the content of a multi-valued attribute in an *entity object* had to be compared with the stored information in ILM to find updates.

10 Discussion

Throughout the thesis work much emphasis has been put on trying to find reasonable solutions to all encountered problems. This section will contain thoughts and reflections around these solutions.

Another way of handling software distribution is to use a product called *System Center Configuration Manager* (SCCM). It is a product from Microsoft and provides features such as Software Distribution, Software Update Management and Operating System Deployment. Even though there may be many benefits of using this alternative, it was omitted to keep the infrastructure simple since SCCM would require another system to be installed and configured.

A release candidate of ILM has been used and some anomalies between the finished ILM product and the release candidate may occur. When this thesis report was written the ILM was delayed to the first quarter of 2010 which may introduce additional changes that can affect the architecture or the prototype.

Creating unit tests for the *Data Access Layer* (DAL) was hard since it was difficult to structure the tests to avoid dependencies between them. Managing test data quickly became tedious and unreliable. Even though these issues affect testability it seems to be more of a general problem when testing data sources then specific for ILM. Based on the unit tests, the structural difference (and the problems related to it) between testing another kind of data source and ILM seems small.

No conclusion can be drawn if the architecture scales with a larger database. Therefore it is unknown what will happen when more object types and more data is added. The XML response returned by some queries can be large, even when requesting just a few objects. Developers will have to be precautious when constructing queries since fetching too many objects will slow down the system considerably.

Performance testing regarding the number of users could not be tested. It is therefore unknown what will happen with performance if many users will use the application at the same time.

Since an extra parsing step was introduced with the *ResourceManagement* objects it can have negative impact on performance compared to parsing XML messages directly to *entity objects*.

The *client library* can be removed without affecting the rest of the prototype. The parsers however will have to be adjusted to support parsing XML messages instead of *ResourceManagement* objects.

We could conclude that the architecture fulfilled the predefined requirements, based on the result of the prototype. It is unknown if the architecture can support full automatization of

all IT processes or if some processes would require extensive changes to the architecture. The two implemented features were chosen because they contained parts that are common in many IT processes. It is therefore likely that the architecture can support more processes than the prototype provides.

Authentication was implemented by using Windows authentication. Windows authentication is supported by WCF which made it possible to use this kind of authentication in the architecture [***42***]. Based on the credentials, that was provided through authentication, access to the Silverlight client could be limited to certain users or groups.

It was desirable to use the credentials to control access to the ILM. This feature could not be fully implemented in the prototype because of time constraints but it seems likely that it will work with the architecture [***7***]. A solution would be to transfer the credentials, used in the *Prototype WCF service,* and use these credentials when communicating with the *ILM WCF service*. The current solution uses administration credentials defined in the code when communicating with the *ILM WCF service*.

11 Conclusions

The thesis has proved that it is possible to use WPF, WCF, WF and ILM to develop applications for automated IT processes. The prototype that has been produced in this thesis is adaptable for different automated processes that could occur in companies. There are some constraints on the infrastructure, for example it must be a Windows environment and have Active Directory and Microsoft Server 2008.

The result could be seen as a step towards a customized solution for companies to handle identity management and access management in an automated way. The prototype relies on the strength and functionality from ILM but contributes with customization and additional functionality.

It is possible to develop an architecture for automatization of IT processes with these technologies, that supports scalability and a clear separation between the business logic and the graphical user interface. The architecture will allow the prototype to grow and will enhance future development.

The testability in this architecture could be improved especially those parts that communicate with ILM.

The client library from Microsoft was used to enhance and simplify the communication with ILM. This client library was only released for demonstration purpose and therefore a more elaborated version could be developed. This version could be more suitable for the architecture and better integrated with the parser.

The current implementation of the parser fulfills all the requirements of the prototype. However, the parser could be further refined to support more data types and it would also be interesting to investigate if the performance of the parser could be increased.

