Candidate name: Thor Halvor Frivold
Subject: TDT4735 Software Engineering

Project title:

Dynamic service context and underlying platform and service mechanisms

Assignment given: August 2005
Project deadline: 20th December 2005
Project delivered: 20th December 2005
Subject teacher: Alf Inge Wang - NTNU
Supervisor: Josip Zoric – Telenor R&D

Trondheim, 20th December 2005
Abstract

This report will examine Service Oriented Architecture (SOA), Web Services, the .NET platform and core service mechanism such as context awareness, service discovery, service composition, service brokering and service mediation. These mechanisms are identified as essential to meet the next generation of dynamic services. - Services who exploit the full potential of networks, platforms and technologies.

I start with a brief description of the mechanisms and then I will take a deeper look into the state of art. Java and .NET will be compared and a description of what a “static service” is versus a “dynamic service” including a gap analysis between them will be given. These gaps will then be described and evaluated to better figure out how to make the bridge between them.

A proof of concept architecture and necessary entities and modules will be proposed. Use Cases from the both the system’s and the clients’ point of view will also be suggested. This will give a better understanding of how the data flow could be implemented.

Keywords: Dynamic service, architecture, SOA, Web Service, .NET
Preface

This report is written as part of the course TDT4735, Software Engineering, Depth Study, which is part of the Masters degree at the Norwegian University of Science and Technology.

I would especially like to thank Josip Zoric for his guidance in writing this report. It has been of invaluable help to me to hear his viewpoints and benefit from his experience. I will also like to thank Njal Arne Gjermundshaug who has been a good study friend the past months. We have had good discussions and effective brainstorms in many of the report’s topics.

Trondheim, December 2005

___________________________
Thor Halvor Frivold
# Table of Contents

ABSTRACT ......................................................................................................................... III

PREFACE .......................................................................................................................... V

TABLE OF CONTENTS ....................................................................................................... VII

LIST OF FIGURES ............................................................................................................. IX

1 INTRODUCTION ........................................................................................................... 1
  1.1 MOTIVATION ............................................................................................................... 1
  1.2 CONTEXT ................................................................................................................... 2
  1.3 SCOPE ......................................................................................................................... 3

2 CORE SERVICE MECHANISMS .................................................................................... 4
  2.1 SERVICE ................................................................................................................... 5
  2.2 CONTEXT MANAGER ................................................................................................. 5
  2.3 SERVICE DISCOVERY ............................................................................................... 6
  2.4 SERVICE COMPOSITION ........................................................................................... 7
  2.5 SERVICE BROKERING ............................................................................................... 7
  2.6 SERVICE MEDIATION ............................................................................................... 8

3 SERVICE ORIENTED ARCHITECTURE ....................................................................... 10
  3.1 ARCHITECTURE IN GENERAL ............................................................................... 11
  3.2 SOA BACKGROUND ................................................................................................. 11
  3.3 SOA MAIN CONCEPTS ............................................................................................ 12
      3.3.1 Service in SOA ................................................................................................. 12
      3.3.2 Messages .......................................................................................................... 13
      3.3.3 Dynamic discovery ........................................................................................... 13
  3.4 SOA REFERENCE MODEL ....................................................................................... 14
  3.5 GENERAL SERVICE COMPOSITION MODEL (GSCM) ............................................ 14

4 WEB SERVICES ........................................................................................................... 17
  4.1 CHARACTERISTICS ................................................................................................. 18
  4.2 TECHNOLOGIES ....................................................................................................... 18
      4.2.1 SOAP ................................................................................................................. 19
      4.2.2 Web Service Description Language (WSDL) .................................................... 20
      4.2.3 Universal Description, Discovery and Integration (UDDI) .............................. 21
  4.3 WEB SERVICE INTERACTION ................................................................................. 23
  4.4 THE GENERAL SERVICE COMPOSITION MODEL AND WEB SERVICES ............ 24
      4.4.1 Nss-1 ............................................................................................................... 25
      4.4.2 Nss-2 ............................................................................................................... 27
      4.4.3 Nss-3 ............................................................................................................... 28
      4.4.4 Nss-4 ............................................................................................................... 29
      4.4.5 Nss-5 and Nss-6 ............................................................................................. 29

5 .NET ............................................................................................................................. 30
  5.1 FRAMEWORK .......................................................................................................... 30
      5.1.1 CLR and CTS ................................................................................................... 31
  5.2 .NET WEB SERVICES ............................................................................................... 33
  5.3 GSCM IMPLEMENTED USING .NET WEB SERVICES ........................................... 33

6 J2EE AND .NET WEB SERVICES .............................................................................. 36

7 SMART CLIENT ........................................................................................................... 38

8 STATIC AND DYNAMIC SERVICES ....................................................................... 39
  8.1 TODAY’S STATIC SERVICES ..................................................................................... 39
  8.2 TOMORROW’S DYNAMIC SERVICES ..................................................................... 41
1.1 Motivation

9 PROOF OF CONCEPT ARCHITECTURE ........................................................................43
9.1 SCENARIO ..................................................................................................................43
9.2 REQUIREMENTS .......................................................................................................43
9.3 USE CASES ..............................................................................................................44
  9.3.1 Use Case from the Customer’s point of view .....................................................44
  9.3.2 Use Case from the System’s point of view .......................................................47
  9.3.3 Database view ....................................................................................................50
  9.3.4 Sequence Diagram ............................................................................................51
10 CONCLUSION ..............................................................................................................54
11 FURTHER WORK ........................................................................................................56
12 GLOSSARY ................................................................................................................57
13 REFERENCE LIST .......................................................................................................58
APPENDIX A GETNEWEVENT WEB SERVICE SOURCECODE ....................................61
APPENDIX B GETNEWEVENT WSDL ..........................................................................62
APPENDIX C GETNEWEVENT WEB PROXY ...............................................................63
List of Figures

Figure 1 - The complex computer system in use by Telenor ASA .......................... 2
Figure 2 - Main service mechanisms .................................................................. 4
Figure 3 - Directory service .............................................................................. 12
Figure 4 - SOA Reference model ....................................................................... 14
Figure 5 - General Service Composition Model .............................................. 15
Figure 6 - The core Web Service Technology Stack ......................................... 19
Figure 7 - UDDI Services ................................................................................. 22
Figure 8 – The interaction of a Web Service ....................................................... 23
Figure 9 - Core functionality of Web Services ................................................... 25
Figure 10 - Nss-1 version 1 ............................................................................... 26
Figure 11 - Nss-1 version 2 ................................................................................. 27
Figure 12 - Context composer example .............................................................. 28
Figure 13 - Service broker example ................................................................... 28
Figure 14 - Service mediator example ................................................................ 29
Figure 15 - Components of the .NET Framework ............................................. 30
Figure 16 - .NET Framework Class Library ...................................................... 31
Figure 17 - Common Language Runtime ............................................................ 32
Figure 18 - Enhanced Web Services using .NET ................................................ 34
Figure 19 - Static message service ..................................................................... 39
Figure 20 - Soccer Web Service ......................................................................... 40
Figure 21 - Evolution of WebServices ................................................................. 42
Figure 22 - Use Case in Customer's point of view .............................................. 44
Figure 23 - Proof of Concept DatabaseView ..................................................... 51
Figure 24 - Example of a Sequence Diagram for the Proof Of Concept ............. 52
1 Introduction

This introduction section will briefly state the main objectives and motivation related to the project I have been assigned. Further I will explain the project context and the scope of the report. Finally I will give a presentation of the report outline, and what the different chapters contain.

1.1 Motivation

Today, millions of businesses and users communicate with each other using various gadgets and networks. Distributed applications (applications which communicate over a network) have not only been successfully adopted by many businesses, they have changed everyday life as we know it. Many of the distributed applications have a Service Oriented Architecture (SOA), which means that the applications provide/use/share services as needed. A service is one or more tasks that are executed or processed when invoked. A simple example is a temperature service. Users can invoke the service using e.g. a mobile phone or through a web site. The clients invoke the service specifying a location, the service does its work (clients rarely care how the service looks under the hood), and a temperature value is returned to the client. This allows for reuse of code, one of the infamous best-practice principles of software engineering. If this principle is taken to the next level, we’re talking about reuse of functionality. If we combine reuse of functionality and SOA, we get a scenario where services are built on top of each other, based on what functionality they provide / require. This activity is called Service Composition, and receives a lot of focus from software researches and developers around the world. Composing Services manually is not rocket science and has been implemented in applications for years (Metz, 2001). The ultimate goal is to make the service composition happen automatically at run time, or at least with as little work as possible. Automatic Service Composition would allow for a new generation of services and SOA. The advantages of such a system are many; e.g. machines can automatically discover services and deliver a vast amount of different services (combinations) to users. Clients can customize and tailor services to a greater level than today. The vision of service composition does not only mean more, better, more usable and richer services, it also means huge savings and very cheap services; the composition is done by machines and code(functionality) can be reused to a greater extent.
1.2 Context

Telenor ASA is a telecommunications company providing various services and targeting a diverse range of devices and audiences. The main business areas of Telenor are cell phone networks, Internet access and satellite television. These services are implemented using a wide range of different IT systems, interfaces, languages and network technologies. An illustration of a few of these systems and services (along with some projects) is shown below by Telenor chief software architect Sigurd Thunem.

