

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET  
FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG  
ELEKTROTEKNIKK



## HOVEDOPPGAVE

Kandidatens navn: Per Eilif Træen

Fag: Datateknikk

Oppgavens tittel (norsk):

Oppgavens tittel (engelsk): **Market Analysis of Web Service frameworks**

Oppgavens tekst:

Web Services are generic services offered by third party service providers. Examples of Web services are user profile services, security services, payment services etc. Web application developers can customize these services to create advanced applications. Recently there have been several initiatives in standardizing Web service development and provision. Web service frameworks are being developed in order to standardize Web service development and deployment. These frameworks are promising initiatives aimed at creating reusable Web-based applications and increasing inter-application communication. Examples are Microsoft's .NET initiative and Sun's Liberty Alliance. The aim of this task is to arrive at an analysis and comparison of existing Web service frameworks and standards. The analysis will be based on a literature study, testing of applications and architectures, development and design of a scenario, and possibly development of a demonstrator.

Oppgaven gitt: 20.januar 2002

Besvarelsen leveres innen: 17. juni 2002

Besvarelsen levert: 17.juni 2002

Utført ved: NTNU / Telenor FoU

Veileder: Monica Divitini (NTNU), Babak Farshchian (Telenor FoU)

Trondheim,

Faglærer



# Market Analysis of Web Service frameworks

Per Eilif Træen



## Abstract

Web services are generic programmable components available over the Internet from third party service providers. Based on open standards, Web services promises to enable communication between components independent of programming language and development platform.

Defining Web services as functionality, available through an open Internet standard, giving programmable access to the service independent of its implementation, this report reveals all truths about Web services. It shows how Web services are realised all the way from the service oriented architecture to the enabling open standards of the eXtensible Markup Language, Simple Object Access Protocol, Web Service Description Language, and Universal Description, Discovery and Integration. Web services differ from components with respect to the looser coupling and coarser granularity, and have a wide variety of application areas, ranging from Enterprise Application Integration to finding the best mortgage rates.

Imagining a scenario where a university presents a portal to students where they can access information about all the courses they take, this could be realised using Web services. This would enable the portal to always present up to date information about the various courses without having to replicate and maintain this information. It would also allow for integration with other universities or external companies.

Microsoft's .NET Framework, the Java 2 Enterprise Edition and Sun Open Net Environment represent some of the frameworks and architectures that can be used for Web service development and deployment. These three technologies are subject of a comprehensive evaluation framework, which include evaluation criteria such as security, mobility, tool support, interoperability and scalability.

The evaluation aims at presenting a comprehensive overview of the capabilities of each technology, without biasing towards any particular choice. The results can then be used to determine which technology is best within a given context where certain criteria are important than others. Results do however indicate that Java 2 Enterprise Edition and Sun Open Net Environment are best suited for large enterprise solution, whereas the .NET Framework offers the best solution for smaller scale Web services.

Investigating the overall design of the university portal scenario reveals that a solution where one uses the Sun Open Net Environment Portal Server to realise all portal logic, including user authentication and interface generation, would be the most tempting approach. Individual Web services with course information can be developed using any of the three technologies, but recommendations lean towards the use of the .NET Framework.

The results are limited by the fact that they are strictly theoretical; no concrete testing has been conducted. This is suggested as a task for further work, together with the development of a prototype scenario and the testing of additional frameworks and architectures.



## Table of content

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Context .....	1
1.2	Motivation .....	1
1.3	Report structure .....	1
<b>2</b>	<b>Introducing Web services .....</b>	<b>3</b>
2.1	Background .....	3
2.2	Defining Web Services.....	4
2.3	Architectural Principles.....	5
2.4	Web Services versus Components .....	6
2.5	Web Services application areas.....	7
2.6	Benefits.....	9
<b>3</b>	<b>Scenario .....</b>	<b>11</b>
3.1	Description .....	11
3.2	Uses cases.....	11
3.3	Requirements.....	13
3.4	Approach .....	14
3.5	Services .....	14
3.5.1	Web Service for university course .....	14
3.5.2	Web Service for university course web portal .....	15
3.5.3	Additional services .....	15
3.6	Overall architecture .....	16
3.7	Development requirements .....	17
3.8	Future possibilities .....	18
3.9	Concluding remarks .....	18
<b>4</b>	<b>Web Service Technologies .....</b>	<b>21</b>
4.1	Enabling standards and technologies .....	21
4.1.1	eXtensible Markup Language (XML).....	21
4.1.2	Simple Object Access Protocol (SOAP) .....	22
4.1.3	Web Service Definition Language (WSDL) .....	27
4.1.4	Universal Description, Discovery and Integration (UDDI) .....	30
4.2	Web Service Frameworks and Architectures .....	32
4.2.1	Microsoft .Net Framework (.NET) .....	33
4.2.2	Java 2 Enterprise Edition (J2EE) .....	35
4.2.3	Sun Open Net Environment (Sun ONE) .....	37
<b>5</b>	<b>Evaluation of technologies .....</b>	<b>41</b>
5.1	Evaluation criteria .....	41
5.2	Evaluation.....	43
5.2.1	Security.....	43
5.2.2	Mobility.....	51
5.2.3	Supported platforms .....	52
5.2.4	Supported protocols.....	53
5.2.5	Component definitions .....	53
5.2.6	Legacy system integration.....	53
5.2.7	Shared context .....	55
5.2.8	Interoperability .....	57
5.2.9	Ease of use.....	57
5.2.10	Scalability.....	58
5.2.11	Deployment .....	58
5.2.12	Tool support .....	59

5.2.13	Modelling tools .....	62
5.2.14	Billing .....	63
5.2.15	Availability and price .....	64
5.2.16	Summary .....	65
5.3	Evaluation related to the university scenario .....	67
5.4	Conclusion.....	70
<b>6</b>	<b>Overall design .....</b>	<b>73</b>
6.1	General design considerations.....	73
6.1.1	Interactions .....	73
6.1.2	Course Web service.....	74
6.1.3	Portal Web service .....	77
6.1.4	User Interface Web service .....	78
6.1.5	User Profile Web service.....	79
6.1.6	Calendar Web Service.....	79
6.2	Microsoft .NET .....	79
6.3	Java 2 Enterprise Edition.....	81
6.4	Sun Open Net Environment .....	82
6.5	Conclusion.....	83
<b>7</b>	<b>Conclusions and further work .....</b>	<b>85</b>
7.1	Overall conclusions .....	85
7.2	Limitations .....	86
7.3	Suggestions for further work.....	86
<b>Appendix A: Bibliography.....</b>		<b>89</b>
<b>Appendix B: Glossary .....</b>		<b>93</b>

## List of figures

Figure 1.	The tiresome process of registering user information .....	3
Figure 2.	The principle behind a user authentication Web Service .....	4
Figure 3.	The service-oriented architecture.....	5
Figure 4.	Generic Web Service Architecture. ....	6
Figure 5.	Architecture of the web services for the scenario.....	16
Figure 6.	The Web service technology stack .....	21
Figure 7.	The SOAP message structure. ....	23
Figure 8.	The .NET execution model.....	33
Figure 9.	The Microsoft .NET framework.....	35
Figure 10.	The basics of the Java programming language.....	36
Figure 11.	The J2EE architecture.....	36
Figure 12.	Solaris Operating Environment. ....	38
Figure 13.	The Sun Open Net Environment Platform.....	39
Figure 14.	The Identity and Policy architecture.....	44
Figure 15.	Authentication Component Architecture.....	47
Figure 16.	SSO architecture. ....	47
Figure 17.	The J2EE-J2ME Application Architecture .....	51
Figure 18.	.NET Passport authentication flow. ....	55
Figure 19.	The .NET Framework with Visual Studio .NET.....	62
Figure 20.	High-level model of the interactions in the university scenario.....	73

## Preface

This report is the diploma that is carried out by students in the final term of their five-year study at the Norwegian University for Science and Technology in Trondheim. It completes the system developments course taken at the Department of Computer and Information Science.

The chosen task for the diploma has allowed me to look into the exiting concept of Web services and many of the standards and technologies involved in realising Web services. Most of the major and minor software vendors around the world have an interest in Web services and they all present varying views and approaches to Web services. Still, there are some key specifications and technologies that form the foundation of Web services and these were the starting point of this analysis after first having uncovered what the Web service concept is all about.

From there on it has all been a matter of setting the mind on Web services, designing a Web service scenario and investigate existing frameworks and architectures for Web service development and deployment. This has first and foremost been a literature study and has involved browsing almost unlimited amounts of text. The challenge has then sometimes been to filter much of the information and catch the essence in order not to diverge from the given task.

The task was designed by Babak Farshchian at Telenor Research and Development in Trondheim together with Monica Divitini at the university. These two have also been my mentors throughout the work and I would like to express my gratitude to them for providing me with valuable feedback on the report. Without them this report would have been much harder to complete.

