AN APPLICATION USING WCF

Toni Pikkumäki

Bachelor's Thesis
December 2012

Degree Programme in Software Engineering
Technology, Communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)
PIKKUMÄKI, Toni

Type of publication
Bachelor´s Thesis

Date
9.12.2012

Pages
34

Language
English

Permission for web publication
(X)

Title
AN APPLICATION USING WCF

Degree Programme
Software Engineering

Tutor(s)
SALMIKANGAS, Esa

Assigned by
Inmics Software Engineering Oy

Abstract

This thesis examined the benefits and implementation of using a WCF service in a client-server application. The main focus of the thesis was on the server side; however the client side was also briefly handled in the implementation section.

The first part of the thesis studied the different parts of the WCF service. The purpose of this part was to offer a good idea of what the WCF service is and what the components related to it are. These components include an explanation on what the client-server architecture is and what Windows Communications Foundation and its most important service related components are. The components being examined include contracts, endpoints and the service itself.

The implementation of the service is based on a project by the assigner, where a similar service was created and it focuses mostly on the server side. The client side part of the implementation details how the service was referenced using the Visual Studio 2012 and how the methods from the service are called using a service client. The purpose of the implementation was also to serve as a tutorial on how to create a WCF service.

Result of the thesis was a working implementation of the WCF service and when it was compared to other possible implementation methods, the WCF service was an optimal choice for applications with multiple concurrent users. The downside of the WHC service was its need to be hosted by a server, which will cause all applications using the service to become impaired if the server fails.

Keywords
.NET, WCF, Windows Communications Foundation, Service, C#

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN KUVAILULEHTI

Tekijä(t) PIKKUMÄKI, Toni	Julkaisun laji Opinnäytetyö	Päivämäärä 9.12.2012
	Sivumäärä 34	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
AN APPLICATION USING WCF

Koulutusohjelma

Ohjelmistotekniikka

Työn ohjaaja(t)
SALMIKANGAS, Esa

Toimeksiantaja(t)

Inmics Software Engineering Oy

Tiivistelmä

Opinnäytetyö tutki WCF servicen käyttämisen hyötyjä WCF sovelluksessa sekä kuinka kyseinen service toteutetaan. Opinnäytetyö painottuu enemmän palvelinpuolelle, mutta käyttöliittymäpuoltakin käsitellään toteutusvaiheessa osittain. Opinnäytetyö tehtiin kyseisestä aiheesta, koska toimeksiantajan projektissa oli sille tarvetta ja teoriapuolesta oli suuri hyöty ymmärtämään, miten ja miksi servicen eri osaset toimivat.

Opinnäytetyössä keskityttiin esittelemään WCF servicen eri osat. Tämän osion tuli tarjota hyvä idea siitä mitä SOA, WCF ja service ovat ja mitä niihin liittyvät eri osat ovat. Esitellyt osiot sisältävät yleisen selostuksen siitä, mitä SOA ja käyttöliittymä-serveri tarkoittaa sekä osion joka tutkii mikä on Windows Communication Foundation ja sen täkeimmät serviceen liittyvät osat. Lisäksi keskityttiin kuvaamaan servicen luomista kohta kerrallaan. Toteutus pohjautui toimeksiantajan projektiin, jossa vastaavaa toteutusta käytettiin. Tämän toteutuksen oli myös tarkoitus pystyä toimimaan eräänlaisena ohjeistuksena WCF servicen toteuttamiseksi.

Toteutus keskittyy pääsäännöllisesti palvelin puoleen ja itse servicen toteutukseen. Käyttöliittymäpuolella toteutus käsittelee ainoastaan sen, miten serviceen viitataan käyttäen Visual Studio 2012, sekä miten luotua viittauksen avulla kutsutaan servicen toimintoja.
Opinnäytetyön toteutuksen tulokseksi tuli toimiva servicen luomisen ohje ja toteutettu service toimi kuten oli suunniteltu. Kun serviceä verrattiin muihin toteutusmenetelmiin, päädyttiin tulokseen, että service on hyödyllinen, kun sovelluksella on useita yhtaikaisia käyttäjiä ja servicen suurimmaksi haittapuoleksi todettiin sen tarve olla palvelimella, joka kaatuessaan estää servicen käytön.

Avainsanat (asiasanat)

.NET, WCF, Windows Communications Foundation, Service, C#

Muut tiedot

CONTENTS

1 BACKGROUND 6
1.1 Introduction 6
1.2 Objectives of thesis 6
1.3 Assigner 7
2. SERVICE-ORIENTED ARCHITECTURE 8
2.1 What is service-oriented architecture? 8
2.2 Client and server 8
3 WINDOWS COMMUNICATIONS FOUNDATION 10
3.1 Basics 10
3.2 Contracts 11
3.3 Endpoints 14
3.4 Service 16
4 IMPLEMENTATION 18
4.1 Background 18
4.2 Data transfer objects 20
4.3 The service 21
4.4 Hosting the service 24
4.5 Using the service in client 27
5 RESULTS AND CONCLUSION 31
5.1 Results 31
5.2 Conclusion 32

FIGURES

FIGURE 1. Server-client architecture and request/response behavior.....9

FIGURE 2. Structure of the application. 19

FIGURE 3. Adding a service reference in Visual Studio 2012. 28

FIGURE 4. Configuring the service reference in Visual Studio 2012. 29

TERMINOLOGY AND ACRONYMS

.NET Framework

.NET framework is a software development environment by Microsoft that can be used to make various types of applications, from web based software to desktop and mobile phone applications. The main development tool of .NET is Visual Studio that primarily focuses on C# and Visual Basic programming languages.

Active Directory

Active directory is Microsoft's domain service that acts as a single location for different network related information, such as domain names, user privileges or user information, like passwords and usernames. (So what is Active Directory? (Windows).)

API

Application programming interface is a collection of functions and behaviors that allow the software developer to use items that are often in form of classes, contained by the API in their own code to perform various tasks and procedures.

Asynchronous method

Asynchronous method is a method that does not follow linear execution of application code. When at some point in code asynchronous method is called, a new thread is created for the asynchronous method, while the method calling code stays within its own thread.

When calling the asynchronous method, a callback method can be given for the asynchronous operation. Once the asynchronous method or function has been completed, the callback method is called.

Best Practice

> *"Best practice" is a phrase used to identify a documented way of achieving specific results under specific circumstances in an effective way.*
>
> (What are Best Practices?)