![IS/IT Architecture Diagram](image)

**Figure 1 - The complex computer system in use by Telenor ASA**

Telenor has in fact over 470 IT systems (just counting the ones in Norway), with hundreds of various services for both public and internal use. A service composition system as the one described above would be of great value to Telenor ASA, and other service providers. Paolo Malinverno, chief researcher at Gartner Group predicts that SOA still will be used for several decades in all kinds of industries and applications (Hysing, 2005). The adviser of this thesis has been Josip Zoric at Telenor R&D. Zoric has, through several EU project such as Daidalos and Prince, identified that Service Composition require components such as:

- Context Manager - Handles context info, primarily for users.
• Service Composer - Composes services by interacting with the components below.
• Service Discovery - Discovers services.
• Service Broker - Takes care of brokering between services.
• Service Mediator - Mediates between services if conflicts arise.

Implementing a system to handle the service composition is difficult, the different components mentioned above needs to be fairly intelligent and flexible. Intelligence is perhaps the quality attribute of software that is hardest to realize. Some compare service composition to the trade of alchemy; they say it’s impossible, at least with the current technology and knowledge of computers and software.

This report is meant to by an analysis in depth and a foundation for future work, and a Master thesis during the spring 2006. See own chapter for future work.

1.3 Scope

In this project I will focus on technology used in Service Oriented Architecture (SOA). I will investigate state of the art and look at the communication processes focusing at context awareness, service discovery, service composition, service brokering and mediation.

To narrow down the task I have decided to concentrate at Web Services and the .NET platform. What is status today and what are missing to have a fully dynamic communication and application.
2 Core Service mechanisms
When creating an architecture for a system with the complexity described in Chapter 1, there will be several service mechanisms which are crucial for the quality of the system. Telenor R&D proposed some important mechanisms that have been successfully implemented in similar projects sponsored by the European Union (Zoric & Frivold, 2005). The components are shown in the figure below:

![Diagram of service mechanisms]

**Figure 2 - Main service mechanisms**

In this project I will concentrate on the five components and mechanisms with light gray background. A4C Security Manager and Quality of Service will not be designed, but instead simulated. This is due the time available for this project.

The mechanisms, Service Discovery, Composition, Brokering and Mediation do not have any agreed definition in the computer engineering society. Some people use only the term Discovery, some use Brokering and Discovery about the same thing, and not everyone use Composition and Mediation etc. This make the research harder than expected, but in the following chapters I will try to give good definitions of how I interpret them. To be able to understand what all these service mechanisms are, I will in the next chapter introduce what I put in the word “service”.

4


2.1 Service

No agreed nor uniform definition of the term Service exists today. However, IBM (Colan, 2004), OASIS (MacKenzie & Nickull, 2005), Sun Microsystems(Mahmoud, 2005), ondotnet.com (Hashimi, 2003) and other major participants in the software industry uses definitions which have a few characteristics in common:

1. A service is visible to the outside world through an interface. The consumers invokes (uses) this interface, they do not care how the service is implemented. This design pattern is called loose coupling, and allows software on each side of the conversation to change without impacting the other, provided that the message schema stays the same.

2. A service interface is platform independent. This means that consumers should be able to invoke services regardless of the operating system (such as Windows XP, OS X, Linux) the consumer and service is using.

3. A service should be independent and self-contained. It should not require information or state from one request to another when implemented.

2.2 Context Manager

The goal for the Context Manager component is to achieve context awareness. Wikipedia’s explanation of “context awareness”:

*Context awareness is a term from computer science, that is used for devices that have information about the circumstances under which they operate and can react accordingly. Context aware devices may also try to make assumptions about the user's current situation.*

The term “context awareness” is first introduced by Schilit and Theimer (Schilit & Theimer, 1994) and they refer to context as location, identities of nearby people and objects, and changes to those objects. Brown et al. got an almost similar definition: “context is the location and the identities of the people around the user, the time of day, season, temperature, etc”. As (Abowd et al., 1999) puts it “These definitions that define context by example are difficult to apply”. (Brown, 1996) defined context to be” the elements of the user’s environment that the user’s computer knows about” and (Franklin & Flachsbart, 1998) see it as “the situation of the user”. There are several
2.3 Service Discovery

more of these definitions that only use synonyms for context, and they are also
difficult to apply in practice. A very good definition is defined by (Abowd et al.,
1999):

*Context is any information that can be used to characterize the situation of an entity.
An entity is a person, place, or object that is considered relevant to the interaction
between a user and an application, including the user and applications themselves.*

It is more up to the developer to decide the context for a given application scenario.
Location, user identity, device identity, and time are example of variable context
information. I will call them the primary context information. From this primary
context information we get indices into other sources of contextual information. For
example, given a person’s identity we can acquire related information such as age,
address, email, preferences etc. With the location we can determine what other objects
or people are near the entity, and also what activity is occurring nearby. The
information determined by the primary information I will call secondary information.
The secondary information can for example be stored in a user profile database. When
using primary and secondary information to provide relevant services to the user we
get “Context-Aware Computing”.

Currently, there are two industry-supported standards for describing context
information on devices such as mobile phones, PDA’s, etc (Butler, 2002). Namely
CC/PP developed by The World Wide Web Consortium and UAProf developed by
the Open Mobile Alliance.

2.3 Service Discovery

Service Discovery is about discovering services in dynamic network environment.
Basically everything that can be accessed over the network is a service (see own
subsection below). There are many services available on the internet, and it is
becoming more and more difficult to manually search through all of the options and
also be sure of that the best match is obtained.

Service Discovery provides mechanisms which allows consumers of services
(services may be also consumers) to discover services, and descriptions of these
services. This mechanism may of course be a service itself. As the number of available services (e.g. Web Services) on the Internet keeps growing, service discovery is the key mechanism to help consumers find the services they are looking for. Service Discovery can be implemented in numerous ways; some of these are described later.

For example, you have services providing the movies at given cinemas, latest news from a newspaper, latest score in a soccer match etc. A Service Discovery takes different input parameters and returns the best matching services. This is a very important feature in a Service Oriented Architecture (see own chapter below).

### 2.4 Service Composition

The process of combining services in order to solve complex problems is called service composition (Milanovic & Malek, 2004a). Services can be combined in different order to best suit the problem requirement. Service composition enables rapid application development and service reuse. Service composition is the activity that allows developers to compose applications and processes using services from heterogeneous environments without regard to the details and differences of those environments. This is one of the primary benefits of Service-Oriented Architecture (SOA), - namely the ability to compose applications, processes, or more complex services from other less complex services.

### 2.5 Service Brokering

Brokers facilitate the interaction between two or more parties. For example, if two parties want to communicate, but they do not share a common language, Brokers may provide translation services, or if two parties do not trust each other, a Broker may provide a trusted intermediary. Merriam-Webster Incorporated (Merriam-Webster, 2005) defines brokering as: "one who acts as an intermediary". Like middlemen and brokers in physical commerce, service brokers assist requester agents in discovering and connecting with one or more relevant service providers (Sycara, Paolucci, Soudry, & Srinivasan, 2004b). Brokering is very similar to mediation, the main difference being that the broker acts more as an interpreter between the consumer and
the services, while the mediator negotiates between different services to meet the consumer’s requirements.

Brokers can also perform activities such as load balancing between different agents, or anonymizing between requesters and providers. In an analysis, (Sycara, Paolucci, Soudry, & Srinivasan, 2004a) use a Broker to perform complex tasks such as interpretation of the capability advertisements of service providers, interpretation of a requester’s query that must be fulfilled by a service provider, finding the best provider based on the requester’s query, and returning the query result to the requester. To accomplish these tasks, ontologies to describe capabilities of a Web Service are required.

### 2.6 Service Mediation

When systems communicate, their business events follow in general different data models. This requires mediation in order to transform the representation of the sender’s event into the representation of the receiver’s event.