Per Eilif Træen



# 1 Introduction

This report begins with a general introduction describing the context and motivation behind the work presented in the report, together with a description of the report structure.

## **1.1 Context**

Web services are generic programmable components offered by third party service providers. Examples of Web services include user profile services, security services, payment services etc. Web application developers can customize these services to create advanced applications. The purpose of this report is to investigate Web service frameworks, which have been developed to standardize Web service development and deployment. This will be solved using an evaluation framework to analyse and compare existing Web services frameworks and standards. The work also includes the development and design of a scenario that can be used for reference in the analysis.

## **1.2 Motivation**

Web services hold the promise of seamless cross-platform integration between applications, independent of programming language, and have therefore received much attention from software vendors all over the world. Web service is the next big thing within distributed computing over the Internet and it is therefore of interest to look into what the Web service concept has to offer and how.

In addition to familiarisation with the Web service concept and the surrounding technologies, the work in this report has involved learning many of the features the .NET platform, Java 2 Enterprise Edition and the Sun Open Net Environment.

To accomplish this a comprehensive and thorough literature study has been conducted. An extensive evaluation framework had to be developed in order to get a detailed and unbiased evaluation of frameworks and architectures for developing and deploying Web services. A scenario that can be realised using Web service technology was described, and the evaluation could then be extended to investigate how well suited different frameworks and architectures are for realising such a scenario.

## **1.3 Report structure**

The report begins with an introduction of Web services in chapter 2. A definition of the term will be presented and a description of the architecture of Web services is also presented. A comparison of Web services with the component technology is given, and then an overview of Web service application areas.

Chapter 3 presents a scenario consisting of a portal for access to course information at a university. The scenario is described with use-cases and functional requirements. A Web service approach is suggested to realise the scenario, with every part of the scenario implemented as Web services. The different Web services are then described and the overall architecture for the scenario is drawn. The chapter is finished by looking at the development requirements for the scenario, which are of importance to the later evaluation, and some suggestions for further possibilities for the scenario.

Chapter 4 presents the technologies one will encounter when dealing with Web services, either as a consumer or a provider. First of all are the enabling technologies, which are the key to the Web service concept. These technologies consist of the eXtensible Markup Language, Simple Object Access Protocol, Web Service Definition Language, and Universal

Description, Discovery and Integration. They are all based on commonly agreed standards, and together they comprise the Web service technology stack.

The chapter continues with a look at frameworks and architectures that can be used to develop Web services: the Microsoft's .NET Framework, Sun's Java 2 Enterprise Edition and the Sun Open Net Environment. These technologies are subject for evaluation in the coming chapters. Chapter 5 contains an evaluation framework that is used to evaluate the candidate technologies. First the criteria for evaluation are listed and described, and then these criteria are used in the evaluation of frameworks and architecture to determine how well each criterion is satisfied for each technology. The evaluation is then related to the scenario, by looking at the relevant criteria for developing the university portal.

In chapter 6 an overall design for the scenario is suggested. It begins with identifying the interaction patterns within the scenario and then presents an overview of the interfaces of the various Web services. This is further mapped on to the three technologies by investigating how each of them would be used to solve the implementation of Web services for the scenario. The chapter ends with a discussion around what technologies should be used to develop the university portal scenario.

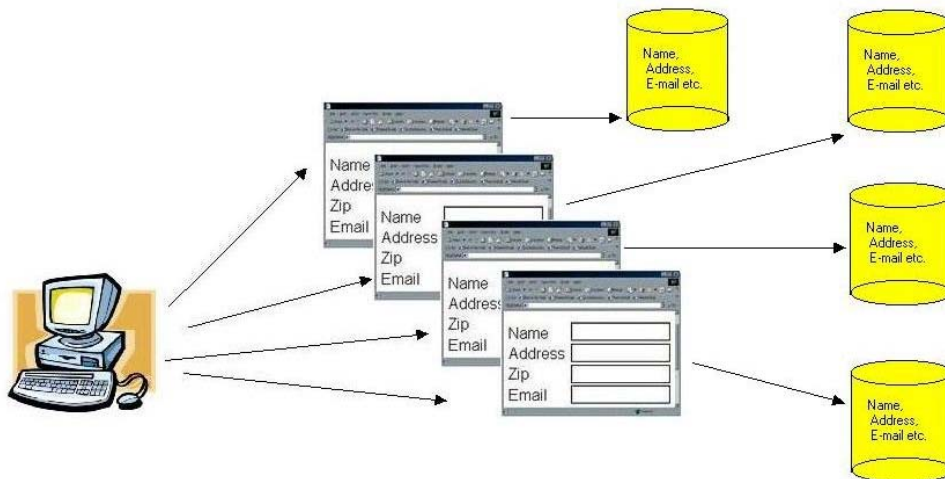
Chapter 7 sums up the report by looking at conclusions drawn throughout the report, the limitations of the results and making some suggestions for further work.

## 2 Introducing Web services

This chapter gives an introduction to Web Services. Some background is presented before a definition of the term Web Service is given. The architecture surrounding Web Service follows next, both from the provider-consumer perspective and for the actual Web Service. It is also shown how Web Services compare to the components technology, and finally some examples of Web service application areas are presented.

### 2.1 Background

Before going into the definitions and terminology surrounding Web Services a brief example of how Web Services can be used is presented. Today there are numerous Web sites requiring users to register with name, address, and telephone number etc., e.g. e-commerce sites. Each time a person wants to register on such a site he/she needs to re-enter all the registration information, a quite irritating procedure. There is also the issue of changing the user details, for example when the user moves to a new address. This will require the user to update his/hers information on all the sites where he/she is registered. Figure 1 illustrates this process.

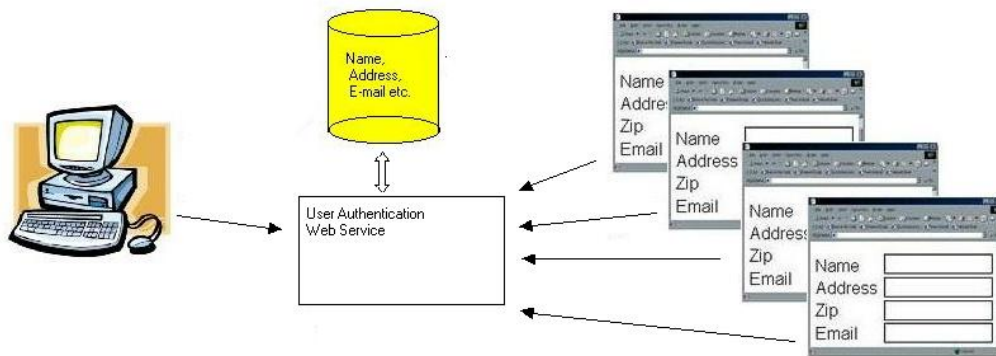


**Figure 1. The tiresome process of registering user information**

It would of course be much more convenient if the user only had to enter the registration information once and then let all the sites where he/she wanted to register have access to this information. This is exactly one of the things that can be achieved with Web Services.

Microsoft's .NET Passport<sup>1</sup> is an example of such a service where users can create one sign-in name and password for use across participating .NET Passport sites. Figure 2 illustrates how this works.

<sup>1</sup> <http://www.microsoft.com/my services/passport>



**Figure 2. The principle behind a user authentication Web Service**

This also shows another benefit of Web Services; the Web sites that earlier had to provide logic for user registration and store the user data are now relieved of this since the logic and the data resides at the Web Service provider. It also shows that a Web Service can have different users with different purposes of use. Another example to illustrate the use of Web Services can be found in for instance a small application for calculating stock values.

Consider that someone have developed a small program for keeping track on the total value of his/hers stocks (one can also imagine using a spreadsheet like in Microsoft Excel). Once a day, after the stock market is closed this person want to see how much the stocks are worth. So he/she goes to the source of information, read the values for the stocks of interest, and fill these values into the program. This task will then have to be repeated daily, unless a way to do this automatically is used. With a Web Service that provides the latest stock prices this can be achieved. It only requires the person to find a service that provides that functionality and adapt the program to do the proper function-calls.

It is particularly in the area of e-commerce that Web Services are meant to make the most significant impact. The previous attempts at distributed computing, like CORBA, JAVA RMI, and DCOM, have resulted in systems that consisted of too tightly coupled components and were not well suited for open reliable business-to-business communication [1]. These approaches require too much compromising and often a considerable amount of changes to the implementation. Web Service communication, on the other hand, is based on open standards and the driving force behind the Web Service technology is seamless cross-platform interoperability, regardless of development platform or programming language.