The prototype supports tracking of changes made to entities which implies a more efficient parsing and it assures that no unnecessary updates of data in ILM will be made. An improvement could be to implement a solution for tracking of changes that would not require a client side implementation. It would probably also be possible to improve performance especially for tracking of changes made to collections.

12 Bibliography

[1] Microsoft Corporation. Understanding Identity Lifecycle Manager "2".; 2008.

[2] Watson K. Beginning Microsoft Visual C#: Wrox Press; 2008.

[3] Templeman J, Vitter D. Visual Studio.NET: The.NET Black Book Scottsdale: Coriolis Group Book; 2002.

[4] Troelsen A. Pro VB 2008 and the.NET 3.5 Platform. Third Edition ed.: Apress; 2008.

[5] Liberty J, Xie D. Programming C# 3.0. Fifth Edition ed.: O'Reilly Media Inc.; 2008.

[6] Casada K. Introduction to Windows Communication Foundation [Document].

[7] Klein S. Professional WCF Programming.NET Development With The Windows Communication Foundation Indianapolis: Wiley Publishing Inc.; 2007.

[8] Myers BR. Foundations of WF: An introduction to Windows Workflow Foundation: Apress; 2007.

[9] Bukovics B. Pro WF: Windows Workflow in.NET 3.5: Apress; 2008.

[10] MacDonald M. Pro WPF in C# 2008 Windows Presentation Foundation with.NET 3.5. Second Edition ed.: Apress; 2008.

[11] Thorn A. DirectX 9 Graphics: The Definitive Guide to Direct3D: Wordware Publishing, Inc; 2005.

[12] MacDonald M. Pro.NET 2.0 Windows Forms and custom controls in C#: Apress; 2005.

[13] Moroney L. Foundations of WPF: An Introduction to Windows Presentation Foundation: Apress; 2006.

[14] Microsoft Corporation. MSDN Visual Studio 2008 Development System. [Online]. [cited 2009 April 21. Available from: http://msdn.microsoft.com/sv-se/vstudio/products/bb931331(en-us).aspx.

[15] Collins-Sussman B, Fitzpatrick B, Pilato M. Version Control with Subversion. [Online].; 2008 [cited 2009 April 22. Available from: http://svnbook.red-bean.com/en/1.5/svn-book.pdf.

[16] Microsoft Corporation. MSDN Visual Studio 2005 Development Center. [Online]. [cited 2009 April 21. Available from: http://msdn.microsoft.com/en-us/library/3h0544kx(VS.80).aspx.

[17] Craft M, Cross M, Kurz H, Barber B. How to Cheat at Designing a Windows Server 2003 Active Directory Infrastracture. First Edition ed.: Syngress Publishing; 2006.

[18] Henrickson H, Hofmann SR. IIS 6: The Complete Reference: McGraw-Hill Professional; 2003.

[19] Microsoft Corporation. Managing Users, Groups, Policy and Credentials Using ILM "2". 2009..

[20] W3C. w3.org. [Online].; 2007 [cited 2009 May 21. Available from: http://www.w3.org/TR/soap12-part1/.

[21] Schulman J. Identity Management Extensibility. [Online].; 2008 [cited 2009 April 23. Available from: http://blogs.msdn.com/imex/archive/2008/11/05/terminology-used-in-ilm.aspx.

[22] Meier J, Homer A, Hill D, Taylor J, Bansode P, Wall L, et al. Application Architecture Guide 2.0 Designning applications on the.NET Platform.; 2008.

[23] Baldoni R, Marchetti C, Termini A. Active software replication through a three-tier approach. In Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems; 2002; Washington: IEEE Computer Society.

[24] Barry D. Web Services and Service-Oriented Architecture: The Savvy Manager's Guide: Morgan Kaufmann; 2003.

[25] MacDonald M. Pro Silverlight 2 in C# 2008: Apress; 2008.

[26] Mono. [Online]. [cited 2009 Mars 28. Available from: http://www.mono-project.com/Moonlight.