(Merriam-Webster, 2005) defines mediation as: “a intervention between conflicting parties to promote reconciliation, settlement, or compromise”. A major challenge when composing services is that the underlying services may be heterogeneous in terms of e.g. protocols, processes and the semantics of messages. In order to bypass interoperability problems like these, mediators play a central role (Domingue, Roman, & Stollberg, 2005). Another widely used definition is simply (Wiederhold, 1994):

*Mediation is the principal means to resolve problems of semantic interoperation*

(Domingue et al., 2005) describes mediation in different categories:

**Sender mediation** refers to the situation where the sender makes sure that the receiver receives the correct format by transforming the events before they are send out. A sender therefore has to be able to deal with all receiver’s event formats and definitions.

**Receiver mediation** refers to the opposite of sender mediation. The receiver receives any format from a receiver and transforms it into the format it needs itself.
**Core Service mechanisms**

*Sender and receiver mediation* is implemented in most B2B (Business to Business) systems where the different enterprises agree on a specific standard (for example XML). In these cases, both the sender and the receiver mediate. The sender transforms any internally used format into the commonly shared format and the receiver does the opposite.

*Broker mediation.* In the brokered mediation approach neither the sender nor the receiver mediates itself. Instead, a third system, the broker, performs the mediation (see chapter 2.5). A sender sends its format of business event to the broker, the broker mediates to the format as required by the receiver, and the receiver receives business events in its format. The problem with this approach is that the broker needs to know all the formats of all senders and receivers.
3 Service Oriented Architecture

Service Oriented Architecture is an architectural paradigm for components of a system and interactions of patterns between them. A component offers a service that waits in a state of readiness. Other components may invoke the service in compliance with a service contract. - (Nickull, 2005):

Service Oriented Architecture (SOA) has been a familiar and popular way of building and planning web-based software applications. The principle of SOA is to split up functionality into independent services, which may be invoked as necessary. SOA has many principles in common with object-oriented programming: a class or an object is wrapped inside a boundary and a well-defined interface is exposed to other objects for interaction (Siddiqui, 2003). The main difference is that SOA is distributed across a network.

Currently it is a popular subject but with no standardized reference model to define it, even though it has been around for over a decade. The Organization for the Advancement of Structured Information Standards is currently working on creating a SOA Reference Model (OASIS, 2005). Among organizations and companies contributing to OASIS SOA Reference Model are Adobe, Fujitsu and Boeing.

In OASIS’ working draft of the Reference Model, this is said about SOA:

A software-architecture of services, policies, practices and frameworks in which components can be reused and repurposed rapidly in order to achieve shared and new functionality. This enables rapid economical implementation in responds to new requirements thus ensuring that services respond to perceived user needs.

SOA does not mean Web Services, .NET or J2EE, but these are specialized SOA implementations that embody the core aspects of service-oriented approach to architecture. Unlike traditional object-oriented architectures, SOA comprise loosely joined, highly interoperable application services. Because these services interoperate over different development technologies, the software components become very
reusable. It enables businesses to have a system of heterogeneous applications which can interoperate through the use of SOA (Mahmoud, 2005).

Before I go in detail of the concept SOA, I will first explain what architecture really is and then I will take a look back at the evolution, and see what the essentials behind SOA are.

3.1 Architecture in general

Software architecture can be compared with an architecture used in the building and construction industry. It explains where the walls, doors and windows are supposed to be, but it does not explain the color of the door. The architecture also describes the relation between the given elements.

Software architecture has a lot of different definitions through the ages. Some says it is high-level design or “the overall structure of the system”. This is true enough, but it is more than that. I choose to use (Bass, Clements, & Kazman, 2003)’s definition: “The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.”

In the next chapters I will go more in detail in the Service Oriented Architecture.

3.2 SOA Background

Early programmers realized that writing software was becoming more and more complex. They needed a better way to reuse some of the code that they were rewriting, and they started to introduce the concept of using modules. With modules, programmers could write subroutines and functions and reuse their code. This was great for a while. Later, developers started to see that they were cutting and pasting their modules into other applications and that this started to create a maintenance nightmare. When a bug was discovered in a function somewhere, they had to track down all of the applications that used the function and modify the code to reflect the fix. After the fix, the deployment nightmare began. Developers didn't like that. They needed a higher level of abstraction, and different strategies emerged: Component
3.3 SOA Main concepts

Based Architecture, Object Oriented programming and Distributed Systems such as DCOM, CORBA, J2EE etc. SOA is an evolution of all this.

3.3 SOA Main concepts

In absence of a formal definition and reference model Sayed Hashimi, in 2003, points out the key components of SOA: Service, message, dynamic discovery and Web Services. Web Services will be discussed in its own chapter.

3.3.1 Service in SOA

A service in SOA is an exposed piece of functionality where the interface contract to the service is platform independent and the service can be dynamically located and invoked. Again, the term “services” does not imply Web Services. - Although, Web Services are well-known implementation of services. A platform independent interface contract implies that a client from anywhere on any Operation System (OS), and in any language can consume the service. Dynamic discovery hints that a service enables a look-up mechanism where consumers can go find a service based on some criteria. For example, if I was looking for the television program service, I might query the directory service to find a list of service providers that could deliver the program list for the given channel.

![Figure 3 - Directory service](image)

As the figure above is showing the directory service is an intermediary between providers and consumers. Providers register with the directory service and consumers query the directory service to find service providers.
Service providers and consumers communicate via messages. Services expose an interface description. This description consists of technical parameters, constraints and policies that define the terms to invoke the services. This should be done in a standardized format.

While the nature of the services may vary, common standards for declaring a service exist today, such as: W3C’s Web Services Description Language (WSDL) and the ebXML’s Collaboration Protocol Profile.

### 3.3.2 Messages

Messages contain the data sent between the provider and consumer. Because the interface description is platform- and language-independent, the technology used to define messages must also be agnostic to any specific platform/language. Therefore, messages are typically constructed using XML documents that conform to XML schema. XML provides all of the functionality, granularity, and scalability required by messages. That is, for consumers and providers to effectively communicate, they need a non-restrictive type of system to clearly define messages; XML provides this.

### 3.3.3 Dynamic discovery

Directory service and dynamic discovery is an important feature of SOA. A discovery occurs when a potential consumer obtains information about the existence of a service, its applicable parameters and terms. A service must communicate its service description to potential consumers. It does so by using *Push* or *Pull* methods. In the *Pull* method, potential consumers request the service provider to send them the service description. This could be invoked as a service itself. In the *Push* method, the service provider, or its agent, sends the service description to service consumers.

There are different implementations of the directory service such as the OASIS ebXML Registry-Repository Technical Specifications and the OASIS Universal Description and Discovery Interface (UDDI) Technical Specification.
3.4 **SOA Reference model**

In Boston (MA, USA) 3 May 2005, the International standards consortium, OASIS, announced the formation of a new committee to develop a core reference model that will guide and foster the creation of specific SOA. Since SOA does not yet have a standardized reference model I will use (Nickull, 2005)’s, model. All SOA implementations share the main concepts of services, service description, advertising and discovery, the specification of an associated data model, and the use of a service contract.

![Diagram of SOA Reference model](image)

**Figure 4 - SOA Reference model**

3.5 **General Service Composition Model (GSCM)**

Following figure shows a General Service Composition Model for a SOA including the main mechanisms from chapter 2:
Note about the figure: The figure illustrates a client (consumer) communicating with a service (producer). The left side is the client and the right side is the service. The horizontal arrows show the direction of the communication. Some services are activated by another service and have therefore an arrow starting longer to the right. Example: “service brokering” can be invoked by “service composition” and is therefore the rightmost. The vertical arrow is indicating the time interval. In the figure above I have also included Quality of Service (QoS) and A4C (Authentication, Authorization, Accounting, Auditing and Charging), but these components will not be handled in detail in this project.

Because of the loose coupling for SOA, the mechanisms do not have to be in given order, but they can be mixed and rearranged from request to request. The order of the components is therefore not absolute. There also exist services that are not illustrated.
3.5 General Service Composition Model (GSCM)

in the figure. For example there might be a need for service discovery before the service composition, or even before the client request. This model is therefore a general model of “how it could be implemented”.

Most definitions of SOA identify the use of Web Services in its implementation. However, one can implement SOA using any service-based architecture. When people normally speak of a SOA, they speak of services residing on the internet or intranet using Web Services. In the following chapter I will take a “deeper dive” into what Web Services really are.
4 Web Services

Web Services is, as mentioned in the previous chapter, one specific implementation of SOA. (Mahmoud, 2005) defines Web Services as software systems designed to support machine-to-machine interaction over a network. The W3C has a more elaborate definition (Booth et al., 2004):

* A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

From a historical perspective, Web Services represent the convergence between the service oriented architecture and the Web. Web Services combine the best aspects of component based development and the Web. Like components, Web Services represent functionality that can be easily reused without knowing how the service is implemented. Unlike current component technologies which are accessed via proprietary protocols, Web Services are accessed via ubiquitous Web protocols (for example HTTP) using universally-accepted data formats (for example XML).