## **2.2 Defining Web Services**

After that brief introduction of Web Services it is time to define what a Web Service really is. IBM, Microsoft and Sun all give different definitions of the term Web Service, both in length and detail. Glass [2] defines a Web Service as a collection of functions that are packaged as a single entity and published to the network for use by other programs. Kirtland [3] describes a Web Service as programmable application logic accessible using some standard Internet protocols. Reichardt [4] says Web Services are self-describing components that can discover and engage other Web Services or applications to complete complex tasks over the Internet. Many other definitions also exist. Looking at the essence of the different definitions the general features of them is:

- A Web Service is functionality,
- available through an open Internet standard,
- giving programmable access to the service,
- independent of its implementation.

The programmable access allows you to embed the Web Service into your own application. This makes it possible, among else, for one Web Service to query another Web Service and get updated information in real-time.

Like components, Web Services present black-box functionality that can be reused without worrying how the service is implemented. One must, however not think of Web Services as another type of components. The difference lies in the fact that Web Services are not accessed via object-model specific protocols such as the distributed Component Object Model (DCOM) or Java's Remote Method Invocation (RMI). Instead, Web Services are accessed via Web protocols that can be found anywhere, such as HTTP and XML. The Web Service has a strictly defined interface regarding the messages the service can accept and generate. This means that consumers of a Web Service can be implemented on any platform in any programming language, as long as they create and consume the messages defined for the Web Service interface.

### 2.3 Architectural Principles

The most fundamental architectural principle underlying Web Services is the service-oriented architecture (SOA) [5]. The architecture focuses on how the components of the service are organized and described to provide a dynamic and automatic discovery and use. Figure 3 shows the SOA.

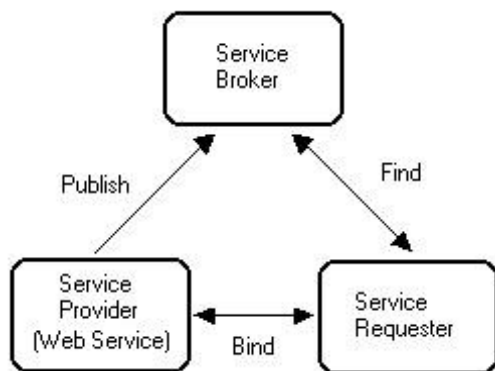


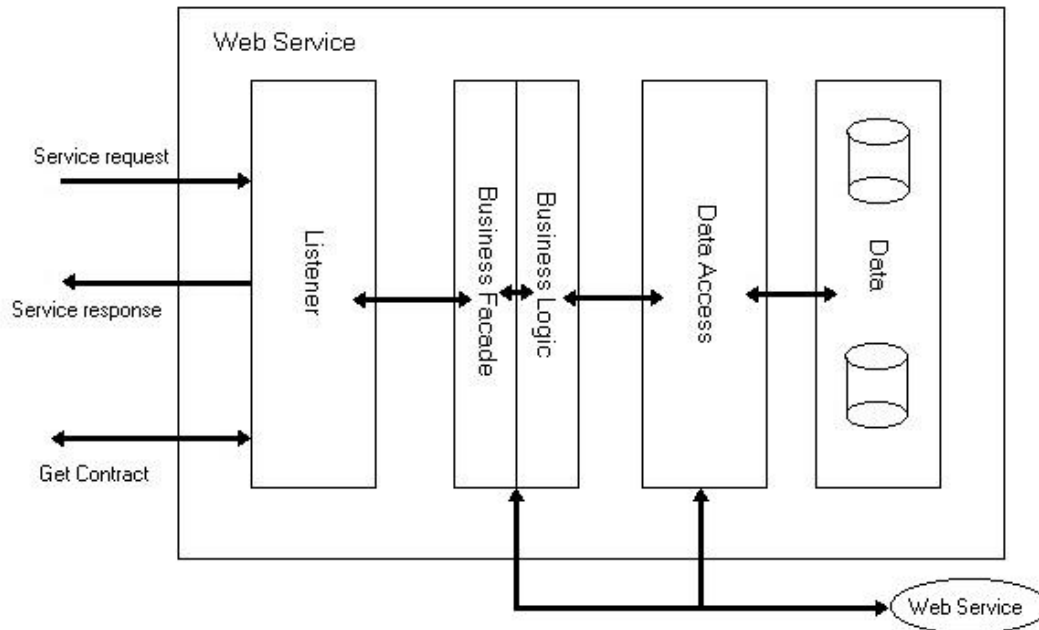
Figure 3. The service-oriented architecture

The three components in the architecture are:

- The service provider
- The service broker
- The service requester

The service provider publishes the availability of its services at the service broker using a *publish* operation, and responds to request from users of the service. The service broker is responsible for registering and categorizing the service providers and offer search services. A service requester uses the service broker to locate the wanted service using a *find* operation, and then makes use of the service through the *bind* operation.

The actual Web Service lies within the service provider component. Kirtland [3] presents a generic architecture for a Web Service. The architecture is shown in Figure 4.



**Figure 4. Generic Web Service Architecture.**

The architecture is divided into five logical layers:

- The data layer: stores the information needed by the Web service.
- The data access layer: presents a logical view of the physical data in the data layer to the business layer. It isolates business logic from changes in the underlying data store and ensures data integrity. The data does not necessarily reside at the service provider but can be data retrievable from another Web service.
- The business layer: contains the business logic of the Web Service. It is often subdivided into two parts – the business façade and the business logic. The business façade provides a simple interface that maps directly to the operations exposed by the Web Service. The business façade uses services provided by the business logic. All the business logic does not need to lie within the Web service architecture but may also be provided by another Web service.
- Listener: The Web Service listener is responsible for the interaction with the client applications. It receives incoming messages containing requests for services, parses the messages, and dispatches them to the appropriate method in the business façade. Whenever there is a response the listener packages the response from the business façade into a corresponding message and sends it to the client. The listener also handles requests for contracts and other documents describing the Web Service.

## 2.4 Web Services versus Components

There are two important aspects in which Web Services differs from components: granularity and coupling [4]. Granularity is associated with the complexity of the description of a service. A simple function call in Unix can be very specific and is said to have a fine level of granularity. Business processes (e.g. purchase orders) that involve several actors can introduce levels of complexity that are less well defined, giving a granularity that is more coarse.

Coupling refers to the nature of the interface between the service provider and the service consumer, and is concerned with how an implementation change affects the consumer. In a tightly coupled application relationship a change in a function call in the component would require all applications using that call to change.

Components then are used for fine-grained, tightly coupled tasks, whereas Web Service can be used for coarse-grained, loosely coupled tasks. Another way of saying this is that components are capable of interoperating with other components designed using the same object model. Web Services are designed to interact with any other component, regardless of its origin, as long it is contained in a self-describing XML-wrapper.

The following table sums up the difference between components and Web Services:

<b>Components</b>	<b>Web Service</b>
Tightly coupled software applications	Loosely coupled software applications
Intranet-based software	Internet-based software
Transport mechanism TCP/IP	Transport mechanism HTTP
Fine granularity	Coarse granularity

**Table 1. Comparing components and Web services.**

This shows that Web Services do not completely replace components rather they complement them. The choice between components and Web Services must be based on the nature of the service to be implemented.

Another distinction is between Web Services and Application Service Providers (ASP). An ASP typically provides the complete solution. The whole application with user interface, workflow, business and data components are bound together providing a working solution. There is little or no room for customisation by the customer. Access is typically through a browser; alternatively the ASP provides software that can be downloaded to the customer's machine. Web Services provide a more flexible solution. The core of the application remain on ASP's machines but are accessed programmatically via the Web Service interface. The customers can then build their own custom business processes and user interfaces, and are also free to incorporate other Web Services to suite their needs. One may also choose to create a Web Service as an aggregation of other Web Services, providing a higher-level set of features (e.g. a travel organiser Web Service that books airplane tickets, makes hotel reservations and orders a rental car).

## **2.5 Web Services application areas**

As mentioned earlier one of the areas in which Web Services will prove beneficial is within e-commerce. The main features of e-commerce is the availability, anyone can place an order regardless of location as long they have a computer that is connected to the Internet. The growth of e-commerce has made it possible to buy almost everything via the Internet. This again puts a lot responsibility on those who would like to succeed in e-commerce, particularly with respect to price and delivery.

Customers are often interested in a wide assortment of merchandise and to realise this the e-commerce site needs to collaborate with different suppliers. In order to provide delivery of goods to customers all over the world the e-commerce site requires a proper logistics system as well.