[27] Ghosh J, Cameron R. Silverlight 2 Recipes: A Problem-Solution Approach: Apress; 2009.

[28] XBAP.org. [Online]. [cited 2009 April 2. Available from: http://www.xbap.org.

[29] Corby K. Scorbs. [Online].; 2006 [cited 2009 May 21. Available from: http://scorbs.com/2006/06/04/vs-template-flexible-application.

[30] Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software: Addison-Weasley; 2004.

[31] Microsoft Corporation. MSDN.NET Development Center. [Online]. [cited 2009 April 14. Available from: http://msdn.microsoft.com/en-us/library/aa347733.aspx.

[32] Microsoft Corporation. MSDN.NET Development Center. [Online]. [cited 2009 Mars 27. Available from: http://msdn.microsoft.com/en-us/library/1t3y8s4s(VS.80).aspx.

[33] Microsoft Corporation. MSDN Code Gallery. [Online]. [cited 2009 Mars 10. Available from: http://code.msdn.microsoft.com/imexsamples/Release/ProjectReleases.aspx?ReleaseId=1824.

[34] Microsoft Corporation. MSDN.NET Development Center. [Online]. [cited 2009 April 16. Available from: http://msdn.microsoft.com/en-us/library/system.reflection(VS.85).aspx.

[35] Sommerville I. Software Engineering. Seventh Edition ed.: Pearson Addison Wesley; 2004.

[36] Osherove R. The Art of Unit Testing: With Examples in. NET: Manning Publications Company; 2009.

[37] Cheon Y, Leavens G. A Simple and Practical Approach to Unit Testing: The JML and JUnit Way. Ames: Iowa State University; 2003.

[38] NUnit.org. NUnit. [Online].; 2009. Available from: http://www.nunit.org/index.php.

[39] JetBrains. The Most Intelligent Tool for Visual Studio. [Online].; 2009. Available from: http://www.jetbrains.com/resharper/.

[40] TestDriven.NET. Testdriven.NET. [Online]. Available from: http://www.testdriven.net/.

[41] Microsoft Corporation. MSDN Microsoft Developer Network. [Online]. Available from: http://msdn.microsoft.com/en-us/library/ms243147.aspx.

[42] Microsoft Corporation. MSDN -.NET framework Developer Center. [Online].; 2007 [cited 2009 April 15. Available from: http://msdn.microsoft.com/en-us/library/ms733082.aspx.

Appendix A – Software Requirements Specification

Software Requirements Specification

for IT Automation Prototype

Version 1.0

Prepared by Marcus Melberg & Markus Westerström

Atea/Spintop

February 5, 2009

Table of Contents

1. Introduction

1.1 Purpose

The purpose of the Software Requirement Specification (SRS) is to describe and state all requirements for the prototype that will be used in the master thesis project, "Automated IT Processes". Each system feature and each functional requirement have been prioritized in scale low, medium or high. This document also aims to explain the purpose of the prototype and how the prototype will be constructed. This SRS will also be used as a reference document during the development phase in this project.

1.2 Document Conventions

This standard outlines the documentation standards and formatting to be used in this document.

The alignment of a document will be justified. (ie. cover entire page.) Paragraphs will contain a single line space between them. The font to be used in the body of all documents is Times 12pt. This will not apply to headers and footers and some low level headings. Headings will have font Times and each word in a heading should start with an uppercase letter. Only three levels of headings should be showed in the table of contents. A fourth level could be used in the document if it is necessary.

1.3 Intended Audience and Reading Suggestions

The intended audience of this document includes system designers, programmers and also supervisors for the master thesis project. Emphasis is placed on what kind of functions that the prototype will be able to perform and the requirements for them and not how to perform these functions. Suggestion for reading is to begin with the overview sections and proceeding with the more detailed sections about the requirements.

1.4 Project Scope

The software prototype that will be produced by this project will be developed for investigation purposes and will provide valuable information to a larger project. The prototype will contain all frameworks and functionality that is needed in order to conduct an investigation that is described by the master thesis "Automated IT Processes". The focus for the master thesis is to investigate and evaluate if WPF, WCF, ILM and WF together are appropriate tools to use when developing automated processes for IT.