In the past, clients accessed these services using a tightly coupled, distributed computing protocol, such as DCOM, CORBA, .NET Remoting, or RMI. These protocols are very effective for building a specific application, but they limit the flexibility of the system since they are constrained by dependencies on platforms, languages etc. Therefore, none of these protocols operate effectively over the web. The Web Services architecture takes all the best features of the service oriented architecture and combines it with the Web. The Web supports universal communication using loosely coupled connections. Web protocols are completely vendor-, platform-, and language-independent.
4.1 Characteristics

As author John Hagel III (Hagel, 2002) notes in his book “The focus of Web Services architecture is not to connect people with Web sites. Instead, it focuses on connecting applications and data directly with each other, automating connections that might otherwise have required human intervention. It is designed to ensure that applications and data can be accessed by authorized entities, regardless of location or underlying platform.”

4.1 Characteristics

(Systinet, 2003) defines some characteristics of Web Services:

- A Web Service is accessible over the Web.
- A Web Service provides an interface that can be called from another program.
- A Web Service is registered and can be located through a Web Service Registry.
- Web Services support loosely coupled connections between systems

4.2 Technologies

Web Services can be developed using any programming language and can be deployed on any platform. Web Services can communicate because they all speak the same language: XML. The applications need standard formats and protocols that allow them to properly interpret the XML. Three XML-based technologies have emerged as the standards for Web Services:

- **SOAP** defines a standard communications protocol for Web Services.
- Web Services Description Language (**WSDL**) defines a standard mechanism to describe a Web Service.
- Universal Description, Discovery and Integration (**UDDI**) provide a standard mechanism to register and discover Web Services.

The core Web Service Technology Stack according to W3C is as follows:

18
4.2.1 SOAP

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP can potentially be used in combination with a variety of protocols like HTTP, SMTP and other protocols. The SOAP forms the foundation layer of Web Service, and provides a simple and consistent mechanism that allows one application to send an XML message to another application.

SOAP originally was an acronym for *Simple Object Access Protocol*, but the acronym was dropped in version 1.2 of the SOAP specification. The SOAP 1.2 Working Group has kept the SOAP name since it is so popular, but decided against spelling it out to avoid misleading developers, because the O (for Object) is not in focus anymore. Today's official definition, found in the most recent SOAP 1.2 specification (Gudgin, Hadley, Mendelsohn, Moreau, & Nielsen, 2003), does not even mention objects:

*SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.*

SOAP consists of three parts (Jepsen, 2001):

- The SOAP envelope construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory. Within this envelope are two additional sections: the header and the body of the message.
4.2 Technologies

- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

An example of a SOAP request packet used in a Web Service:

```xml
<soap:Envelope xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <AddNumbers xmlns="http://tempuri.org/">
      <lnNumber1>6</lnNumber1>
      <lnNumber2>4</lnNumber2>
    </AddNumbers>
  </soap:Body>
</soap:Envelope>
```

The response of this SOAP HTTP request may be:

```xml
<soap:Envelope xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <AddNumbersResponse xmlns="http://tempuri.org/">
      <AddNumbersResult>10</AddNumbersResult>
    </AddNumbersResponse>
  </soap:Body>
</soap:Envelope>
```

As mentioned, SOAP defines a standard communications protocol for Web Services. The next section will go in detail in the language used to describe these Web Services.

4.2.2 Web Service Description Language (WSDL)

Web Services Description Language Version 2.0 (WSDL 2.0) provides a model and an XML format for describing Web Services. It indicates how potential clients are intended to interact with the described service (Chinnici, Moreau, Ryman, & Weerawarana, 2005).

To be more specific, a WSDL description specifies:

- What functionality the Web Service offers
Web Services

- How the client should communicate with the Web Service (format of messages supported protocols)
- Where the Web Service is accessible (access point to an instance of the Web Service)

W3C specifies seventeen different elements that WSDL use in order to describe a Web Service (Chinnici et al., 2005). Some of these are:

- A **Service** component describes a set of endpoints at which a particular deployed implementation of the service is provided.
- An **Endpoint** component defines the particulars of a specific endpoint at which a given service is available.
- An **Interface** component describes sequences of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a sequence of input and output messages, and an interface is a set of operations.
- A **Binding** component describes a concrete message format and transmission protocol which may be used to define an endpoint.

### 4.2.3 Universal Description, Discovery and Integration (UDDI)

UDDI is an acronym for Universal Description, Discovery and Integration, and it is a platform independent XML-based registry used by businesses all over the world to list their services on the Internet. UDDI provides a mechanism to register and categorize Web Services that you offer and to locate Web Services that you would like to consume.

When looking for a Web Service, a developer queries the UDDI registry, searching for a business that offers the type of service that he wants. See Figure 8. Similarly, client code can use the UDDI registry as a naming service. It asks UDDI to return the endpoints for a service of a particular name or type, and bind to one of the returned URLs dynamically.
4.2 Technologies

UDDI Services includes a Web interface with searching, publishing, and coordination features. It is enabling enterprise developers to publish, discover, share, and interact with Web Services directly through their development tools and business applications. OIO InfrastructureBase has good explanation of the entities in UDDI services at their website:

![Diagram of UDDI Services](image)

**Figure 7 - UDDI Services**

The following definitions describe each entity and its role in relation to other entities.

- **Provider** A business or group that offers one or more XML Web Services. For example, a business, business unit, organization, organizational department, person, computer, or an application can be a provider in UDDI Services. A provider represents the "parent" entity under which all contact, service, and interface information is stored and organized. In my proof of the concept in Chapter 9 Telenor ASA act as the service provider.

- **Contact** A human or computer resource that can be contacted for information about a provider or the XML Web Services it offers. A provider may have as many contacts as is necessary to identify each of its available contact points.

- **Service** An entity that describes and provides access to a function you would like to share with other users. Services can perform any function across a network, from simple requests to complicated business processes. The services in the customer Use Case example, Figure 22, are good examples. These services may have one or more bindings.
**Binding** The point where a specific implementation of a service can be accessed, such as the URL where an interface can be found. Bindings may also include one or more *instance info* structures.

**Instance Info** A reference to a tModel that contains relevant technical information about a binding, such as an interface specification document or Web Services Description Language (WSDL) file.

**tModel** tModels are typically used to provide technical information about an interface, such as a WSDL file, that describes the conventions that are supported by an interface.

### 4.3 Web Service Interaction

![Diagram of Web Service Interaction](image)

**Figure 8 – The interaction of a Web Service**

The figure above shows how the core technologies of a Web Service relate to one another. A simple scenario which corresponds to the illustration above may be:

1. A Web Service called `getNewEvent()` for returning events of a soccer match is registered in an UDDI registry and a link to the WSDL of the Web Service is given.
2. A Service Consumer browsers / searches the UDDI registry and finds a suitable Web Service, for example the `getNewEvent()` service.
3. The Service Consumer retrieves the WSDL description of the Web Services in order to find out how to use it.
4. The Service Consumer prepares a request formulated in XML and further encapsulated in a SOAP message.
5. The SOAP message is transported using HTTP to the Web Service SOAP Server.
6. The Web Service SOAP Server receives the SOAP request and invokes the local application which is associated with the Web Service.
7. The result of the local program is formulated in XML, encapsulated in SOAP and returned to the Service Consumer using HTTP.

4.4 The General Service Composition Model and Web Services

After describing the General Service Composition Model (GSCM) for SOA in section 3.5, we can start investigating how Web Services can be used to implement all the core mechanisms. Below, each step in GSCM is mapped onto the core functionality Web Services offer:
As the figure shows, there are some major components of the GSCM (marked with “Nss-x” in red) that are not very well supported using the core functionality of Web Services. Service Composition, Service Brokering, Service Mediation are components that require more intelligent and extensive solutions than the Web Services itself can offer. These components and solutions on how to solve these issues will be handled more in detail in the following chapters.

### 4.4.1 Nss-1

Context Manager, as explained in chapter 2.1, is a very important component to provide an efficient approach for personalizing network services and to achieve a context-aware service. A Context Manager is not a part of the core Web Service
4.4 The General Service Composition Model and Web Services

functionality; therefore a custom class must be created. When designing a context manager two main areas must be covered: How should the context information be updated, and when should it be updated.

**How:** There exist several solutions to get, update and save the context information. In the following examples I will use context as Time, Location and Address. One solution can be:

![Sequence Diagram](image)

**Figure 10 - Nss- 1 version 1**

In the figure above the client first request a getTime() Web Service, second getLocation() and third getAddress(). In the end the client update the context database through the UpdateContext() Web Service. This is not performance efficient, because of all the overhead in every request. If it is a central Web Service we will also have a bottle-neck when several clients request at the same time. The client can be implemented in all kind of devices, and it is therefore important to minimize the client processing. It is important to find a balance of the processing work (Flinn, Park, & Satyanarayanan, 2002). – A balance between a thick and a thin client.