Entity

An object that represents a single item in Entity Framework, like for example a Person. When used with Entity-relationship model, entity can describes a table or collection of tables in a database.

Entity Framework

> *Entity Framework (EF) is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write.*
>
> (What is entity framework?)

Event argument

In .NET framework, event argument is a class that contains information about actions done in event the arguments are for. Event arguments are usually passed along with the event and are used to both get information and in some cases affect the event itself.

Guid

Guid, short for Globally unique identifier, is a collection of bytes that are completely unique and can be randomly generated when needed.

Session

Session is a single use-case of application or application part, with memory and data specific to that session only. When new session begins, all previous session information is lost unless stored.

Service-oriented architecture

Service-oriented architecture (SOA) is a collection of technologies, principles and guidelines for developing a collection of services that are capable of communicating with each other. (Service-oriented architecture (SOA) definition.)

SOAP

Simple Object Access Protocol (SOAP) is Extensive Markup Language using protocol that is used to transfer information to and from a Web Service.

SSL

SSL, short for Secure Sockets Layer is a cryptographic protocol providing communication security over the Internet. (Transport Layer Security.)

WPF

Windows Presentation Foundation (WPF) is used to create graphical layers for application. WPF uses .Xaml type of files with code behind logic. WPF can be described as an alternative to for example the more traditional Windows Forms implementation for an application user interface.

Wrapper class

A class that contains or mimics another class and is used to call that class without actually calling the class itself is called wrapper class. This allows using classes in different ways than they were originally meant to be used.

WCF

WCF, short for Windows Communication Foundation, is built using .NET framework, which is designed to make communication between different parts of applications or multiple different applications easy.

1 BACKGROUND

1.1 Introduction

In object oriented software development, a good quality application separates UI, short for User Interface, from the data layer and core functions of the application, which is commonly known as business logic, forming the “middle layer” of the application. Multiple ways exist to implement this business logic, some of which are programming environment specific, while others are shared between platforms.

In .NET environment one of these ways to implement the middle layer is by using .NETs WCF service. This service provides a dynamic and flexible way to achieve the middle logic, by separating the application to two distinct portions, client and server. The client portion contains the UI and client related business logic, while the server side contains the data layer, in addition to the service and business logic of its own.

Through the service, the server portion of the application forms a service-oriented architecture (SOA) for the application. Core behavior of this SOA is attained with the help of Windows Communication Foundation (WCF), which is a collection of APIs provided by the .NET framework, WCF service being one of the primary APIs of this framework.

Implementation of this service was based on a software development project from the assigner. In this project, service was used to asynchronously transfer data to and from the database, using Entity Framework as the data layer and Windows SQL server to store it.

The implementation shown in this thesis is based on a project by assigner company of this thesis. The project was a working hour tracking software, using WCF service to communicate between the client and server sides.

1.2 Objectives of thesis

The objective of this thesis was to illuminate what the WCF based service is, giving a good understanding of its behavior and what is required of the application in order to use it. In addition, the aim was to clarify what the benefits of implementing WCF

service are and how to implement it. When investigating the benefits of the service, the aim was to compare the service with common ways to implement similar business layers and find out the strengths and weaknesses of the WCF service when compared to the others.

The aim of the implementation section of this report was to give a basic understanding of what the final product was like, while also serving as a tutorial for implementing a WCF service.

The main focus of the thesis was on the server side of the application structure, focusing on the service and parts of code that are required and optimal with it. Both the data layer and the client side were mostly ignored.

1.3 Assigner

Inmics Software Engineering Oy is a software development company, a branch of the larger Inmics Oy, based in Jyväskylä Finland. The company develops both custom tailored software for multiple clients and maintains its already existing products and services.

The parent company Inmics Oy contains multiple places of business all over Finland. Inmics Oy provides for instance, in addition to software development, hardware acquisitions for companies, hardware maintenance, education and its trademarked UpCare® service.

2. SERVICE-ORIENTED ARCHITECTURE

2.1 What is service-oriented architecture?

Service-oriented architecture (SOA) is a group of technologies, principles and guidelines for developing a collection of services that are capable of communicating with each other. The nature of this communication between any amount of services can vary from transferring different forms of data and information to coordinated activity between multiple services. Architectural model of the SOA is meant to make applications built around services more flexible, by making services the primary depository for logic and resources of the application. (Service-oriented architecture (SOA) definition.)

The basic model of SOA has existed for decades and is a flexible way of connecting different resources in a network, allowing the resources to be processed separately or at the same time. Service-oriented architecture is not restricted by different technologies, making it possible to create services that can be used by a wide range of applications across different programming languages and platforms. (What is SOA?)

The most common use for SOA is in the business environment, however, only when the architecture is properly utilized, will the business environment be able to use the full potential of SOA. In order to properly function in the service environment, a business function made to be used with the SOA must be able to function by itself, meaning that it does not require any functionalities from outside the service. (SOA Concepts.)

2.2 Client and server

In this thesis, there are multiple references to a ‘Client’ and a ‘Server’. These are two distinct parts of an application that communicate with each other through a service. Client-Server architecture is a network of clients, a computer for instance, connecting to a single or multiple server computers. These server computers, which are usually of larger capacities than the client computers, can contain both data and functionalities. What functionalities a server contains are specific to that server, a server com-

puter can, for instance, serve as a host for a database, contain applications to perform different tasks or give client access to the processing power of the server. The best example of the client-server architecture is the World Wide Web, which is used by web browsers as clients to access Web pages that are stored in servers.

Client is the part of the application that is available for the end user. This part consists of the user interface and all business logic that is related to it. The business logic of the client side usually contains the core application structure along with algorithms and functions related to it.

A server is the part of the application that contains the data layer, service and all business logic related to them. Both data layer and service can have their separate business logic and components. Contents of the server are usually more information related than the client's contents, such as database access, email service or printing, amongst other similar behaviors.

As illustrated by Figure 1, the server side is accessed by a client with a request. A request is a block of information that is sent from one object to another, in this case from a client to the server. Once the server has processed the data of the request, it will send it back to the client as a response, which contains information in a same manner as the request, but is going to the opposite direction. These requests and responses can be handled in the server side by for instance the WCF service.