The investigation and the measurement of the different tools are outside the scope of the development process for this prototype

The prototype will basically include two separated parts, one part that will be focused on the GUI developed in WPF and another part that will contain all business logic developed in WF and ILM. The communication between these two parts will be constructed in WCF. The prototype will be a well functional application that can be used in automated IT processes but it will have a limited number of functions.

1.5 Definitions, Acronyms and Abbreviations

API – Application Programming Interface
GUI – Graphical User Interface
ILM – Microsoft Identity Lifecycle Manager
IT – Information Technology
WCF – Windows Communication Foundation
WF – Windows Workflow Foundation
WPF – Windows Presentation Foundation

1.6 References

The following references were used in the construction of this SRS:

- Description of Master Thesis in Software Engineering, “Automated IT Processes” by Marcus Melberg and Markus Westerström
- ISO 15504

2. Overall Description

This section will provide information and an overview of the prototype. It will also describe what the context, functionality and running environment for the prototype are.

2.1 Prototype Perspective

The prototype that will be developed will be used to test a new architecture for automated IT processes. The architecture may later replace an older one to provide better functionality, easier maintenance and better separation between GUI and business logic. The prototype will be a web application providing basic functionality.

The prototype will contain two parts; a client part and a server part. Figure 1 illustrates the intended architecture for the prototype where WPF Client is the GUI which will communicate with server side through a WCF service. The server side will communicate with ILM through another WCF service. ILM will then handle the communication and the synchronization with different data storages and Active Directory. When ILM store data into the data storages it will also handle the process of triggering possible workflows that will execute work and provide functionality associated with the data that was stored. The work done by the workflow might eventually trigger another workflow and when all intended work is done by the server ILM should give a response back to the server side of the prototype and this will then be propagated to the WPF Client.

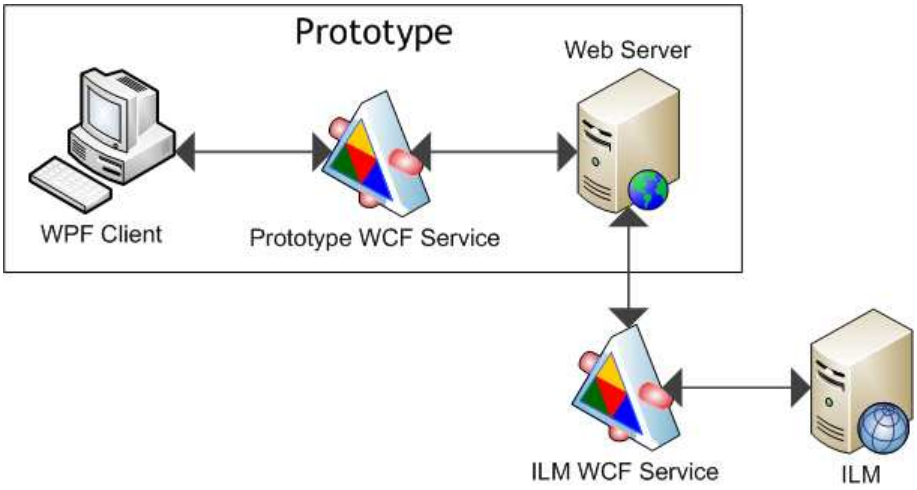

Figure 1. System architecture

2.2 Prototype Features

It exist two major features in the prototype which both has been chosen to reflect typical usage scenarios. These features will provide meaningful insight on how functionality can be implemented using the new architecture and the related frameworks.

The two major features are Distribute Software and Register User. Register User aims to provide functionality to add a user to a existing network system while the Distribute Software handles both

the ordering itself but also installation of the software on the user's local computer. Details about these software features are described in section 3. System Features

2.3 User Classes and Characteristics

The main user of a product similar to the prototype is the working staff in a company that runs the product. The working staff could be any person working in a medium large company. They will use most of the features that are offered in the product and they might have limited technical knowledge. This will require the GUI of the application to be intuitive and easy to use to hide the complexity of the underlying processes.