A perhaps better sequence schema is this:
The biggest difference is the creation of several services, - services which have their own responsibilities. This is the heart of SOA. The client is simplified and sends only one request, namely “UpdateContext”. To make this work the client has to send with some parameters. For example, this could be a user identification number and device identification number. Instead or in addition of the “updateDb” command, the server can create a container of the context which it stores in the cache. Then it will be an even better performance since the processes do not have to query the database all the time.

**When:** This is another issue. When is the best time to do a context update? I focus on a pull technology in this report, and therefore the simplest way of doing it, is by giving updating the context before retrieval. When processing several commands in sequence there is no need for updating every time, but only prior to the first one. To gain performance the data can be cached at server for a 5-10minutes or so.

### 4.4.2 Nss-2

“Service composition” is explained in chapter 2.4. A service composer can act the same way as the “Web Service” component does in Figure 11. Another model of an equal scenario can be:
4.4 The General Service Composition Model and Web Services

The figure above shows several Web Services: Time -, Location -, Address -, Database provider and even the Context composer can be implemented as a Web Service. The idea here is the same as in the subsection above. A client, or another service, should be able to ask the “context composer” to update the context, and then this service will trigger the time-, location- and address provider services. Finally it will update the database through the database provider. Another possibility instead of only updating the database is to create a context container which is created and saved in cache.

There is no system support for this feature in core Web Services. Therefore a custom class has to be made.

4.4.3 Nss-3

To achieve dynamic service architecture a service broker is needed. I explained these components in chapter 2.5. The figure below shows a service broker:

A scenario can be that another service, for example the Service Composer, sends the request “getPicture” to the Service Broker, and parameters telling what kind of picture
format that is required. The broker then has to do a service discovery, and do some reasoning to pick which service to use. The core Web Service is not supporting this component, and a custom class is needed.

### 4.4.4 Nss-4

Service mediation is also needed to have a fully dynamic service (Chapter 2.5). The figure below shows a service mediator:

![Service Mediator Example](image)

**Figure 14 - Service mediator example**

In this figure we have five services, three of them are simple converting services, converting from one picture format to another (from TIF to JPG, from JPG to GIF and GIF to BMP), and two of them are the Service Broker and the Service Mediator WebService. A scenario can be that a client wants a picture in BMP-format, because this is the only format supported by his/her device. It is therefore send a “convertPicture” request, including parameters and preferences, to the Service Mediator. It is the Mediator’s responsibility to make the reasoning and to find out that the “mission” can be accomplished when using all three services together.

These kinds of operations are neither supported by the core Web Service architecture, and a custom class is needed.

### 4.4.5 Nss-5 and Nss-6

The QoS and A4C components will not be explained in detail. In this project a custom class will be created, to simulate the work and processes.
5 .NET
During the summer of 2000, at the Professional Developers Conference (PDC), the Microsoft Corporation announced its vision of the next generation of Windows services, .NET. Since that time, many steps have been taken to ensure that .NET becomes a reality. .NET is the Microsoft Web Services strategy to connect information, people, systems, and devices through software. Integrated across the Microsoft platform, .NET technology provides the ability to quickly build, deploy and manage solutions with Web Services. .NET-connected solutions enable businesses to integrate their systems more rapidly and in a more agile manner.

To better explain what .NET really is, I will first introduce the .NET Framework

5.1 Framework

Languages: There are over 30 languages that support the .NET Framework. In addition to the languages in the figure you also have support of the following script and languages: J#, Eiffel, COBOL, Haskell, Mondrian, Mercury, Forth, Perl, Pascal etc.
**CLS:** Common language Specification. Defines the reasonable subset of the CTS (see own subsection) that should be sufficient to support cross-language integration, and excludes areas such as unsigned integers, operator overloading, and more.

**ADO.NET:** Supports ASP.NET and Windows Forms so they easily can connect data stores.

**Base Class Library:** Called Framework Class Library (FCL). The purpose of this layer is to provide the services and object models for data access and manipulation, data streams, security, thread management etc. In many respects the Windows API has been abstracted into these base classes. These libraries are an object-oriented collection of classes that can be reused and enhanced to enable rapid application development. The classes support the creation of applications that range from ASP.NET Web pages and Web Services to traditional Windows and command line applications. Below you can see an overview of the class library.

![Figure 16 - .NET Framework Class Library](image)

**CLR:** Will be handled in the next sub section.

### 5.1.1 CLR and CTS

The Common Language Runtime (CLR) serves as the execution environment for the .NET Framework. The CLR is responsible for managing the compiled code of .NET applications, which can be written in several different languages including VB, C#,
5.1 Framework

Java, Perl etc. The cross-language integration is achieved through the major components of CLR: Intermediate Language and Metadata.

**Intermediate Language** (IL) is an assembly language that runs on almost any type of CPU. As managed code, IL is just-in-time (JIT) compiled when .NET applications are executed. JIT compilers convert IL into machine language that is specific to the host CPU. In .NET this IL language is called MSIL (Microsoft Intermediate Language).

**Metadata** is a description of the implemented code. It is responsible for ensuring that the CLR executes the code securely. To prevent modules of software from breaking type definitions, Metadata stores information regarding classes, methods and types.

Figure below shows the steps involved in executing a .NET Framework application:

![Diagram showing the steps involved in executing a .NET Framework application](image-url)

**Figure 17 - Common Language Runtime**

Ultimately, the CLR exists to safely execute managed code, regardless of source language. In order to provide for cross-language integration, to ensure type safety, and to provide managed execution services such as JIT compilation, garbage collection etc, the CLR needs knowledge of the managed code that it is executing.

**CTS**: The meet this requirement, the CLR defines a system called the Common Type System (CTS). The CTS defines the rules by which all types are declared, defined and
managed, regardless of source language. CTS is designed to be rich and flexible and is the basis for the cross-language integration.

### 5.2 .NET Web Services

The .NET framework is almost tailored to the use of Web Services. Using the newest version of Microsoft Visual Studio .NET (Microsoft, 2005c) many things are “out of the box”. When you create a Web Service you automatically get support for a WSDL schema that describes the Web Service by accessing the .asmx page with a query-string of “wsdl”: “webservice.asmx?wsdl”. This .asmx page either contains a class that provides the Web Service functionality or references a specific external class that handles the logic in an external class file. Classes are standard .NET classes and the only difference is that every method that you want to expose to the Web is prefixed with a [WebMethod] attribute. Once the .asmx page has been created the Web Service is ready for accessing over the Web.

.NET provides a very useful client interface about your Web Service, showing all the methods and parameters along with information on how to access the Web Service over the Web. You can also use this page to test basic operation of your Web Service without calling the Web Service with a ‘real’ client. The URL to the WSDL file is submitted to UDDI which handle Service Discovery in .NET.

### 5.3 GSCM implemented using .NET Web Services

As we saw in Figure 21, there is a gap between components, and Web Service functionality. There are Semantic Web technologies, such as MetaData, OWL-S, and tools like Jena which can be added to gain a more complete mapping. Due to the focus of .NET, those components will not be handled in this project. Instead I will use known Microsoft products and application Microsoft take part in. Se figure below: (The red text in the leftmost lane indicates a new component and/or class).

---

33
5.3 GSCM implemented using .NET Web Services

In the figure custom classes (Cc in short) must be created several places. “New” technologies such as BPEL and application like BizTalk are inserted.

**Business Process Execution Language for Web Services** (BPEL4WS/BPEL) (IBM, 2005) can be implemented as a Service Composer to control the business process flow. BPEL makes it possible to semantically describe the Web Services, and it is therefore possible for a Web Service to go dynamically through the business flow of various connected Web Services. The negative part with BPEL, is that the workflow must be designed by the programmer by hand and it is therefore no software that create this workflow programmatically.