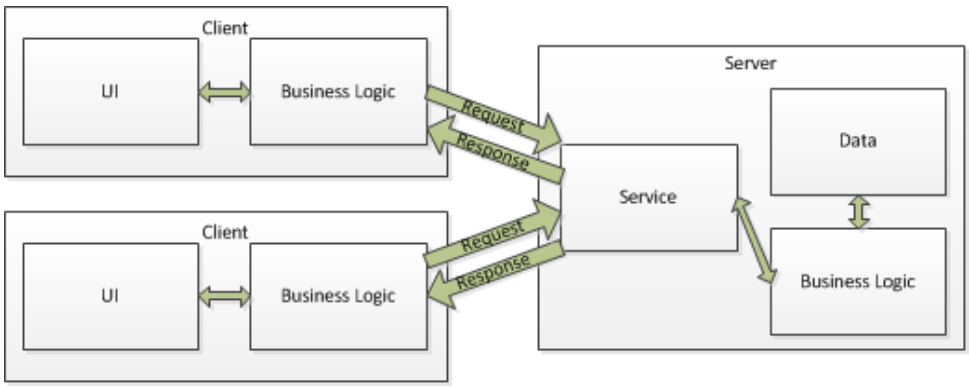

FIGURE 1. Server-client architecture and request/response behavior.

Of these two different sides, it is the server side that is more crucial in the final application structure. If the client side encounters a fatal error, only a single client in which the error occurred is affected. However, if the server side fails, all clients using the server cannot function properly. Without being able to send requests to the server, major functionalities of clients depending on the server are left impaired.

(What is Client-Server Architecture?)

3 WINDOWS COMMUNICATIONS FOUNDATION

3.1 Basics

WCF, short for Windows Communication Foundation, is a programming model from Microsoft for creation of service-oriented applications and is built using .NET framework. The service component of the WCF is in charge of making communication between different parts of applications or multiple different applications easy, by allowing to send information between different service endpoints.

One common use of the WCF is to provide a way to distribute and collect data and information to and from multiple clients at the same time, without affecting the user experience on the client side. Data can be given to the service, for instance storing information to a database, and after being received by the service, actions in the client side, such as terminating the application too early, can no longer affect the data storing process since that is handled by the server. In the same way, a client can ask a service for the information it requires and while the service is still retrieving it, the user can continue to use the application. Methods, such as the ones previously explained are referred to as asynchronous methods.

Since WCF is a flexible SOA protocol, it has been integrated with numerous other Microsoft technologies, such as Microsoft Silverlight, a platform that is used to create Web applications for developers of Web sites, has used WCF in connecting said applications to endpoints of WCF.

(What is Windows Communication Foundation.)

WCF can effectively be used with multiple application types and across different .NET technologies. In the Windows development network, following samples of these uses for WCF are given:

- *A secure service to process business transactions.*
- *A service that supplies current data to others, such as a traffic report or other monitoring service.*
- *A chat service that allows two people to communicate or exchange data in real time.*
- *A dashboard application that polls one or more services for data and presents it in a logical presentation.*
- *Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.*
- *A Silverlight application to poll a service for the latest data feeds.*

(What is Windows Communication Foundation?)

3.2 Contracts

In .NET, contracts are ways for programmers to specify different conditions for different parts of the code. For instance a contract can specify different pre- and post-conditions that must be met when execution of the code enters or exits a specific method or property. Contracts can also be used to describe object invariants in the code, meaning the expected state for a class that is in a good state.

Using of contracts is beneficial for the developer because, for instance, they enhance testing of the code by providing static contract verification, runtime checking and documentation generation. This also allows different automated testing tools to use contracts to generate more reliable and dynamic testing events.

A single property is not limited to one contract. The programmer can specify as many contracts as needed for a single item. Since there are multiple different types of contracts with multiple different meaning and effect, a great amount of combination of different conditions can be specified for a single item. Some contracts, like “Browsable” for example, can be used to determine how a property is shown to an end user,

with the example “Browsable” with Boolean parameter “false” will cause the contract to hide the property, while some other contracts can be used to describe how a class interacts with other classes.

In WCF, most commonly used contract types are data contracts, service contracts and operation contracts. From Microsoft Developer Network, definition of a data contract is as follows:

> *A* data contract *is a formal agreement between a service and a client that abstractly describes the data to be exchanged. That is, to communicate, the client and the service do not have to share the same types, only the same data contracts. A data contract precisely defines, for each parameter or return type, what data is serialized (turned into XML) to be exchanged.*
>
> (Using Data Contracts.)

These data contracts are applied to multiple parts of the WCF service. For instance, when defining new methods the service can run, the use of data contracts is vital to give a good understanding of what kind of data the method can handle and for example, in what ways the method can fail. To indicate these possible exceptions for a method, one can define both pre-existing and custom service faults via the contract.

When data is passed to and from the service via objects, it is vital that the object is defined as a service contract with all properties the values of which are to be transferred from client to service (or other way around) be marked with appropriate contracts. If a property does not have a contract that defines the property as a member of the data contract, it will be transported as an empty or default value of that property type. These data members can be both pre-existing types, like string or int, or custom classes, for instance another data contract class. The following code snippet shows the data contract in action, with namespace it is derived from being the System.Runtime.Serialization.

(Using Data Contracts.)

```
using System.Runtime.Serialization;

namespace DataContractExample
{
    /// <summary>A class that is marked as a data contract.</summary>
    [DataContract]
    public class DataContractExample
    {
        /// <summary>A string that is marked as a data member.</summary>
        [DataMember]
        public string Name { get; set; }

        /// <summary>Integer that is marked as a data member.</summary>
        [DataMember]
        public int ID
        {
            get { return id; }
            set { id = value; }
        }

        /// <summary>
        /// Id field that does not have a contract
        /// but will have its value treated as a data member
        /// since ID -property is a data member.
        /// </summary>
        private int id;
    }
}
```

Service contracts are used to define the service and determine how the service behaves and how it communicates with the client. A good example of these definitions are specifications of single client-service sessions. With the service contract, client can create a new instance of the service client for every single service related action, or be made to use a single instance for the whole lifetime of the single client.

Operation contracts are used like data contracts and they define how different methods and functions under the service behave. These operation contracts are usually used inside a service defined by service contracts and, for instance, inserted into methods a service. By using the operation contracts with these methods, these methods become a part of the service and can be called from the client.