Both these areas require effective communication and sharing of information and a system is needed to facilitate this. Looking at how this is done without Web Service one could imagine a business in e-commerce that offers a variety of products. In one possible approach this business manages its own storage to which it orders goods from the different suppliers. It is run by staff that monitors stock supplies and place orders accordingly. The other approach is to forward orders from the customers directly to suppliers, thereby removing the need for own storage. These approaches do however both suffer from the same limitations. The storage is not directly attached to the web interface where the customers place their orders. This means

that the customer placing the order has no idea whether there is anything in stock of the merchandise he is after. The business running the e-commerce site has the same problem since it cannot know how much the supplier has in stock. The customer will eventually discover that the order he has placed arrives many weeks later than expected. This again makes it hard for e-commerce businesses to make promises about minimum delivery time and live up to it.

The introduction of Web Services solves this problem. Any business can expose their business processes using Web Services, independent of how the processes are implemented. This means that the e-commerce business can have direct access to stock information at the suppliers. The information can also be presented directly to the customer at the shopping interface, so that the customer is free to choose another e-commerce site or at least find out when the product is available again. In the case where the business keeps its own storage automatic ordering is also a possibility. The inventory system can be connected to an order Web Service from the supplier so that as soon as the number of items drops below a certain level an order is placed for that item.

The same principles apply for the logistics function. A logistic Web Service can track the location of the products the customer has ordered at any time. Orders can automatically be forwarded to the logistic department which then has immediate knowledge of what must be transported and where.

From a customer's point of view an improvement can also be found in the introduction of so-called electronic wallets, or just wallets. These allow the user to have his credit information in one place together with all the orders he has placed. The wallet can be used at any e-commerce site supporting that service and it relieves the customer of having to leave his credit card details at every e-commerce site he uses. Another aspect of this is the possibility of immediate credit checking, using a credit checking Web Service.

This illustration of Web Services used for e-commerce shows that Web Services not only improve the business-to-customer possibilities, but also give a more solid business-to-business application integration. This example showed the latter in the relation between the shop, the storage and the logistics.

Considering that the shop, storage and logistics were all part of the same corporation this shows how Web Services can be used for Enterprise Application Integration (EAI). The different departments may use different kinds of applications, but by wrapping these into Web Services the applications can communicate language-independently and the business legacy system can be tightly integrated.

A real world example of EAI using Web services is a solution selected by British Telecom. In order to enable interoperability between existing legacy applications, they encapsulated the legacy applications into XML Web services. By having applications wrapped into Web services one does not just improve the interoperability between existing applications but also prepare the current applications for communication with future applications.

Web Services also promote Intranet-solutions such as desktop access to internal information systems. An anonymous bank in Northern America decided to use Web services to make available information contained within its internal information system. The purpose was to make all this information available at the employees' desktop.

While these examples mainly show how Web services can be used within e-commerce and business integration, Web services span a much wider field. The authentication Web service from Microsoft, the .NET Passport, has already been mentioned. Together with .NET Alerts, .NET Passport constitute the foundation for .NET My Services<sup>2</sup> (formerly known as Hailstorm), a family of Web services that will let users store key personal information

---

<sup>2</sup> <http://www.microsoft.com/myservices>

securely and control access to it. These services are due to be available sometime during 2002 and the list of services include, among else, .NET Contacts (electronic relationship/address book), .NET Locations (Electronic and geographical location and rendezvous), .NET Calendar (Time and task management) and .NET Documents (raw document storage). Adobe Altercast<sup>3</sup> is an image server offered as a Web services. Adobe has a wide range of network publishing tools that among else make use of intensive graphics. Managing such graphic-intensive Web sites can be quite a cumbersome procedure, and failure to maintain the Web sites often result in visually unattractive or outdated sites. With Adobe Altercast customers receive an automated solution to this. Altercast can be integrated into customer workflows by installing it as a Web service on the customer's Intranet or as an Internet service. This will allow any other Web service to access the image manipulation services and use these to more or less automatically maintain the graphics intensive Web sites. The XMethods Web site<sup>4</sup> contains an overview of all publicly available Web services, with information about publisher and implementation tool in addition to descriptions of the various services. Examples of services found on XMethods are the "Mortgage Comparison Calculator" that helps determine the best mortgage for people, the "Encryption" Web service that encrypt/decrypt text in multiple level using the chipertext method and "Headline news" that supplies the latest news coverage from various sources. All of this illustrates the wide variety of applications for Web services, from large-scale business integration and e-commerce to smaller user level services such as stock quotes or weather forecasts. What it all boils down to is that Web services can be used for all kinds of information exchange and information processing using open standards, regardless of the amount and kind of information.

## **2.6 Benefits**

Having presented the concepts of Web services and what can be achieved using Web services, this chapter sums up by looking at the most significant benefits of Web services. Some of the attractive properties of Web Services are:

- **Interoperability:** Any Web Service can interact with any other Web Service using standard protocols for communication supported by all major (and minor) software vendors. Web Services can be written in any language, there is no need to change development environment to start producing and consuming Web Services.
- **Ubiquity:** Web services communicate using HTTP and XML, technologies that are present in most devices. Any device that supports these technologies can both host and access Web Services.
- **Low Barrier to Entry:** The concepts of Web Services are easy to understand, and vendors like IBM, Sun and Microsoft offer free tools for Web Service development. And as mentioned above, there is no need to change development environment.
- **Industry support:** All major vendors support the technologies surrounding Web Services and collaborate in the development, improvement and standardisation of these technologies.

---

<sup>3</sup> <http://www.adobe.com/products/altercast/>

<sup>4</sup> <http://www.xmethods.org>



# 3 Scenario

This chapter presents a scenario that will be used in the evaluation of Web Service development frameworks and architectures later in the report. The aim is to have a realistic, familiar and not too complex scenario that can be realised as one or more Web Services.

## 3.1 Description

The scenario is a web portal for access to courses on a university. As it is today access to information about the different courses is through web pages provided by the teachers, one for each course. To locate these pages the student can either click his way from the start-page for the university, search the university web pages or use a URL provided by the teacher. There is no relation between the pages for the various courses and no form of collection of related courses (e.g. a Web page for the courses in 3. year Computer Science).

The information on these pages can be:

- time for lectures
- curriculum (books/articles used in the course and support-literature)
- progress plan (content of lecture, often related to the relevant chapters in the books)
- exercises (descriptions, hints, deadlines)
- information related to exams

The problem with this is that the information is spread on different web pages and might have different representations. There is no way to gather the information from different courses and present them collectively, not unless someone replicate all the information and put it on a new web page. This is not an adequate solution since it would require administrators to constantly check the individual courses for updates and then apply these to the collection web page. A web service solution would solve this because it gives the administrators programmable access to the real source of information (no need to replicate) and thus always the latest version of the information one requires. Web services also provide developers with the possibility of creating higher-level web services from other web services. In this context this would be a web service that lets students select the university courses, represented as web services, of interest and presents information from all of these in one place.

## 3.2 Uses cases

Before presenting the requirements of this scenario some different uses cases for the scenario will be shown. The use cases cover many of the aspects surrounding the scenario and show more clearly what the requirements for this scenario are. There will be three use cases, one for a student at the start of the semester, one for a student at some arbitrary point of time within the semester, and one for a teacher wishing to update information about the course he is teaching.

### Student at the start of a semester

The summer holidays have ended and Peter, a third year building engineer student, is raring to go for another semester. But before he can get started courses must be chosen. Peter therefore sits down before a PC connected to the Internet and enters the address for the university's course portal web service. At this point he reaches a greeting page where he logs on to the portal using the username and password assigned to him by the university when he first began studying. Peter reaches his own personalised portal, where he can search for courses and register for these. In this case Peter knows that there are 3 courses that must be chosen in

order to complete degree he is after, and in addition Peter must chose from one out of five other courses. Peter enters the code for the courses that must be chosen, three letters and four digits, one at a time, and for each course he is presented information regarding the course: what it is about, when and where lectures are, who the teacher is and where he/she can be contacted etc. Peter can then choose to register for the course and upon doing this, the course is added to his portal and he gains access to more detailed information about the course. He also chooses one from the five optional courses and now has the minimum number of required courses for this semester.

Next, Peter checks the calendar on the portal to see when he has lectures in all four courses and also other important dates related to the courses. Here he sees that the optional course he selected has lectures at the same time as one of the three compulsory courses, so he goes back and selects another that does not crash. On to the calendar again he sees that at the end of the week an exercise will be handed out. There is no time to waste Peter thinks and tries to save his preferences and log off. In this case however, this is not as easy as he thinks. The problem is that Peter has made a mistake when adding courses so that he has not selected all the required courses for this semester. Peter's selected courses were checked against the study plan for his degree when saving the preferences and this check discovered a mismatch in his selection. Peter then can either log off without saving the changes he has made or he can go back and change his selection. Peter opts for the last alternative and corrects his errors. He now also sees from his calendar that he can now pick the optional course he originally wanted and does so.

Having completed his selection correctly, Peter logs off and runs along to the campus bookstore to buy all the books he need for this semester, information he got from the web portal.