**BizTalk** is more a business to business application, and can not be used as a general Service Composer, but it can be used in some cases. Adapters exist to make BizTalk

---

![Diagram of General Service Composition Model](image-url)

**Figure 18 - Enhanced Web Services using .NET**

<table>
<thead>
<tr>
<th>GSCM and Core Web Services</th>
<th>GSCM and Enhanced Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP request over HTTP</td>
<td>SOAP request over HTTP</td>
</tr>
<tr>
<td>Nss-5</td>
<td>Custom class (Cc)Http</td>
</tr>
<tr>
<td>Nss-1</td>
<td>Cc wBPEL</td>
</tr>
<tr>
<td>Nss-2</td>
<td>Custom SB-class</td>
</tr>
<tr>
<td>Nss-3</td>
<td>UDDI / CoBPEL/BizTalk</td>
</tr>
<tr>
<td>Discovery by UDDI</td>
<td></td>
</tr>
<tr>
<td>Nss-4</td>
<td></td>
</tr>
<tr>
<td>Start Instantiate Service</td>
<td></td>
</tr>
<tr>
<td>Nss-6</td>
<td>Co</td>
</tr>
<tr>
<td>Nss-5</td>
<td>Co</td>
</tr>
<tr>
<td>SOAP response over HTTP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Nss – Not System Support. There is not system support for this feature
Cc – Custom class. Creation of a custom class is needed.
handle different broker and mediation issues but only in specific cases. It is also possible to integrate UDDI in BizTalk (Microsoft, 2005a), but the required level of dynamic I seek is hard to achieve. There will always be some static implementation when I am using these tools.

As it is today, .NET is not well-equipped for creating fully dynamic services. As Figure 21 indicates, other mechanisms and tools have to be implemented.

Some might say that J2EE is better than .NET Web Services, and that J2EE might solve this “dynamic problem”. In the next chapter I will examine the differences and similarities with these two technologies.
6 J2EE and .NET Web Services

Both Sun’s J2EE and Microsoft’s .NET support WSDL, SOAP, UDDI, XML and the technologies and tools needed to get Web Services running. To find out which is best, J2EE or .NET, several benchmarks have been implemented and tested. Sun had a benchmark (Sun Microsystems, 2004) which shows Java outperforming .NET in their Web Service Test by a wide margin. As a response to this report Microsoft did a similar benchmark (Microsoft, 2004) and got the same results for the performance of the Java Web Services as reported by Sun. They also got a .NET result which where more than three times better than Sun reported. .NET roughly matched or slightly exceeded J2EE. Some issues with Sun’s benchmark is that they did not use a major commercial application server in their testing, choosing instead a lightweight Tomcat application that does not support the core J2EE features. A benchmark who has taken this in consideration is Doculab’s “Performance Benchmark Study” (Doculabs, 2003). The benchmark reports the relative performance of .NET and J2EE Web Services when used against a variety of databases and servers. Some of the results from this benchmark are:

- Microsoft .NET provided the best performance among the products evaluated in all tests
- Against a backend Oracle 9i database, .NET on Windows Server 2003 achieved 110 percent better throughput than the leading J2EE product against the same Oracle 9i database.
- The results from the interoperability tests prove that with Web Services, interoperability between .NET and J2EE platforms is a reality, and performance is not a blocking issue
- Overall, .NET provided significantly better performance results versus the J2EE products acting as both a Web Service client and a Web Service host.

From all the benchmarks implemented in this subject, I believe the result is tight coupled with whom and for who the test are taken. An advantage with .NET is the fact that Web Services were part of the core .NET framework from the beginning; .NET is designed for Web Service purposes and not added features on top of an existing framework as is the case with J2EE. Another advantage with .NET could be
that a lot is coming “out of the box”. WS-I, Web Service Interoperability Organization, has organized standards like WS-Security, WS-MetaDataExchange, WS-Messaging etc for Web Services. Many of these standards are supported by .NET in an add-on called WSE 3.0, Web Services Enhancements. An advantage for J2EE is that it is supported by a wider choice of vendors, .NET only support Microsoft but in return they have great support for other programming language, and not only Java.

When choosing between J2EE and .NET it its more likely to be influenced by the present system, and by the programming language already familiar. There are no big issues telling you which to choose, and the summary of Doculab’s benchmark test also told us that interoperability between them is a reality and at least performance is not blocking.
7 Smart Client

I would like to include a chapter about “Smart Clients”. This is more like a term used in the “Microsoft World”. Not everyone is familiar with it, so I will therefore give a simple explanation. Smart Clients will be an essential module in the architect of the “proof of concept”.

A Smart Client is a portable device such as PDA, some mobile phones, Tablet PC, or laptop PC as long as the device contains enough local processing power to provide a good user interface and experience similar to what a user would fin on their desktop PC. The term “smart client” comes from the age of “dump terminals” on client/server systems where PCs being introduced on the network as the “smart clients” because they could do processing locally rather than just displaying what was transmitted.

Beyond local processing power, there are several factors that categorize whether or not a device can be classified as a smart client. (Minatel, 2005) wrote the following in an article from 2004: “First, a smart client needs to be connected, and able to connect with other devices on the network in some way (...) Smart clients also need to continue to work when the network is not available (...) a smart client should be easily tailored to the specific device and the smart client applications should be able to be updated easily.”

Microsoft (Microsoft, 2005b) defines smart clients as computers, devices, or applications that can provide:

- The best aspects of traditional desktop applications, including highly responsive software, sophisticated features for users, and great opportunities for developers to enhance existing applications or create new ones.
- The best aspects of “thin clients,” including a small form factor, economical use of computing resources such as processors and memory, ease of deployment, and easy manageability.
- A natural, easily understood user interface (UI) that is high quality and designed for occasional connectivity with other systems.
- Interoperability with many different types of devices.
- The ability to consume Web Services.
8 Static and dynamic services

In the past chapters I have spoken a lot of dynamic and not dynamic services. In the next subsections I will explain more in detail what I mean with a static or dynamic service.

8.1 Today’s static services

As I mentioned in the introduction: “Today’s systems use very much predefined, static service and architecture solutions, which do not satisfy the service dynamics, required to exploit the next generation services”. An example can be a MMS service sending messages with info every time a defined soccer team scores. This message will be composed and send to the subscribers with same amount of chars, amount of info and even same resolution at the picture. To send the message to a subscriber without the possibility for MMS it could be send as a SMS instead, -off course the Multimedia part in the message cannot be send. Similar if the subscriber only has pc with internet, the message could be send as an e-mail. See figure below. This is the way it is often done these days, and this is what I call “static”.

![Static message service diagram](image)

Figure 19 - Static message service

To improve this static service, it is possible to create mobile software which communicates with the particular soccer service. This is due to the Java and .NET support which is implemented in newer mobile phones. The mobile phone can now be categorized as a smart client (Chapter 7). The model below introduces an improved solution and is based on Service Oriented Architecture (Chapter 3).
8.1 Today’s static services

Scenario of the figure above:
1. The soccer service is added in a directory service
2. The consumer searches the directory service and finds the soccer service.
3. The consumer uses the service
4. The service checks the database for new events, and returns a message to the consumer.

This SOA approach in Figure 20 has a greater level of dynamic compared to Figure 19 but there is still long way to become an ultimate dynamic service. The “Broker / Discovery” service in the figure above is normally implemented with UDDI, and UDDI is only searchable by keywords. In other words, it is only possible for humans and not computers to search to registry. It is therefore up to the programmer to implement the Web Services at the client. –which become a very static way of doing it

At “scenario number 3” the client consumes the service of the provider. From the service schema (WSDL) it is possible to find out that the service takes a String as input, but it is impossible for a computer to know what this String is. If it is a name, id, password, etc. A human can read and interpret what he thinks it is because the xml-tags typically look like this <Username>string</Username>. It is not hard to
understand that the string input must be your username, but for a computer “username” is just a new string. For example no logic is related to the property “Username” in a Customer object. The input parameter does not have to be only a string, but it can also be an object, and a service can have the following xml-tags:

\[
<\text{Customer} \text{object}</\text{Customer}>
\]

When working with large enterprise applications UDDI is not commonly used. I had ha discussion with Objectware (Nissen & Frivold, 2005) and he confirmed this lack of use. The normal approach in these cases is by using for example Java Naming and Directory Interface (Sun Microsystems, 2005) when using Java and BizTalk (Microsoft, 2005a) in .NET. Because of the complexity and time-limit for this project, these technologies will not be handled in-depth.

The static service which is popular today is no described. The next section tells more about what is all the software engineers today working with. What will be a more common approach in the future?

### 8.2 Tomorrow’s dynamic services

Mechanisms to create dynamic services have already been introduced: Context Awareness, Service Discovery, Service Composition, Service Brokering and Service mediation.

*Context Awareness* is gained, as explained earlier, by both primary and secondary information. The primary context is the most variable one. This information must be provided to the service by the client. The location, IP, country, domain and even battery power, can change, and the service must be aware of this. Maybe the battery power is so low that a video, which is very power consuming in a mobile phone, is not appropriate to send. Then the service must custom-fit the message and for example just give a link to the video where it can be downloaded whenever the client want. It is possible that the client has a desire to receive the video no matter what. This is a personal parameter that can be submitted at a web site, and then become primary context.
8.2 Tomorrow’s dynamic services

*Service Discovery and Composition* must be more dynamic and intelligent than for example UDDI is now. More logic and semantic explanation of the services are preferable. Different solutions of this are proposed. (Paolucci, Julien Soudry, Srinivasan, & Sycara, 2004) describe the implementation of a Web Services based broker using OWL-S. (Milanovic & Malek, 2004b) describe another BPEL and compare this approach to OWL-S.