All contracts are classes, having a constructor and possible methods and properties. These properties, or as they are called by Visual Studio, “Named Parameters”, can alter the contract in various ways, like in the case of “Data Contract”, these properties can be used to specifying name, namespace and type, as shown with name and namespace below.

```
[DataContract(Name = "DataContractExample", Namespace = "DataContractExample")]
```

3.3 Endpoints

All WCF services require endpoints to work. Endpoints are used to specify in what level a client can connect to the service, and how this connection will be made. Without endpoints, a client could not connect to the service at all.

Three crucial parts of endpoints are:

- an address, where the endpoint is located at
- a binding, how to access the endpoint via a channel
- a contract, what the operation of an endpoint is and how to communicate with it.

(Pawel Jarosz, 2011.)

All endpoints have an address, the location of the endpoint. Endpoint address can be given via code or configuration file, latter being more commonly used since this address can change, making changing of the address for former a lot more inconvenient (see the following configuration file and code behind examples.). This address is where a client connects, when it is communicating with a service the endpoint belongs to.

A binding is a set of data and protocols that determines how the endpoint is communicated with. The binding contains information of the transport protocol and encoding to use as well as the security requirements for the endpoint. Common transport protocols are TCP or HTTP, which determine how requests and responses behave. Common encodings are text and binary, which determine in which form the information is received and sent by the endpoint. Security requirements can be for instance SSL.

In addition to address and binding, endpoints require a contract. This contract is to determine what functionalities the endpoint exposes to the client. Microsoft development network specifies these functionalities as:

- What operations can be called by a client.
- The form of the message.
- The type of input parameters or data required to call the operation.
- What type of processing or response message the client can expect.

(Endpoints.)

Endpoints can also be assigned several behaviors. These behaviors usually affect different properties of the endpoint.

Creation of an endpoint with an arbitrary address, binding using “basicHttpBinding” protocol and a contract reference to an interface called “ITestService” is shown with both a settings file and a code file in the code examples below.

```
<endpoint address="/Path/Address"

          binding="basicHttpBinding"
          contract="TestService.ITestService">
```

```
/*
 * Creating the endpoint properties.
 */
Uri address = new Uri("/Path/Address");
BasicHttpBinding binding = new BasicHttpBinding();

/*
 * Creating the service host and assigning it the endpoint.
 */
ServiceHost host = new ServiceHost(typeof(TestService), address);
host.AddServiceEndpoint(typeof(ITestService),
                        binding,
                        address);
```

3.4 Service

What is a service?

A service allows creation of data layer that is completely independent, meaning that it does not rely on specific objects or behaviors. It can be accessed by different applications across different platforms with the help of contracts. Even though the class structure of an application may vary from the service's class structure, having common contracts allows data being transferred from one to another.

There are multiple service types, with .NET specific WCF service being only one of them. WCF service is Web based and can be located both locally with the application and separately in the web or local network. There can be multiple clients connecting to the service simultaneously, as a single client can also connect to multiple services at once. The hosting of the WCF service uses the same technology than web hosting in the World Wide Web.

Creation of the service is generally handled with three different main components:

- An interface that specifies core methods and properties for the service, along with contracts for said items (See the code sample below.). This is, however, only for Best Practice implementation, since the contracts can be specified inside a single service class as well.
- Service class, implementing the service interface.
- Configurations file specifying how the service will work.

For instance, using Microsoft Visual Studio 2012 to create a service project, these items will be automatically generated. Once the service has been created, it can be hosted with multiple ways, as long as the necessary endpoints and service references are specified by the host.

Below is shown a code block with interface of a service with single method declared as an operation. Used class, "DataContractExample", has been declared as a data contract inside its own class.

```
/// <summary>Core interface of the test service.</summary>
[ServiceContract]
public interface ITestService
{
    /// <summary>Get list of all DataContractExamples.</summary>
    /// <returns>All data contract examples.</returns>
    List<DataContractExample> GetDataContracts();
}
```

Benefits of using a service

When comparing a service to more traditional methods, the usefulness of the service depends heavily on functions that are related to the data of the final product. When an application is intended to be used by multiple users simultaneously, sharing a data source, the service will be a great deal more suitable choice than more regular local data handling methods. On the other hand, if the application has only one user using one data source, or only one user using it at a time, implementing a service may be a bit excessive.

The service can also be the best choice when an application has potential to spread to multiple platforms, since the service is accessible from multiple platform types. With the service, a multiplatform application can communicate with its intended data layer through the same methods and even communicate between its different platforms.

However, the need to be hosted by a server is also the greatest weakness of the service. If the host fails, all applications using the service lose a major part of their functionality until the server is brought up again. This can be avoided, or at least damage caused by the service's loss be lessened, by using multiple services or alternative data access methods as a backup.

4 IMPLEMENTATION

4.1 Background

The application the thesis is based on was a working hours and costs tracking system for a client of the Inmics Software Engineering Oy. The application had multiple user levels, possible working environments and projects for users. Basic users could use the application to see what they have been assigned to do and register their working hours within a timeframe of their own choice. More advanced users could plan and organize the workloads and costs of personnel and monitor how these plans were realized, along with other similar functions.

The structure of the project consisted of:

- client side, with user interface created with WPF and client related business logic
- server side with data layer, service, service hosts and business logic for all mentioned
- whole application wide business entities and resources.

The project used Entity Framework to handle database access, and custom made data transfer objects, different from entities created by the entity framework in the rest of the application. The reason for using these instead of automatically generated versions by the Entity Framework was to make them more accessible by the service via the contracts and since the client would never be directly in contact with the data layer, custom entities would have been required to be created for client side. Resources of the project consisted of text based dictionaries to allow multi-language implementation of the application. The structure of the application is illustrated in FIGURE 2.

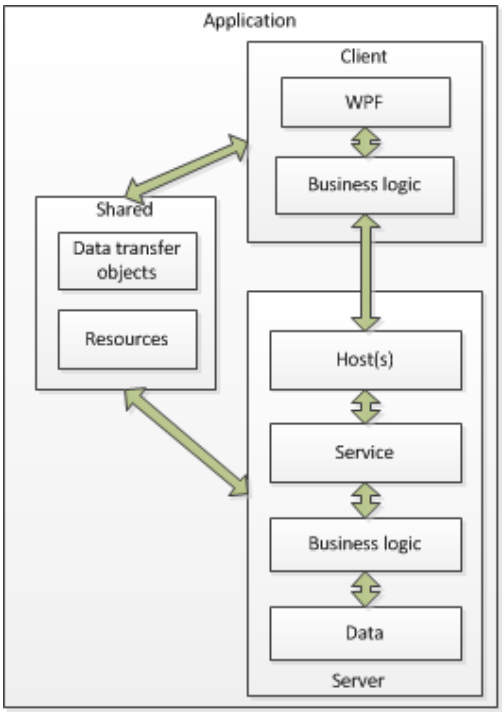

FIGURE 2. Structure of the application.