Another user is the administrator of the product that probably has good technical knowledge. These users will only use the administrator functionality provided in the product to perform actions that require better underlying technical knowledge then the working staff.

2.4 Operating Environment

At this stage the product is divided into three different components, one client, one web server and one server that host ILM.

The client demands a browser that is compatible with the Silverlight runtime. The operating system for the client can vary as long as it supports a compatible browser.

The web server will run IIS 7 and will host the prototype.

The server that hosts ILM will run on Windows Server 2008 and it will use the .NET3.5 runtime. It will use the ILMv2 to host workflows, handle requests from the web server component and to communicate with external resources.

2.5 Design and Implementation Constraints

The prototype relies heavily on frameworks written in C# and VB. Since we have more knowledge about C# it will be the programming language for the prototype. Furthermore the .NET 3.5 library will be required so maximum functionality from the frameworks can be used.

Communication between the client and the server will be handled by a pre built WCF component. All communication between the client and the server should go through this component.

ILMv2 that hosts the workflows, provide persistence support and communicates with other external resources.

To increase productivity and provide maximum support to the used frameworks the whole product should be developed using Visual Studio 2008 (VS2008). VS2008 provide good support for most of the used frameworks and will increase the productivity of the team.

Even though the prototype should be developed as it would be used by customers it is important to emphasis that it is only a prototype and won't be further maintained by anyone.

2.6 User Documentation

Since the product is a prototype, no user documentation will be constructed. Since the code may be of interest the documentation of the code will be extracted and be available as an API for the prototype.

2.7 Assumptions and Dependencies

The GUI will be implemented using Silverlight, which is a subset of WPF. Therefore great care should be taken not to rely on functionality provided through WPF but is not supported in Silverlight. To provide more functionality the Silverlight toolkit has been added. It provide new and updated controllers on top of the built-in controllers that ships with Silverlight.

The workflows that will be implemented should use the WF framework and be hosted by the ILMv2 component. Workflows will be required to communicate with ILMv2 using the pre-defined interface.

Active Directory (AD) will be used to handle user information, user groups, policies and authorization.

3. System Features

This section provides information about the system features, use cases and requirements.

3.1 Register User

Hight priority. It should be possible to register a new user.

3.1.1 Stimulus/Response Sequences

Actor: User
Goal: Register a new user
Precondition: User is logged in.
Postcondition: A new user is registered. A folder has been created, home directory, which the user has read and write privileges on.

Main Flow
1. User enters personal information
2. System validates the personal information
3. System creates a directory and gives the user read and write privileges on the created directory
4. System returns information of the new registered user and the location of the folder

Alternative Flows
2a. Validation fails
1. The system return information on which information that was wrong
2. Go to 1

3.1.2 Functional Requirements

3.1.2.1 Enter personal information

User should be able to enter personal information when registering. Following fields should be available.

- Username
- Password
- Name
- E-mail
- Address (Optional)
- Telephone number (Optional)

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.1.2.2 Validation of input

The system should perform validation of input. The system should validate following:

- Password length

- E-mail format correct
- Validation of required fields, (that they contain any text)

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: Medium

3.1.2.3 Feedback to user

The system should give feedback to user, show which fields that are not correct.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: Medium

3.1.2.4 Create a directory

Create a directory (home directory) that only the registered user has read and write privileges on.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.2 Distribute software

Hight priority. The user should be able to choose software to install on the computer.