The future Web Services are moving towards the Semantic Web area of Computer Technology:

![Figure 21 - Evolution of WebServices](image-url)
9 Proof of Concept Architecture

Because of the short time for this project and the lack of possibilities in the .NET framework for creating a service with the required level of dynamic, I will describe and modelate a “proof of concept”. This will illustrate how the architecture and requirements for the concept, and hopefully clear up some misunderstanding.

9.1 Scenario

To have a common idea of what the architecture and the models is describing I will try to give a story which I will use as a “test scenario”:

A person, Joe, is subscribing a service from Telenor ASA which will give him a message with the last event in a soccer-match between the two teams Rosenborg and Vålerenga every time he requests. Joe is currently at work but he will go home soon. Joe owns a modern car with an integrated PDA in the dashboard, and he got an old mobile phone because the new one he got for his birthday is not working.

The goal here is to deliver an event message to Joe no matter where he is (work, car or at home), and the message should be custom composed for every device he use. For example, when he is at work Joe wants a rich message send to his computer through the office broadband network. A message with a video clip, picture in full resolution and all the text offered. When he is in the car on his way home, he do not want the video-clip because of the small bandwidth and because of driving safety, but he would like a link so he could download the video-clip if he feels for it. Further he want the picture resolution to be “as good as it gets” at his PDA, and he want all the text info provided. When he is at home, he wants the message to be sent to his mobile phone. He only wants to receive a message when someone scores, and he wants to know the current score and who made the goal.

9.2 Requirements

To be able to solve and do reasoning about the scenario in previous section and to be able to compose a “perfect fit” message the system got several requirements. Some of them are:
9.3 Use Cases

1. The system must know **who** the person is.
2. The system must know **where** the person is.
3. The system must know **what** device the person is using.
4. The system must know **which** special preferences the person got.
5. The system must know the presence of the person (in meeting, outdoor etc).
6. The system must also know the connection bandwidth, address etc.

9.3 Use Cases

9.3.1 Use Case from the Customer's point of view

In the following figure the Use Case in the Customer’s point of view is displayed. The Customer access through different gateways/proxies and is given five invokable methods.

![Diagram showing use case in customer's point of view](image)

Figure 22 - Use Case in Customer's point of view
In the next five tables display the Use Cases with text information.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>C1 Register/unregister as a user of the system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Customer</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A webpage/script must exist</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The registration must be stored in the system (database middleware)</td>
</tr>
</tbody>
</table>
| Normal events | 1. Customer enters the registration webpage which invokes the system using Web Services  
2. Customer registers all info  
3. Customer unregisters  
4. Customer saves the information |
| Variation | 1. It is possible to use another medium than a webpage to register. For example sending a sms etc.  
2. Not all info is needed. Some is optional. |

<table>
<thead>
<tr>
<th>Use Case</th>
<th>C2 Subscribe/unsubscribe to a service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Customer</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The Customer is registered</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The subscription must be saved in the system (database middleware)</td>
</tr>
</tbody>
</table>
| Normal events | 1. Customer enters the subscription webpage which invokes to the system using Web Services  
2. Customer logs in  
3.1. Customer searches/browses and finds a desired service  
3.2. Customer makes a subscription  
3.3. Customer unsubscribes services  
4. Customer submits data to the webpage |
| Variation | If user is not registered. Redirect to register page |

<table>
<thead>
<tr>
<th>Use Case</th>
<th>C3 getNewEvent service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Customer</td>
</tr>
</tbody>
</table>
### 9.3 Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>C4 UpdateContext service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Customer</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The Customer is registered and logged in</td>
</tr>
<tr>
<td><strong>Postconditions</strong></td>
<td>The system (database middleware) is updated with the correct context information</td>
</tr>
<tr>
<td><strong>Normal events</strong></td>
<td>Customer invokes the UpdateContext service through a webpage</td>
</tr>
<tr>
<td><strong>Variation</strong></td>
<td>Can also be triggered by SMS, Smart client etc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case</th>
<th>C5 GetServiceList service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Customer</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The customer is registered and logged in</td>
</tr>
<tr>
<td><strong>Postconditions</strong></td>
<td>A correct list is returned</td>
</tr>
<tr>
<td><strong>Normal events</strong></td>
<td>1. Triggers the GetServiceList event</td>
</tr>
<tr>
<td><strong>Variation</strong></td>
<td>1a. Can also be triggered by SMS, Smart client etc</td>
</tr>
</tbody>
</table>
## 9.3.2 Use Case from the System's point of view

In the next five tables display the Use Cases with text information.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>S1 Register/unregister as a user of the system</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>Register/unregister Web Service</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>A Customer must have invoked the Register/unregister Web Service with the appropriate parameters</td>
</tr>
<tr>
<td><strong>Postconditions</strong></td>
<td>The registration is saved in the database middleware.</td>
</tr>
</tbody>
</table>
| **Normal events** | 1. The Register/unregister Web Service checks whether the submitted information is valid  
2. The Customer information is stored in a database  
3. The Register/unregister Web Service returns a confirmation message. |
| **Variation** | The Register/unregister Web Service returns an error message. |
### 9.3 Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>S2 Subscribe/unsubscribe to a service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Subscribe/unsubscribe Web Service</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A registered Customer must have invoked the Subscribe/unsubscribe Web Service with the appropriate parameters such as userID and serviceID</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The subscription must be saved in database middleware</td>
</tr>
</tbody>
</table>
| Normal events | 1. The Subscribe/unsubscribe Web Service checks whether the specified user and service exists  
2. A connection between the user and service is created in the database middleware.  
3. The Subscribe/unsubscribe Web Service returns a confirmation message. |
| Variation | The Subscribe/unsubscribe Web Service returns an error message. |

<table>
<thead>
<tr>
<th>Use Case</th>
<th>S3 getNewEvent service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>getNewEvent Web Service</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A registered Customer has invoked the getNewEvent Web Service with the appropriate parameters such as userID and serviceID</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The getNewEvent Web Service returns the newest content the service can offer to the customer</td>
</tr>
</tbody>
</table>
| Normal events | 1. The getNewEvent Web Service invokes the composer with parameters such as the userID and serviceID  
2. The Composer looks up the Service in the database and looks for metadata  
3. The Composer asks the Multimedia Repository for the newest items matching the Service metadata  
4. The Multimedia Repository returns pointers to the newest multimedia items to the composer along with corresponding metadata  
5. The composer requests the context manager from the database  
6. The Context is retrieved form the database, and the Composer looks up the requirements for the Context |
7. Based on the requirements for this Context, the Composer may decide that the appropriate multimedia items needs adjustment
8. The Composer asks Service Discovery if it can offer any services that can modify the items so that they meet requirements
9. Service Discovery looks through its register, and returns pointers to the correct services to the Composer
10. The Composer invokes the resize/trim/crop Web Services that are required to make adjustments to the multimedia item
11. The new multimedia item is stored in a new location accessible by the user
12. A pointer to the multimedia item is returned to the user.

**Variation**

8b. The Composer asks Broker and Mediator for special services and they perform some reasoning to fit the Composer’s needs. The Broker and Mediator use the Service Discovery.
12b. The multimedia item is returned to the user and interpreted.

<table>
<thead>
<tr>
<th>Use Case</th>
<th><strong>S4 UpdateContext</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>UpdateContext Web Service</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>A registered Customer has invoked the UpdateContext Web Service with the appropriate parameters</td>
</tr>
<tr>
<td><strong>Postconditions</strong></td>
<td>The database is updated with the correct context information</td>
</tr>
</tbody>
</table>
| **Normal events** | 1. The UpdateContext Web Service checks whether the specified user and service exists  
2. The users context information is changed in the database  
3. The UpdateContext Web Service returns a confirmation message. |
| **Variation** | Can also be triggered by SMS, Smart client etc |
9.3 Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>S5 GetServiceList service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>GetServiceList Web Service</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A registered Customer has invoked the GetServiceList Web Service.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>A correct list of available Web Services is returned</td>
</tr>
<tr>
<td>Normal events</td>
<td>1. The GetServiceList Web Service invokes the Service Discovery.</td>
</tr>
<tr>
<td></td>
<td>2. Only the appropriate services are collected</td>
</tr>
<tr>
<td></td>
<td>3. A XML list of the WSDL's are returned</td>
</tr>
<tr>
<td>Variation</td>
<td>Can also be triggered by SMS, Smart client etc.</td>
</tr>
</tbody>
</table>

9.3.3 Database view

In this proof of concept implementation I will have three main data tables, and several related tables. The main tables are Customer, Device and Match. A Customer is the client or person using the service. A Customer have a presence like “outdoor”, “at home”, “at work”, several preferences like “only text”, “never video”, “cheap as possible” etc. A Customer can be subscribed for several services and have several devices. A device is for example a mobile phone, PDA or a laptop, and further the devices got different limitations as max resolution of pictures, different support of multimedia types (picture, video, audio), different support of communication technologies like GPRS, UMTS, and GSM etc. In this example the service is giving information of a specific event. An event is a match between two soccer teams. Every time an event in the match occurs, it is added to the EventList. The items in EventList can consist of a video, audio, picture and some text.
The database model above is not complete nor an optimal model, but it is a minimum of the data needed in this proof of concept.