Users of the application were required to be logged in to use the application. This login was made automatically when a user started the application, using login information from the operating system, or if this fails, manually by entering a domain, username and password. Usernames and passwords were not handled by the application, but were retrieved from already existing Active Directory services, where the said users already existed with their passwords.

The implemented application presented in this section is not the actual application this thesis is based on, but rather an application made after the actual one. The decision to do the implementation in this way was to make sure that there would not be any sensitive information included with the example.

The implementation is a WCF service that has a single method to get a list of users. This method was chosen since it gives a good understanding of how the service works and every single service based method in the actual application can be created using the same syntax used with the example.

4.2 Data transfer objects

During the development process, the first step was to create data transfer objects that would be application wide and shared by both the client and the service. These objects were introduced as “Data Contracts” with all their properties as “Data Member”. This was done because these entities were passed with requests and responses.

When creating a service model, these shared objects are not a necessary step, since they could be replaced by another object that only shares contracts. This approach was useful because both client and server side were part of the same solution and thus could both access these objects.

The creation of data transfer objects was very straight forward object oriented programming. They were created as any object would, with addition to assigning contracts for each of them. As illustrated by the following code sample, the user contains different “Data Member” properties, specifying information about the user.

User objects were given a Guid hash that will be generated for each user individually during a log in process and a copy of this hash will be saved to the database. This hash serves as an identifier for the current use session of the application and is used by the server to verify that the user of the application is currently logged in. This verification is carried out every time a request is made. If a user logs out, this hash is removed from the database, but if the user closes the application without logging out, the hash will remain and in both cases, when the user logs in the next time, a new hash is generated.

Other entities of the same level as User were made using the same formula, with a similar ID field for all of those entities that were based on database tables with a numeric identifier as the primary key.

```
/// <summary>Represents a single user in the application.</summary>
[DataContract]
public class User
{
    #region CONSTRUCTORS

    /// <summary>Create a new implementation of the user.</summary>
    public User()
    {
        this.ID = 0;
        this.Name = string.Empty;
        this.Hash = Guid.Empty;
    }

    #endregion CONSTRUCTORS

    #region PROPERTIES

    /// <summary>Unique identifier of the user.</summary>
    [DataMember]
    public long ID { get; set; }

    /// <summary>Name of the user.</summary>
    [DataMember]
    public string Name { get; set; }

    /// <summary>
    /// GUID used to authenticate requests
    /// by logged in users.
    /// </summary>
    [DataMember]
    public Guid Hash { get; set; }

    #endregion PROPERTIES
}
```

4.3 The service

The service itself was created in two parts; the first part being an interface that declares contracts, methods and properties of the service and the second part being the implementation of that interface.

Before creating the interface, custom exception handling classes were made. These classes would help to identify the nature of possible exceptions and how these exceptions occurred. All of these custom exceptions can be seen in the code snipped below.

Two of these custom exception classes were Service- and Authentication- faults, which are shown in the following code example. These faults could be triggered by exceptions during the execution of the service methods and store all possible information of that exception. When setting these faults to be possible faults for meth-

ods, they were introduced as implementations of “FaultContract”, as seen later in the code sample of the service interface, the fault contract being derived from SOAP protocol.

```
/// <summary>Fault describing a generic error during a request.</summary>
[DataContract]
public class ServiceFault
{
    /// <summary>
    /// Create a new implementation of the service fault.
    /// </summary>
    /// <param name="error">Exception that occurred.</param>
    public ServiceFault(Exception error)
    {
        this.Error = error;
    }

    /// <summary>Get/set exception that occurred.</summary>
    public Exception Error { get; set; }
}

/// <summary>
/// Fault describing an error during
/// authentication of a request.
/// </summary>
[DataContract]
public class AuthenticationFault
    : ServiceFault
{
    /// <summary>
    /// Create a new authentication fault.
    /// </summary>
    /// <param name="error">Exception the fault is for.</param>
    public AuthenticationFault(Exception error)
        : base(error)
    {
        this.Error = error;
    }
}
```

The benefit of using the fault contracts was that, they allowed exceptions to be transferred from a server to a client. Without these contracts, a server would not be able to send detailed information of the exception to a client.

If an error occurred in the service, a new implementation of the third custom exception, Service Exception, was created, with its constructor capable of determining which one of the previously mentioned fault types the exception was. This “ServiceException” would then be passed to the client to indicate the failed request and

pass the information about the exception forward. The implementation of this “ServiceException” is shown by the following code snippet.

```
/// <summary>Exception that has occurred inside the service.</summary>
public class ServiceException
    : Exception
{
    /// <summary>
    /// Create a new implementation of the
    /// service exception.
    /// </summary>
    /// <param name="exception">
    /// Exception that occurred during the
    /// execution of the service.
    /// </param>
    public ServiceException(Exception exception)
    {
        this.Exception = exception;
    }

    /// <summary>
    /// Get/set exception that occurred
    /// during execution of the service.
    /// </summary>
    public Exception Exception { get; set; }
}
```

After the exceptions were created, an interface containing all methods with their contracts for the service was made (see the following code sample.). This interface would also serve as the target of the contract property for endpoints of the service, as will be illustrated later.

```
/// <summary>Interface for the service.</summary>
[ServiceContract]
public interface IService
{
    #region METHODS

    /// <summary>Get list of all existing users.</summary>
    /// <param name="guid">Used to authenticate the user.</param>
    /// <returns>List of all existing users.</returns>
    [FaultContract(typeof(AuthenticationFault))]
    [FaultContract(typeof(ServiceFault))]
    [OperationContract]
    List<User> GetUsers(Guid guid);

    #endregion METHODS
}
```