3.2.1 Stimulus/Response Sequences

Actor: User
Goal: Distribute software
Precondition: User is logged in
Postcondition: Software is installed on the user's computer

Main flow:

1. User asks for a list of all available software
2. System returns all the software that the user is allowed to order
3. User choose software to install
4. System validates the order
5. System place an order on all the software that was validated
6. System returns information about the order

3.2.2 Functional Requirements

3.2.2.1 See available software

The user should be able to list all software that is available for the role that the user is associated with.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.2.2.2 Choose software

From the software list the user should be able choose one or more applications to install that has not been previously installed.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.2.2.3 Feedback to user

The system should give feedback to the user if the selected software was successfully ordered or not.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: Medium

3.2.2.4 Receive status

The user should be able to receive status upon the installation of all the successfully ordered software. If it waits for approval from administrator, not available at the moment, waiting for installation to start, successfully installed.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: Medium

3.2.2.5 Install software

The ordered software that receives status “waiting for installation to start” should be installed on the user computer (that ordered the software).

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.2.2.6 Information about installed software

The system should provide information about what software that has already been installed.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: Medium

3.2.2.7 Distribute software as administrator

A user with administrator privileges should be able to distribute software for any user.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: Medium

3.3 Login

Hight priority. A user should be able to login to the system.

3.3.1 Stimulus/Response Sequences

Actor: User
Goal: Login to the system
Precondition: User is not logged in to the system
Postcondition: User is logged in to the system

Main flow:
1. User enter login information
2. System validates login information
3. System sets the User as logged in
4. System returns information that the login was successful

Alternative flow:
2a. Validation of login information failed
1. System returns information that the login was unsuccessful

3.3.2 Functional Requirements

3.3.2.1 Login possibility

A user should be able to login on the system using, by providing username and password.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.3.2.2 Login feedback

The system should provide feedback if the login was successful or not.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.4 Logout

3.4.1 Description and Priority

Hight priority. A user should be able to logout from the system.

3.4.2 Stimulus/Response Sequences

Actor: User
Goal: Logout from the system
Precondition: User is logged in to the system
Postcondition: User is logged out from the system

Main flow:
2. User asks the system to logout.
3. The system gets the current logged in user and sets him as logged out.

3.4.3 Functional Requirements

3.4.3.1 Possibility to logout

A user should be able to logout from the system.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.4.3.2 Provide feedback about logout

The system should give feedback to the user, if the user was successfully logged out or not.

CATEGORY: Functional
CLASSIFICATION: Mandatory
PRIORITY: High

3.5 Handle User Information

Low priority. Users should be able to change their personal information.

3.5.1 Stimulus/Response Sequences

Similar to Stimulus/Response Sequences Stimulus/Response Sequences3.1.1.

3.5.2 Functional Requirement

3.5.2.1 Change user information

A user should be able to change his/her personal information.

CATEGORY: Functional
CLASSIFICATION: Optional
PRIORITY: Medium

3.5.2.2 Change user information as administrator

A user with administrator privileges should be able to change any user's personal information. A user with administrator privileges should be able to change any user's group privileges.

CATEGORY: Functional
CLASSIFICATION: Optional
PRIORITY: Medium

3.5.2.3 Perform validation of user information

The system should perform validation of the entered user information. The system should validate following:

- Password length
- E-mail format correct
- Validation of required fields, (that they contain any text)

CATEGORY: Functional
CLASSIFICATION: Optional
PRIORITY: Medium

3.5.2.4 Provide feedback on changes

The system should be able to provide feedback on the changes that have been made and indicate if any information were not correct entered.

CATEGORY: Functional
CLASSIFICATION: Optional
PRIORITY: Medium

4. Nonfunctional Requirements

4.1 Performance Requirements

The performance of the prototype is not an important requirement but the performance should be similar to the already existing webbportal that Spintop has developed.

4.2 Safety Requirements

The source code of the prototype should not contain any uncrypted sensible information such as passwords.

Authentication of a client should be performed by using Window.Authentication.

4.3 Software Quality Attributes

Testability: All of the functional requirements should be able to be tested using unit test. Graphical elements, such as buttons or links, do not have to be tested using unit tests.

Usability: All actions users do should provide feedback if it has been successfully done or not.