### **Student at arbitrary time in semester**

Karen, a second year biology student, wakes up one Monday morning, tired after another hard student weekend. She remembers vaguely that this week is going to be a busy one, but she is not quite sure about all that is going to happen. Therefore she turns on her PC and logs on to the university's course portal web service. Having logged on successfully she notices that there are two new messages so she navigates to the message board for all the courses she are registered for. One of the messages concerns a cancelled lecture while the other tells of a new exercise that has been made available in one of her courses. Karen moves on to the page for that course to study more detailed what the exercise is about.

Karen then decides to check the calendar for this week and sees that there are two exercises due for delivery this week, one on Wednesday and one on Thursday. She hardly has time to complete both by then so she has to prioritise. Karen checks the courses she has the exercises in to see how many she has had approved earlier, and decides to drop one of the exercises since she already has completed the required number of exercises for that one course.

Back to the calendar she sees that the cancelled lecture has also been marked and in addition a local industry has a presentation on Wednesday that could be of interest. The dentist appointment that Karen entered last week is no longer away than Tuesday. Finally she discovers that she has a lecture in just half an hour so it is time to log off and get dressed.

### **Teacher updating information on course**

The exam period is fast approaching, and Paul Jones, a mathematic professor at the university needs to make a final update on the course he is teaching. He sits down behind his PC and enters the address of the web service for his course, and then types the password necessary for read- and write access.

The first thing Mr. Jones does is to make available the exams from earlier years together with proposals for solutions. Next he checks the list of registered students to see if there are any who have not completed the required number of exercises to be allowed to take the exam. There are quite a few students missing one approved exercise so he prepares an extra exercise for them to take and creates a message for the students of concern. Then he needs to update the curriculum so that it properly reflects the theme of his lectures and gives the students an indication of what is expected from them. He also decides to hold an extra lecture to refresh the students of things he went through earlier in the semester and makes a message about this in addition to an event that will be added to the students' calendar. Everything is now updated like Mr. Jones wanted it to be so he decides to log off from the course Web service and starts to prepare the questions for this year's exam.

### **3.3 Requirements**

Based on the use cases from the previous section it is now possible to identify the most significant requirements for this scenario. The requirements are seen mainly from the perspective of the students using the portal, but there are also requirements from teachers updating the course information and also from external users interested in just a few courses and not the portal.

Beginning with the most important requirements, those from the students, these are:

- personalised access/registration: using the username and password provided by the university, students can access their own personalised portal and upon logging off changes to the user's preferences are saved. Users not registered as students at the university can register here and are billed the appropriate fee.
- search for courses: students can search for courses using a course code consisting of three letters and four digits. The result of such a search yields information about that course: briefly what it covers, when and where lectures are held, and who teaches it. It should also be possible to search for courses by name/theme, giving a list of all matches where one can pick a course and get the same information about that course.
- register for courses: having found a course of interest the student should be able to register for this course and at the same time add the course to the portal. The choice of courses must be checked against the study plan for the degree the student is taking, and if the selection is invalid the student must be notified of this.
- access to detailed information regarding each individual course: after registering for the course the student gains access to more detailed information like curriculum, exercises, number of approved exercises and messages related to the course.
- a message board for all registered courses: the portal should have a message board containing messages from all the courses the student is registered for. When logging on the student should be notified of any new messages.
- a calendar for events concerning all registered courses: the portal should have a calendar containing all events related to the courses the student registered for. An event can for example be a lecture, a deadline for an exercise or the exam date. The student must also be able to add his/hers own events, e.g. doctor appointments or group meetings.

Next come the requirements from the teachers point of view:

- effortless: teachers do not want to spend too much time providing students with information relevant to the courses they teach. One cannot expect all teachers to be able to implement their own Web Service for doing this. Therefore a framework must

be provided so that teachers easily can present whatever they feel is related to their course.

- easy maintenance: just as the initial information provision should be straightforward for the teachers, so should the maintenance of this information be. Adding, removing or updating information should be able to do at any given time, as long as the teacher has a device to access the course information.
- restricted access: in both the aforementioned points the restriction of access is important. The teachers of course want students to see their information but only the teachers should be able to change it. Therefore some kind of user authentication is required to ensure that only the persons involved in teaching the course has write access to the information.

There may also be users external to the university who may be interested in information from one or a few courses. These users can for example be companies or other universities. They are not interested in any access through the portal but rather directly to the courses, but should still have access to the same information as any of the students have. Also, since they are not part of the university they need to be billed for the access to the courses.

### **3.4 Approach**

The approach in this chapter is to imagine the scenario implemented as a set of Web services, with one service for each course and a service that acts as a portal for the students. The portal presents information from a course to the student by communicating with the service for the course. The teacher adds information to the course web service.

There are three main actors in this scenario: student, teacher and university. In addition come external actors like other universities or companies. The university acts as a service provider both for teacher, student and any external interested party, making them the service consumers. The university provides the course web service that the teacher interacts with by adding, removing and updating information about the course he/she is teaching. The course Web service is also where the external consumers interact, retrieving information for the specific course they are interested in.

For the student the university provides the portal that the student interacts with by adding and removing courses, requesting information about the courses and read messages. The student is also able to see some information collectively, like the exam dates for all the courses he/she has chosen. The university must also provide user authentication functionality for the student and a user-interface based on what device is used to access the service.

The university may also choose to include some external service providers to realise some of the wanted functionality, for example by including a calendar web service.

### **3.5 Services**

The attention is now turned to the actual Web Services needed to realise the scenario. This section looks at the functionality provided by the different services and whom they interact with.

#### **3.5.1 Web Service for university course**

The role of this Web Service is to present information related to a specific university course. It provides functionality for the teacher to add, remove and modify information related to a course. It also provides functionality for the portal service to access information and present it to the student, and must also present information directly to any interested external users.

A course is uniquely identified with a course code, consisting of three letters and four digits. For the teacher to interact with the service he/she must provide this code and a password, thereby protecting it from unauthorized access.

A course consists of the following elements:

- description: a general description of the course
- exercise: contains text describing the exercise, deadline for delivery and status (approved, not approved)
- lecture time: contains day, time of day and location for lectures
- curriculum: describes the literature that is part of the course
- exam date
- progress plan: describes the content of lectures related to the curriculum
- message
- student: students that are registered for the course

For each of these elements there should be associated *add*, *remove*, *update* and *get* methods, e.g. *addExamDate* and *removeMessage*.

For the external users there must be some form of billing for the use of the service. In this scenario a one-time registration fee is most suitable.

### 3.5.2 Web Service for university course web portal

This is the service the students use to request information regarding the courses they are interested in. It interacts with the student through a user interface (see next subsection) and also with the course services the student has chosen.

The portal is tailor-made for the student and must remember the preferences of each student.

The portal consist of the following elements:

- course: the courses selected by the student
- message board: contains messages from all courses the student has chosen
- user interface: presents information to the student and expose the expected functionality
- calendar: gives an overview of important dates related to the selected courses
- user profile: contains the individual student's login details and preferences

The student must be allowed to search for courses using the course code, but it should also be possible to browse through a list of available courses. The student's choice of courses must be checked against the proper study plan to verify that the selection of courses is valid. The portal provides the *get* methods that maps directly to the corresponding *get* methods in the course service. These methods return the required information and present it to the student via the interface.

The portal must also handle the registration of students to the courses. The registration adds to the student element of the course and links the user profile of the student to that course.

The message board contains the messages concerning the courses the student have selected.

The portal queries all the chosen courses for messages and sort them by date and present them to the student.

The final three elements, calendar, user profile and user interface are presented in the following subsection.

### 3.5.3 Additional services

In addition to provide a collection of course Web Services, the portal could also incorporate some additional services.

One is a calendar Web Service, a service that can be provided by any external calendar service provider. The objective of the calendar is to present the student a timetable with important dates and time of lectures from all the courses the student has selected. It should notify the student of both conflicting events and important dates. Depending on the functionality of the calendar service, it should also provide methods for letting the students add their own personal information, like dentist appointments or individual group meetings. The user profile service provides login functionality to the portal and storage of user preferences. It is based on the username and password that students use on the university to access computers and e-mail accounts etc. All user details are stored in the university database. Included in this information is also the study plan associated with each student, which is used to verify the choice of courses. Students not currently registered at the university can also be registered using this service, adding their details to the database. The user interface service provides a user interface based on the device that is used to access the portal. It displays the information requested by the user and dispatches user actions to the appropriate methods in the portal service. The amount of information and functionality the user interface can provide depends on the device it is intended for. Access via a web browser would provide more than access via mobile phone. Like the user profile service, the university also provides this user interface service.

### 3.6 Overall architecture

Based on the description of the scenario and the services that should be used to realise it, the architecture for the scenario and the relationship between the services should be as illustrated in Figure 5.