**9.3.4 Sequence Diagram**

To illustrate the work between the components in more detail I have created a sequence diagram which illustrate the work done when a customer, called client in the diagram, request the `getNewEvent()` method at the server. The client could be using any of the devices mentioned in the previous subsection.
As the figure above illustrates the Service Composer, Service Broker and Service Mediator, all call the `findService()` method of the Service Discovery. This is not necessary every time in every scenario, but it illustrates that there are a loose coupling between the services, and in principle all the services can be mixed in order. My illustration is just an example.

In this specific example the Service Broker did not have a picture in the format according to the multimedia type supported by the customer’s device. Therefore it had to look up in the Service Discovery for a picture-converting-service. The `ConvertPicture()` method of the Service Mediator is then requested with the input `fromFormat` and `toFormat`. The Service Mediator does not find a service with perfect match but from the Service Discovery it finds out that it can use two distinct converting services in “pipe and filter” to create the wanted format. For example one service converts from TIF to JPG and one from JPG to GIF. If the `toFormat` is GIF
and `fromFormat` is TIF, then the two services can be used in series to get a GIF picture.

These last subsections illustrate the complexity of the components and services. The next chapters will give a conclusion of the work and point out what should be done in future work.
9.3 Use Cases

10 Conclusion

In this report, I have examined Service Oriented Architecture, Web Services, .NET and several service mechanisms. Mechanisms required to meet the next generation of dynamic services are context manager, service composer, service broker, service mediator and service discovery. It exists different understandings of these services in the software communities, and I try to explain the way I interpret them in this report.

The SOA is really a very simple architecture, but despite it is very hard to find a common explanation of what it really is. SOA does not yet have a standardized reference model, but OASIS has formed a committee to develop a core reference model that will guide and foster the creation of specific SOAs. Several definitions have been evaluated in this report and they all agree about that SOA are distributed across networks and the principle is to split up functionality into heterogeneous and independent services that may be invoked as necessary.

Further in the report the Web Service architecture is explained, and the choice to implement either Java or .NET Web Services are discussed. It is not possibilities or limitations of the specific language that decide which to choose, but rather the present system in the enterprise and the language already familiar.

When mapping the general service mechanisms required in dynamic services onto the core Web Service technology, several lacks of support is identified. The core Web Service has limited support for service discovery and no support for automatic and dynamic context management, composition, brokering or mediation. An enhanced Web Service which implements WS-I’s like WS-Security, WS-Transactions, WS-ReliableMessaging etc, can cope with some of the flaws, but there is still need for the creation of many custom classes. Even if technologies and software like BPEL, BizTalk etc are integrated. OWL-S, semantic onthologies and other semantic web technologies are not in the scope of this report, but up to the present date, it looks like they are the only way to handle the implementation of dynamic services.
My modeling of a proof of concept gives an architecture of the processes between the given mechanism, and propose a way to interact. This is only one of many ways to do it, and its purpose is to give an example of how the data flow can be implemented.
11 Further work

This report makes a foundation for further work on the subject. Technologies, mechanisms and state of the art are covered and a proof of the concept architecture is purposed. To better explain and identify the problem areas a code example should be created to give a physical example.

Because of the all the lack of support for the mechanisms, there exists a lot of work to be done at all areas. How the mechanism physically should be implemented, what are their input and output and so on.

Plug-ins for the UDDI or the BizTalk server or improvements of BPEL are interesting areas for further work. During the creation of this report Microsoft Visual Studio 2005, Microsoft SQL Server and Web Service Enhancement 3.0 are released, and in January 2006 - Microsoft BizTalk 2006 is released also. It is possible that these softwares and technologies might have improved support for dynamic services and their composition. I have especially expectations for BizTalk, and the possibility to create a plug-in for it.
## 12 Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>Third-Generation (mobile telephone technology)</td>
</tr>
<tr>
<td>4G</td>
<td>Fourth-Generation (mobile telephone technology)</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>CLR</td>
<td>Common Language Runtime</td>
</tr>
<tr>
<td>CLS</td>
<td>Common Language Specification</td>
</tr>
<tr>
<td>CTS</td>
<td>Common Type System</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>GSCM</td>
<td>General Service Composition Model</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IL</td>
<td>Intermediate Language</td>
</tr>
<tr>
<td>IT</td>
<td>Information technology</td>
</tr>
<tr>
<td>JIT</td>
<td>Just in time</td>
</tr>
<tr>
<td>MMS</td>
<td>Multimedia Message Service</td>
</tr>
<tr>
<td>OS</td>
<td>Operation System</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RSS</td>
<td>RDF site Summary (RSS 0.9/1.0)</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WS-I</td>
<td>WS-Interoperability</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
## 13 Reference List


7. **Colan, M.** (2004). *Service-oriented architecture expands the vision of web services.* IBM.


9.3 Use Cases


60
Appendix A  getNewEvent Web Service sourcecode

The following code implements the framework for a getNewEvent Web Service. This example takes a string as input (which should be a string in XML format), and it returns a message as a string in XML format. This is the only code needed for Visual Studio .NET. When building and publishing a WSDL file and a Web Service proxy is automatically created (Appendix B, Appendix C).

```csharp
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace SoccerService
{

    {
        [WebMethod]
        public string getNewEvent(string ContextAsXML)
        {
            Message newMessage = new Message();
            //include code here
            return newMessage.toXML();
        }
    }

    public class Message
    {
        private string text;
        //include other properties here

        public string toXML()
        {
            string messageAsXML = "";
            //code to convert message to XML
            return messageAsXML;
        }
    }
}
```
9.3 Use Cases

Appendix B  getNewEvent WSDL

This WSDL file in XML format is automatically created by Visual Studio .NET:

```xml
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:tm="http://schemas.xmlsoap.org/wsdl/tns/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/
  xmlns:xm="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://tempuri.org/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://tempuri.org/"
 xmlns:xmls="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xs:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <xs:element name="getNewEvent">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="ContextAsXML" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getNewEventResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="getNewEventResult" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getNewEventSoapIn">
    <wsdl:part name="parameters" element="tns:getNewEvent" />
  </wsdl:message>
  <wsdl:message name="getNewEventSoapOut">
    <wsdl:part name="parameters" element="tns:getNewEventResponse" />
  </wsdl:message>
  <wsdl:portType name="Service1Soap">
    <wsdl:operation name="getNewEvent">
      <wsdl:input message="tns:getNewEventSoapIn" />
      <wsdl:output message="tns:getNewEventSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="Service1Soap" type="tns:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  </wsdl:binding>
  <wsdl:service name="Service1">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      <soap:address location="http://localhost/SoccerService/Soccer.asmx" />
    </wsdl:service>
    </wsdl:definitions>
</wsdl:definitions>
```
Appendix C  getNewEvent Web proxy

The Web Service Proxy is created automatically in Visual Studio .NET. The proxy can be used to test the web service and to analyze the input and output of the Web Service. From the figure the getNewEvent() method take a string as input and also returns a string.

Service1

Click here for a complete list of operations.

getNewEvent

Test
To test the operation using the HTTP POST protocol, click the ‘Invoke’ button.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContextXML</td>
<td></td>
</tr>
</tbody>
</table>

Invoke

SOAP

The following is a sample SOAP request and response. The placeholders shown need to be replaced with actual values.

```xml
POST /SoccerService/Soccer.aspx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 92

SOAPAction: "http://tempuri.org/getNewEvent"

<?xml version="1.0" encoding="utf-8"?>

<soap:Body>
<getNewEvent>
  <ContextXML>string</ContextXML>
</getNewEvent>
</soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 91

<?xml version="1.0" encoding="utf-8"?>

<soap:Body>
<getNewEventResponse>
  <getNewEventResult>string</getNewEventResult>
</getNewEventResponse>
</soap:Body>
</soap:Envelope>
```