The service itself contained the implementation for the created service interface. In addition to the interface, the service contained a contract that determined its behavior. With this contract, the service was made to allow multiple threads at the same time and allow the service be a new instance for every session. The service would pass properties it received via the request to business classes to be handled (see the following code block.).

```
/// <summary>Service implementing IService.</summary>
[ServiceBehavior(UseSynchronizationContext = false,
 ConcurrencyMode = ConcurrencyMode.Multiple,
 InstanceContextMode = InstanceContextMode.PerSession)]
public class Service
    : IService
{
    #region INTERFACES

    /// <summary>Get list of all existing users.</summary>
    /// <param name="hash">Used to authenticate the user.</param>
    /// <returns>List of all existing users.</returns>
    public List<Business.Entities.User> GetUsers(Guid hash)
    {
        try
        {
            // Getting users with business logic.
            return UserComponent.GetUsers(hash);
        }
        catch (Exception exception)
        {
            // Throwing the service exception.
            throw new ServiceException(exception);
        }
    }

    #endregion INTERFACES
}
```

4.4 Hosting the service

The service, once created, was hosted as a web service. The web service was based on ADO.NET project that contained the service file (.scv) and web configuration file (web.config) that detailed the service.

There are of course other ways to host the service as well. Hosting the service as part of the application is only optional and can be hosted separately. For instance, as done during the development of the application in the test environment, the service was hosted inside a design time address via a small console application.

The service inside the service file is only a wrapper class that inherits from the previously created service class. Since the wrapper class did not bring any changes to the service itself, the previously created contract class was referenced in the configuration file. The wrapper class was used to allow the service to be used as a web service, which the original one was not. The first of the following code examples is the Web Service as defined in the WebService.svc file while the second one is the code behind of the WebService file.

```
<%@ ServiceHost Language="C#"
                Debug="true"
                Service="Hosts.WebService.WebService"
                CodeBehind="WebService.svc.cs" %>
```

```
/// <summary>Web implementation of the service.</summary>
 public class WebService
     : Service.Service
 {
    // This is only a wrapper class.
 }
```

The service is defined in Web.Config file (see the following code snippet.) under "configuration" –tag. The main tag created was "system.serviceModel". With this tag, the application reading the configuration knows that service definitions have started.

The first thing to do was to create a binding configuration the service would use and give appropriate properties for that binding to make the service work as intended. In binding shown by the following snippet, a "WsHttpBinding" was chosen as the type of the binding. This binding is part of "System.ServiceModel" namespace and cannot be used by .NET application below .NET version 3, however, it offers more advanced security options for request and response operations than more accessible "BasicHttpBinding". While the basic version does not provide any security for client – server communication, the "WsHttpBinding" has this security enabled by default.

The properties were given for the binding. Firstly a name, which is required in order to use the binding later, and secondly a close timeout, which in turn would close

down the service once given time had passed without successful response being forwarded. There are of course numerous other properties, some specific to binding types and other shared by some types.

The behavior for the service was also defined with the custom service behavior. This behavior was given name, so it could be later referenced by the service itself. With the behavior, metadata and debug information were enabled by setting respective attributes to “true”. The benefit of enabling these attributes was mainly for development purposes only, since they enable more efficient debugging of the service. In general, these attributes should be disabled when an application is published.

The next item to define was the service itself. The service was defined under “Services” –tag, which can contain definitions for multiple services, since a single project is not limited by the amount of services it can have.

The name given for the service was the name of the hosting web service class with a full namespace. The previously created behavior was given for the service and after this, an endpoint was created. All though in the given example there is only one endpoint, a service can have multiple endpoints specified.

The address of the endpoint was set to local host to ease the testing of the application. This address can, however, be practically anywhere the application has access to, be it in a local network or behind a WWW-address.

The binding for the endpoint was of the same type as the previously created configuration and both the binding type and the name of the configuration were referenced in the endpoint.

The last item to define for the endpoint was the contract, which was the interface shown in the second to last code snippet in the service section of the implementation part. For the contract attribute, full namespace and interface name was given.

After the service was created, another tag under “configuration” tag was also made. This tag, called system.webServer, was made to ease debugging of the service and control what information of the server would be made public.

```
<system.serviceModel>

  <!-- Bindings -->
  <bindings>
    <wsHttpBinding>
      <binding name="wsBindingConfiguration" closeTimeout="00:01:00" />
    </wsHttpBinding>
  </bindings>
  <!-- /Bindings -->

  <!--Behaviors-->
  <behaviors>
    <serviceBehaviors>
      <behavior name="serviceBehaviorConfiguration">
        <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
        <serviceDebug includeExceptionDetailInFaults="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <!--/Behaviors-->

  <!-- Service -->
  <services>
    <service name="Hosts.WebService.WebService"
             behaviorConfiguration="serviceBehaviorConfiguration">
      <endpoint address="http://localhost:1571/WebService.svc"
                binding="wsHttpBinding"
                bindingConfiguration="wsBindingConfiguration"
                contract="Service.Contracts.IService"/>
    </service>
  </services>
  <!-- /Service -->
</system.serviceModel>

<system.webServer>
  <modules runAllManagedModulesForAllRequests="true"/>
  <directoryBrowse enabled="true"/>
</system.webServer>
```

4.5 Using the service in client

Now that the service was created and configured, it was ready to be implemented in the client. The first thing to do was to reference the web service. In Visual Studio 2012, this is done by right clicking a project in the client side and selecting the “add service reference” –option. From the window that opened, the button “Discover” was clicked, which brought the service to the services list from which the service was selected, as shown in FIGURE 3.