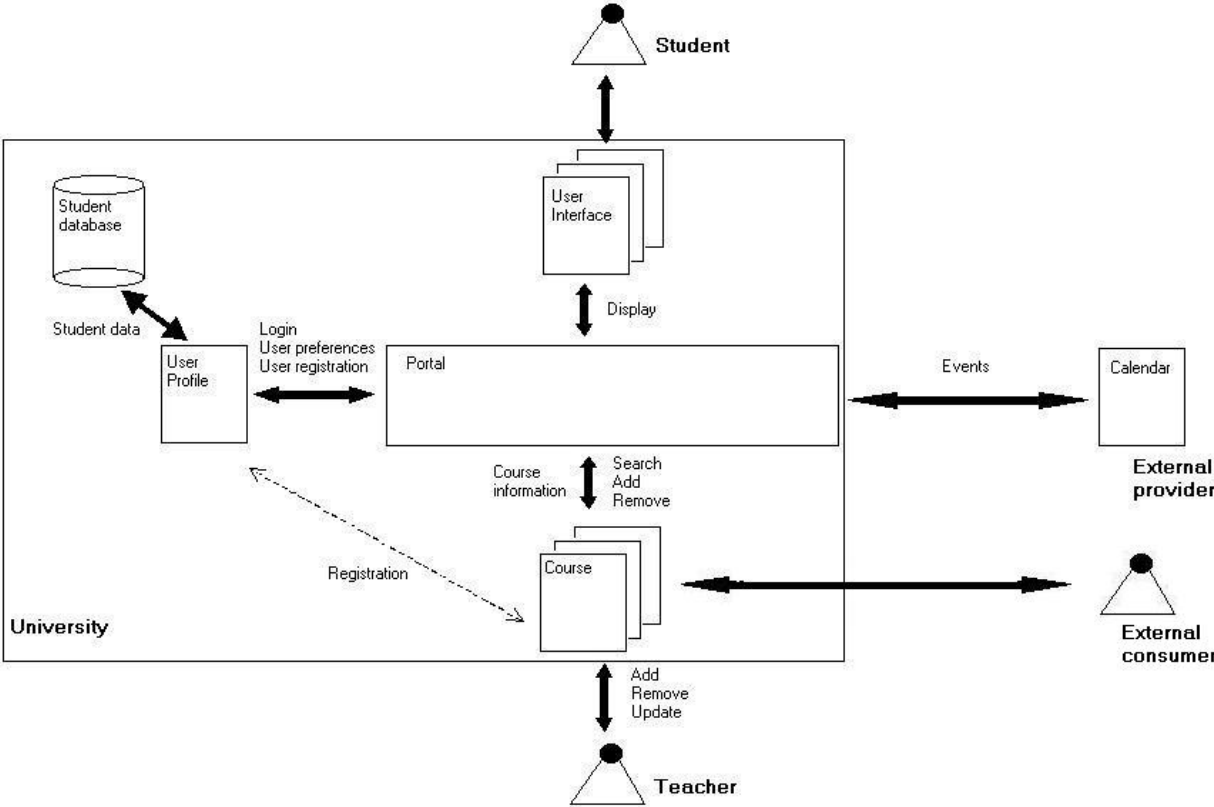


Figure 5. Architecture of the web services for the scenario

### 3.7 Development requirements

So far the description of the scenario has focused on the functional requirements of the university Web portal together with the design and interaction of the various actors. The scenario could probably be developed using any kind of Web service development environment to achieve the required functionality. However, also of interest are the underlying capabilities that the Web service development environment provides and what effect they have. These requirements are therefore directed at the Web service development environment and what it should provide to the developers of the scenario.

- **Security:** This is related to how well the Web service development environment provides security. Security covers aspects such as user authentication and authorization, and protection of data and resources. The user authentication is concerned with students accessing the portal, teachers accessing a course to modify information and external users accessing a course service. Authorization is related to what kinds of information users are allowed to access. Students who are registered for a course will have access to more information about the course than the unregistered ones, and external users are only allowed access to courses they have paid for.
- **Mobility:** The portal should be accessible through other devices than a PC, such as mobile phones or PDA's. It is also of interest to see what kind of information can be presented on the different devices and what kind of portal functionality that can be exposed.
- **Platform support:** This covers both the platforms on which the development environment can be used and the platforms for which the Web services can be developed. Having to change development platform in order to adopt a development environment is hardly any option for the university.
- **Component definitions:** This is related to how well the Web service is described. Normally the description would be the name of the methods one can call and the parameters these require. It would be a benefit if the development architecture provides some way of describing the operations of the Web service.
- **Legacy system integration:** The university has a database with all the student information and much of this information should be exposed through Web services. A development environment must be chosen that enables the legacy system of the university to be used without having to modify it.
- **Shared context:** This is related to the security aspect but particularly regards the integration of Web services. If the user logs on to a Web service that uses a Web service that also requires the user to log on, the best solution would be if the user did not have to log on one more time, but rather the Web service could use the same log on information the user already has provided.
- **Interoperability:** The course Web services should be able to interoperate with external Web services, so the interoperability that a Web service development provides is of importance here.
- **Ease of use:** If the development environment provides some sort of framework that eases and speeds up development time this would be beneficial for the developers.
- **Scalability:** The amount of students at the university or the number of courses offered always has the potential of expanding. It is therefore of importance that additional database server can be added without affecting the performance of the overall system.
- **Tool support:** The developers at the university would prefer to be able to use tools they are familiar with and not have to go through the process of learning too many new tools or a whole new programming language.

- **Billing:** The university are not offering their courses to external users for free so they would like the development environment to provide some kind of support for billing the external users, either as a one-time registration fee or a payment based on the frequency of use.

The importance of each of these requirements varies but they will all be considered when evaluating the architectures for Web service development later in this report.

### **3.8 Future possibilities**

As the evolution of Web Services continues, so can this scenario. It is just a matter of who provides the services and what these are.

Possibility number one is ordering the books for the course using a purchase Web Service provided by the university bookstore or any other manufacturer of the relevant books. If the book is available online it should also be possible to read this through the device the student is using to access the portal.

Much of the information that is available is often stored as Word-documents, PDF-files, Power Point presentations etc. Instead of the student having to download this to his/hers own computer and open it there (requiring the student to have the necessary software for reading the document), the student can read the information directly by having the portal use a Web Service that processes the appropriate document format and presents it, e.g. a Web Service for Adobe Acrobat Reader.

It should also be possible for the course Web Services to communicate with Web Services that are in some way related to the course, thereby providing additional and relevant information to the course.

### **3.9 Concluding remarks**

This chapter presented a scenario that will be used as a context for evaluation of different Web Service technologies and frameworks in the following chapters. The scenario is a web portal for students where they choose courses they are interested in and access information regarding these courses.

Both the portal and course services are provided by the university and are consumed by students and teachers respectively. In addition external actors like companies or other universities may have interest in only one or a few courses directly and not through a portal. The course web service acts as a framework for information regarding university courses, since all the teacher need to do is provide the information related to the course they teach as input parameters in the service methods. The teacher does not have to worry about details surrounding the presentation of data, nor does he/she have to spend time implementing the course as a Web service. Another benefit of having the university provide the course service to the teacher, instead of the other way around, is that it gives standardised access to the course service. This makes it easier to implement the portal since all methods are the same for all courses.

The university also provides services for login and user interface, and a calendar service from an external service provider is also used. The login is used for user authorization and saving user preferences, whereas the user interface service provides an interface to the portal based on the device used by the student.

Having the university provide most of the services will prove beneficial when it comes to interoperability issues. Within the university there should be no particular communication problems between services since the university provides all. Regarding the calendar service one can assume that there is more than one such service to choose from, and the university can pick the one that suits them best.

For external users interoperability could become a bigger problem particularly where the users would like to incorporate the course service into some of their own services.

Security is also an important issue, with regards to all parties. Students want to have their personalised portal protected from unauthorized access, the teachers want to be the only ones able to change information related to the course, and external users paying for the services want the billing to happen in a safe manner.

The most positive outcome of this approach is that students will have access to all information in one place, and that information is always up to date and not a replication from some other place.



# 4 Web Service Technologies

This chapter presents the standards and technologies involved in the development of Web Services and also presents some of the many frameworks and architectures that exist for creating Web Services.