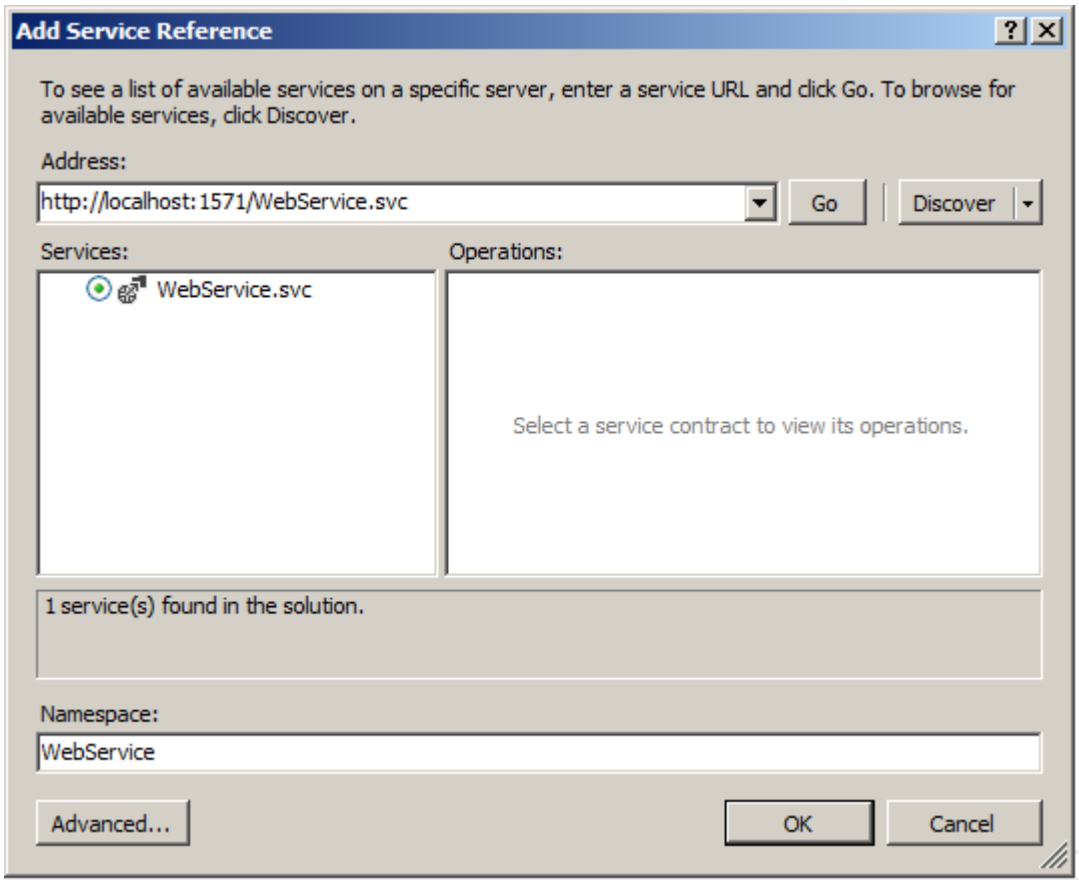

FIGURE 3. Adding a service reference in Visual Studio 2012.

Before accepting the service, few modifications were made for its settings. First of all, the namespace was replaced with a more describing name, since it would be through this namespace the service would be used when coding the client. All classes the service reference would generate were automatically inserted into this namespace.

Then, through advanced settings the service was made to generate asynchronous operations by checking the respective check box and selecting Generate asynchronous operations radio button (see FIGURE 4.). If this was not selected, the client would have been unable to perform asynchronous requests. The collection type was set to a generic list. The collection type and dictionary type settings define in which form these data types were generated by the service reference.

All other settings were left to their default state and the service reference and all of its related components, including service references in .config file of the client, were automatically generated by Visual Studio to the given namespace.

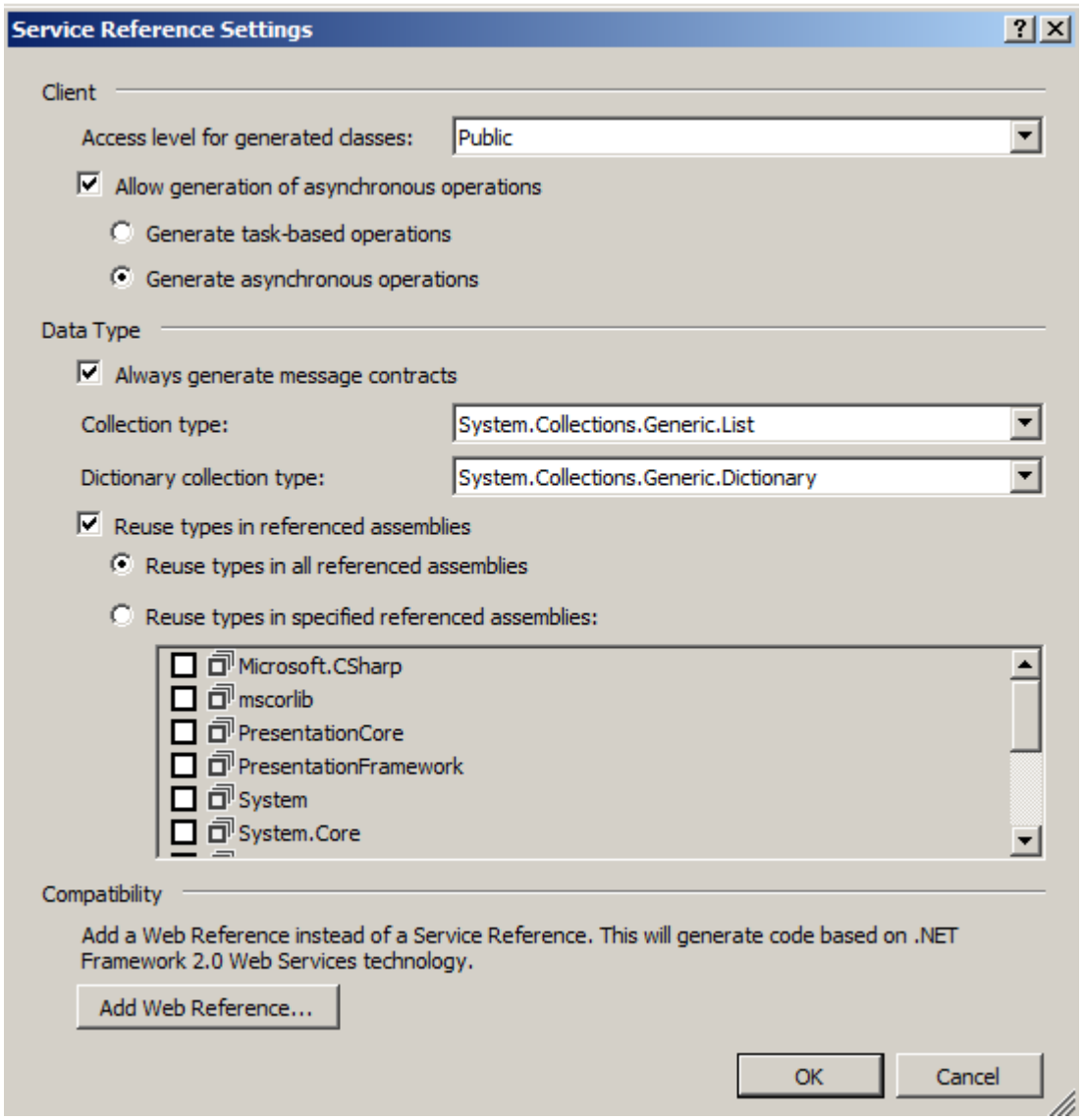

FIGURE 4. Configuring the service reference in Visual Studio 2012.

Since both the client and the server of this application shared data transfer business entities, these entities were not automatically generated to the namespace of the web service.

After adding the service reference, using the service is done using the automatically generated service client object. Since the method is used asynchronously, an event handler for the response should be introduced- as illustrated below.

```
/// <summary>Get list of all users asynchronously.</summary>
public void GetUsersAsync()
{
    WebService.ServiceClient proxy = new WebService.ServiceClient();
    proxy.GetUsersCompleted += this.proxy_GetUsersCompleted;
    proxy.GetUsersAsync(
        new WebService.GetUsersRequest(this.CurrentUser.Hash));
}
```

After the request has been completed, this event is triggered. It is through the event arguments of this event, that the results of the response can be read. These automatically generated event arguments contain information of an possible error during the service operation and other possible conditions, such as was the asynchronous method cancelled. Since the method created for the service had a return type, the event arguments also contain a result property, which in turn contains the actual result- as illustrated by the following code snippet.

In addition to handling the results, the proxy is manually disposed by closing both the proxy itself and its “InnerChannel” property. Disposing the proxy makes sure that the connection to the service is cut and no longer counts towards the limit of concurrent open connections the service may have.

```
/// <summary>Forward received users.</summary>
/// <param name="sender">Not used.</param>
/// <param name="eventArgs">
/// Contains the response of the server operation.
/// </param>
protected void proxy_GetUsersCompleted(object sender,
    WebService.GetUsersCompletedEventArgs eventArgs)
{
    try
    {
        try
        {
            /*
             * Disposing the proxy.
             */
            WebService.ServiceClient proxy
                = sender as WebService.ServiceClient;
            proxy.InnerChannel.Close();
            proxy.Close();

            if (eventArgs.Error != null)
            {
                /*
                 * Forwarding exception if an error
                 * occurred.
                 */
                throw eventArgs.Error;
            }
            else if (!eventArgs.Cancelled)
            {
                /*
                 * Forwarding received users if
                 * the request wasn't cancelled.
                 */
                this.OnUsersReceived(eventArgs.Result.GetUsersResult);
            }

        }
        catch (Exception exception)
        {
            this.OnErrorOccurred(exception);
        }
    }
    catch (Exception) {   }
}
```

5 RESULTS AND CONCLUSION

5.1 Results

A great deal of good documentation was found for the theory part and all major components of the service were introduced. Most of the documentation was from the Microsoft Developer Network, which contained a great amount of information about most major Microsoft related frameworks. Since the implementation part was

also meant to serve as a tutorial for how to implement a WCF service, the implementation part benefitted from the theory for introducing all major parts of the service so the contents of the code examples in the implementation part had already all of their crucial parts explained.

Downside of the WCF service is that when, for any reason, the service goes down, all clients depending on the service becomes impaired until the service has been brought back up again.

Overall, the implementation of the service went smoothly and the created service functioned as intended, following the desired Service-oriented architecture. The most noteworthy problem was with using the service in the client side, when after several successful requests, the service suddenly stopped responding. This was caused by the service client proxy that was used to access the service. After asynchronous method of the proxy ended, it was not properly closed and once enough instances of it were created, the service reached its limit of allowed simultaneous connections. The solution to this was, after the asynchronous method was over, to manually close both the proxy and its inner channel, since these inner channels are actively used when connecting to the service.

5.2 Conclusion

WCF service is one of the most noteworthy ways to implement an client-server application in .NET environment. It allows effective separation of the server side and everything inside the server side from the client side. Client-server application using WCF service can have both a single or multiple clients connecting to a single or multiple services.

The use of a service is not restricted to a single application and can be used across multiple platforms as well. Since the WCF service uses contracts to identify data and methods, multiple application types, using the same or different programming language, can access the service because data types and other useful information can be identified from the contracts.

The WCF service can easily be implemented with few simple steps, the basic structure of which can be automatically generated by Visual Studio 2012. The most important parts left for a programmer to do are to specify service configurations and assign correct contracts to methods, properties and classes, to be used by the service. If the WCF service is done using the best practice and up to SOA standards, the contracts of the service should be separated into their own interface, which the actual service implements.

REFERENCES

What is entity framework. n.d. Microsoft Developer Network, Data Developer Center. Referenced 12.10.2012. http://msdn.microsoft.com/en-us/data/ef.aspx

What is Windows Communication Foundation, n.d. Windows development network, library, referenced 13.10.2012, http://msdn.microsoft.com/en-us/library/ms731082.aspx

Endpoints, n.d. Windows development network, Library, referenced 14.10.2012, http://msdn.microsoft.com/en-us/library/ms733107.aspx

Using Data Contracts, n.d. Windows development network, library, referenced 16.10.2012, http://msdn.microsoft.com/en-us/library/ms733127.aspx.

What is Transport Layer Security, wiseGEEK. Referenced 4.12.2012. http://www.wisegeek.com/what-is-transport-layer-security.htm

What are Best Practices?, n.d. wiseGEEK. Referenced 4.12.2012. http://www.wisegeek.com/what-are-best-practices.htm

So what is Active Directory? (Windows). n.d. Windows development network, referenced 30.11.2012. http://msdn.microsoft.com/en-us/library/windows/desktop/aa746492%28v=vs.85%29.aspx

What is Client-Server Architecture?. n.d. wiseGEEK, referenced 30.11.2012. http://www.wisegeek.com/what-is-client-server-architecture.htm

Pawel Jarosz, Cracow University of Technology, Lecture, Jyväskylä 2011

What is SOA? n.d. wiseGEEK, referenced 7.12.2012. http://www.wisegeek.com/what-is-soa.htm

Douglas B, n.d. Service-oriented architecture (SOA) definition. Service-architecture.com, referenced 7.12.2012. http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

SOA Concepts. 2007 Exforsys Inc®, referenced 8.12.2012. http://www.exforsys.com/tutorials/soa/soa-concepts.html