## 4.1 Enabling standards and technologies

When building or using Web Services there are some key specifications and technologies one will encounter [3]. As shown in the introduction chapter, these specifications and technologies must address requirements regarding:

- A standard data representation
- A common extensible message format
- A common extensible service description language
- A way to discover service providers

The technologies are organised in a layered fashion when implementing the Web service architecture. Figure 6 shows the Web service technology stack:

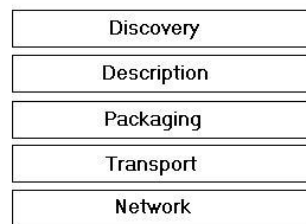


Figure 6. The Web service technology stack

In order to have a widespread acceptance of Web Services there has to be some commonly agreed (open) standards. The World Wide Web Consortium<sup>5</sup> (W3C), the Organization for the Advancement of Structured Information Standards<sup>6</sup> (OASIS) and the Internet Engineering Task Force<sup>7</sup> (IETF) all play an important part in the development and acceptance of such standards, and all the technologies used in the Web service technology stack are open standards, available to everyone.

The standards and technologies presented in this section can be used in a variety of ways to yield different frameworks and architectures for developing Web Services.

### 4.1.1 eXtensible Markup Language (XML)

For most Web Services the eXtensible Markup Language, XML, is the standard for representing data, using metadata, or tags. Any service that can process an XML file and output messages in an XML format can communicate with any other service or application that can do the same.

XML is a meta-markup language for text documents [6]. This means that it does not have a fixed set of tags and elements, like HTML. XML allows developers and writers to define the elements they need as they need them, hence the name extensible markup language.

---

<sup>5</sup> <http://www.w3.org>

<sup>6</sup> <http://www.oasis-open.org>

<sup>7</sup> <http://www.ietf.org>

Although XML is flexible in the creation of elements, it is strict in many other respects. It provides a grammar for XML documents that regulate the placement of tags, which elements are legal, how attributes are attached to elements etc. A document is said to be *well-formed* if it satisfies this grammar. The grammar is specific enough to allow development of XML-parsers that can read and understand well-formed XML documents.

The markup in an XML document describes the document's structure, how elements are associated with each other. It does not, however, say anything about how the document should be presented.

To enhance operability one may agree to use only a certain set of tags. These tags are called an XML application. This must not be confused with the term "software application"; rather it is an application of XML to a certain domain.

The markup permitted in a particular XML application can be documented in a document type definition (DTD). The DTD lists all legal markup and specifies how and where in the document the markup is allowed. Document instances can be compared to a certain DTD and those that match it are said to be *valid*, otherwise they are *invalid*.

The following is a small example of what XML looks like:

```
<xml version="1.0">
  <person ssn="121266 56789">
    <firstname>John</firstname>
    <lastname>Doe</lastname>
    <address>someplace</address>
    <zipcode>9999</zipcode>
    <city>somecity</city>
  </person>
```

The use of namespaces is important in XML and it has two purposes:

1. To distinguish between elements and attributes from different XML application that share the same name.
2. To group all related elements and attributes from a single XML application together so software can recognize them easily.

Namespaces are implemented by attaching a prefix to each element and attribute, where is prefix is mapped to a URI (Uniform Resource Identifier), thus removing any ambiguities from elements with the same name but with different meaning. The use of an element from a namespace looks like this `<prefix:elementname>`. Prefixes are bound to namespaces using the `xmlns:prefix` attribute set to the appropriate URI.

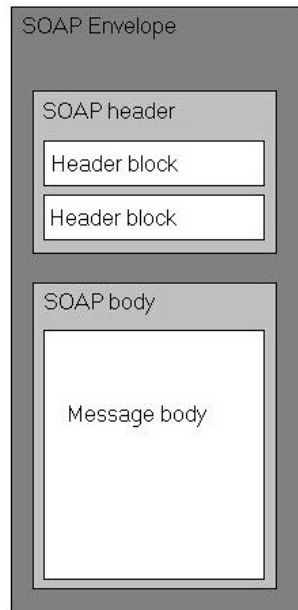
#### 4.1.2 Simple Object Access Protocol (SOAP)

The standard message format for communication with Web services is the Simple Object Access Protocol, SOAP. SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment [7]. It is an XML based protocol that consists of three parts:

- an envelope that defines a framework for describing what is in a message and how to process it
- a set of encoding rules for expressing instances of application-defined data types
- a convention for representing remote procedure calls (RPC) and responses

##### The SOAP Envelope

A SOAP message consists of an envelope, containing an optional header and a required body. Figure 7 illustrates this.



**Figure 7. The SOAP message structure.**

The header holds blocks of information related to how the message must be processed. If an envelope has a header element, there can only be one of it, and it must appear as the first child of the envelope, before the body. The header may contain any valid well-formed XML that has been namespace qualified. Each element placed within the header is called a header block. The purpose of the header block is to provide contextual information relevant to the processing of the SOAP message. This information can be routing and delivery settings, authentication or authorization assertions, and transaction context.

Every envelope must contain exactly one body, and the body may contain as many child nodes as required. The body contains the actual message that is to be delivered and processed. This can be any valid, well-formed and namespace qualified XML that does not contain any processing instructions or Document Type Definitions (DTD's).

When a SOAP message is sent from one application to another there is an implicit requirement that the recipient must understand how to process the message in the body. With header blocks it is different. A recipient may or may not understand header blocks but still be able to process the message in the body. If a sender of a message wants to add the requirement that the recipient must understand the header blocks, this is done by adding a *mustUnderstand="true"* attribute to the header block. If this attribute is set and the recipient does not understand the header blocks where it is present, then the recipient must reject the entire message.

The following example should suffice to illustrate the basics of a SOAP message:

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Header>
    <m:transaction xmlns:m="soap-transaction"
      s:mustUnderstand="true">
      <transactionID>1234<transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:purchaseOrder xmlns:n="urn:MyOrderingService">
      <from><person>David Jones</person>
    </n:purchaseOrder>
  </s:Body>
</s:Envelope>
```

```

        <dept>Sales</dept></from>
    <to><person>Jim Johnson</person>
        <dept>Office Appliances</dept>
    <order><quantity>30</quantity></to>
        <item>Fountain Pen</item></order>
    </n:purchaseOrder>
</s:Body>
</s:Envelope>

```

The XML syntax for expressing SOAP messages is based on the namespace found at <http://www.w3.org/2001/06/soap-envelope>. This namespace points to an XML schema that defines the structure of what a SOAP message looks like. The namespace is for SOAP version 1.2.

## Encoding Styles

An encoding style is a set of rules that define exactly how native applications and platform data types are to be encoded into a common XML syntax [8].

The *encodingStyle* attribute is used to define the encoding style for a particular set of XML elements. It can be placed anywhere in a document and applies to all subordinate children of the element in which it is placed. Encoding styles are how applications on different platforms share information, even though they may not have common data types or representations. The encoding style found in Section 5 of the SOAP specification [7] is one possible mechanism for providing this. The rules here tell in specific detail how basic application data types are to be mapped and encoded into XML format when placed within a SOAP envelope.

Encoding styles are optional and not always necessary at all, for example when the content to be placed in the envelope is an XML document.

Before presenting the Section 5 rules it is in its place to explain the notion of accessor and value, two important terms in the context of encoding rules. A value represents either a single data unit or a combination of data units, e.g. the weather temperature or a person's name. An accessor represents an element that contains or allows access to a value.

```
<firstname>David</firstname>
```

In the above statement `firstname` is an accessor and `David` is a value.

A compound value represents a combination of two or more accessor grouped together as child nodes of a single accessor:

```

<name>
    <firstname>David</firstname>
    <lastname>Jones</lastname>
</name>

```

There are two kinds of compound values – structs and arrays. A struct is a compound value where each accessor has a different name, like the example above. An array is a compound value where each accessor has the same name:

```

<people>
    <person name="David Jones" />
    <person name="Jim Johnson" />
</people>

```

An accessor may be either single-referenced or multi-referenced, using the special *id* and *href* attributes. A single-referenced accessor has no identity except as a child of its parent element. The *id* attribute is used to give an identity to the value of a multi-referenced accessor and those accessors wishing to refer to that value use the *href* attribute:

```

<person name='David Jones'>
  <address href=#address-1>
</person>
<address id='address-1'>
  <street>Some Street</street>
  <city>Fantasyville</city>
</address>

```

### SOAP encoding rules

SOAP defines three ways to express the data type of an accessor [8]:

1. Using the `xsi:type` on each accessor, thereby explicitly referencing the data type according to the XML Schema specification:

```

<person>
  <name xsi:type="xsd:string">David Jones</name>
</person>

```

2. By referencing an XML Schema document that defines exactly what datatype an element is within its definition:

```

<person xmlns="personschema.xsd">
  <name>David Jones</name>
</person>

```

The `personschema.xsd` here defines the `<name>` element as `type=xsd:string`.

3. By referencing some other kind of schema document that defines the datatype of a particular element within its definition:

```

<person xmlns="urn:some_namespace">
  <name>David Jones</name>
</person>

```

Here “`urn:some_namespace`” indicates a namespace where the value of the `<name>` elements is of type string.

The SOAP encoding style supports the data types defined in the “XML Schema data types” specification<sup>8</sup>. All data types used within the SOAP encoded blocks of XML must be from the XML Schema definition or be derived from the types within that definition.

### Interaction – the SOAP message exchange model

The third and final part of SOAP is concerned with how the messages are sent and received. The processing of a SOAP message involves taking it apart and doing something with the information it carries. The actual details of the implementation of the processing are left to the individual applications, but SOAP defines a general framework for such processing. This framework is primarily concerned with how applications exchange messages.

A very specific message exchange model is found in Section 2 of the SOAP specification [7]. The main idea behind this exchange model is that SOAP is a one-way transmission of an envelope from a sender to a receiver. The message may pass through an arbitrary number of intermediate processors that each does something to the message until it reaches the intended recipient. This is analogous to the pipeline processing model found among else in the Unix operating system.

A SOAP intermediary is a Web service specially constructed to add value or functionality to the transaction between a provider and a consumer. The set of intermediaries is called a message path, and each intermediary along the path is called an actor. The SOAP

---

<sup>8</sup> <http://www.w3.org/XML/Schema>

specification does not cover the creation of a message paths, this is handled in the various SOAP extensions, such as Microsoft's SOAP Routing Protocol. SOAP does however specify a means of expressing what parts of the SOAP message that are to be processed by which actor along the message path. This mechanism is known as targeting and only applies to header blocks.

By using the *actor* attribute a header block can be targeted for processing by a specific actor on the message path. The value of the *actor* attribute is set to the unique identifier of the intermediary being targeted. Intermediaries that do not match the identifier must ignore that header block.

As an illustration of this, consider the purchasing order example earlier in this section. Suppose that in order to verify that the person sending the order is who he says he is, the message is sent to a signature verification service. Here the header block containing the signature is extracted and checked, before a new header is added telling the recipient if the signature is valid.

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Header>
    <x:signature actor="uri:SignatureVerifier">
      ...
    </x:signature>
  <s:Body>
    <n:purchaseOrder xmlns:n="urn:MyOrderingService">
      ...
    </n:purchaseOrder>
  </s:Body>
</s:Envelope>
```

On the Web service technology stack in figure 6, SOAP fits on as a standardized packaging protocol layer on top of the transport and network layer. As a packaging protocol SOAP does not care what transport protocols are used (e.g. HTTP, FTP, SMTP, POP3, Jabber) to exchange messages, making SOAP very flexible in how and where it is used.

Due to the fact that it is so widely used on the Internet, HTTP is by far the most common transport protocol for exchanging SOAP messages. This is also reflected in the SOAP specification, which outlines specifically how SOAP messages map onto HTTP.

The following are examples of an HTTP request and an HTTP response containing a SOAP message.

HTTP request:

```
POST /WeatherService HTTP/1.1
Content-Type: text/xml
Content-Length: nnnn
SOAPAction: "urn:WeatherService#GetTemperature"
```

```
<s:Envelope xmlns:s "http://www.w3.org/2001/06/soap-envelope">
...
</s:Envelope>
```

HTTP response:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
```

```
<s:Envelope xmlns:s "http://www.w3.org/2001/06/soap-envelope">
...
</s:Envelope>
```

The `SOAPAction` HTTP header is defined in the SOAP specification and indicates the intent of the HTTP request. The value of the header is arbitrary; its function is to tell the HTTP server what the SOAP message wants to do. Servers can use the `SOAPAction` header to filter unwanted requests.

The key purpose of using SOAP is to hide the implementation of the actual service. SOAP is independent of any particular transport and implementation technology. A Web Service will have what is commonly referred to as a SOAP listener (corresponding to the listener in figure 4) that handles incoming SOAP messages. The listener also contains a proxy component, responsible for translating the incoming SOAP messages into code that can be passed on to the implementation of the Web Service. Likewise, if there is a response to the message then it must be translated into the corresponding message format and handed back to the listener for delivery to the requester.

### 4.1.3 Web Service Definition Language (WSDL)

The description layer on the Web service technology stack is supported by the Web Service Definition Language (WSDL). WSDL is a mechanism for describing precisely what the Web Service does. It is an XML based schema for describing the operational information related to the Web Service, such as the interface and the endpoints. WSDL defines XML grammar describing contracts between endpoints that are to exchange messages. It can be said to be analogous to the Interface Definition Language (IDL), used to describe components. W3C [9] defines WSDL as an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

A WSDL document defines a network service using the following elements:

- `types`: a container for data type definitions
- `messages`: an abstract definition of the data being communicated
- `operation`: an abstract description of an action supported by the service
- `port type`: an abstract set of operations supported by one or more endpoints
- `binding`: a concrete protocol and data format specification for a particular port type
- `port`: a single endpoint defined as a combination of a binding and a network address
- `service`: a collection of related endpoints

#### Using WSDL

The root element of every WSDL file is the `<definitions>` element. First of all you must provide various namespaces in the `<definitions>` element. The three external namespaces declarations that must be made are WSDL, SOAP, and XSD (XML Schema Definition).

Within the `<definitions>` element there are one or more `<portType>` elements, each of which is the set of operations one wants to expose. You should have one `<portType>` for each service. The `portType` must also have its `name` attribute set, giving a name to the service. The methods of your service are referred to by the `<operation>` elements, and the `name` attribute is used for naming each method.

Then you need to specify the parameters that will be sent and received. In WSDL these parameters are referred to as messages. The `<message>` element sets the `name` attribute to the name of the message. Then the `<part>` element is used for each part of the message (corresponding to parameters for function calls). For each `<part>` element you set the `name`

and the *type* attribute. The *type* attribute can be seen as the equivalent to data types in C++ or Java. WSDL uses a few data types that the XSD define (e.g. int, float, long, string) and lets you use them as they are or build complex data types based on these primitive ones. This is done using the `<types>` element, which encloses data type definitions. For maximum interoperability and platform neutrality, WSDL prefers to use XSD as the canonical type system and treats it as the intrinsic type system. However, WSDL also allows type systems to be added via *extensibility* elements (the use of an external namespace is called the extensibility element [10]). The extensibility element will then appear within the `<types>` element to identify the type definition system being used and provide an XML container for the type definitions.

When creating `<message>` elements you do not have to specify whether these messages are incoming parameters or return values. This is dealt with in the `<operation>` element within the `<portType>` element by adding `<input>` and `<output>` elements to the operations. The `<input>` element refers to a message by its name sees it as a parameter the user will provide when invoking the method. The `<output>` element refers to a message and treats it as the return value of the operation call.

So far this is just an abstract definition of operations and messages looking something like this for an imaginary “myService” Web Service:

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="MyService"
  targetNamespace="http://www.myservice.com/myservice-
interface"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.myservice.com/myservice"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <types>
    <xsd:schema targetNamespace="http://www.myservice.com/myservice"
      xmlns="http://www.w3.org/1999/XMLSchema/">
      <xsd:complexType name="myType">
        <xsd:element name="elementOne" type="xsd:String" />
        <xsd:element name="elementTwo" type="xsd:int" />
      </xsd:complexType>
    </xsd:schema>
  </types>

  <message name="Incoming">
    <part name="partOne" type="tns:myType">
  </message>
  <message name="ReturnMessage">
    <part name="returnValue" type="xsd:int">
  </message>

  <portType name="MyService_port">
    <operation name="myOperation">
      <input message="Incoming"/>
      <output message="ReturnMessage"/>
    </operation>
  </portType>
</definitions>
```

The next step is to bind this to the actual implementation. This is done by providing a reference to an external framework that defines how the WSDL user reaches the implementation of the service. This framework is called a *binding*.

In the W3C specification for WSDL [9] three binding extensions are presented. These are SOAP, HTTP GET/POST, and MIME. Using SOAP, WSDL will then specify a SOAP server that has access to the actual implementation, and after that SOAP handles the job of taking the user from the WSDL file to the implementation.

The <binding> element is used within the <definitions> element with a *name* that identifies the binding and a *type* that identifies the portType associated with the binding. The WSDL <binding> contains an extensibility element, the declaration of which external technologies will be used for binding purposes, using a namespace different from that of WSDL.

A <soap:binding> element has two attributes; *style* is an optional attribute for describing the nature of the operations within the binding, and *transport* specifies the lower-level transport service (like HTML) the binding will use. After the <soap:binding/> element comes the binding for the operations, with binding details for each individual operation. Using SOAP you have to provide the extensibility element <soap:operation/>. This element has a *soapAction* attribute that a SOAP client will use to make a SOAP request. The <soap:body/> element is used for each <output> and <input> element and provides the namespace with the name of the *service* on the SOAP server.

The bound WSDL file now looks like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="MyService"
  targetNamespace="http://www.myservice.com/myservice-
interface"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.myservice.com/myservice"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <types>
    <xsd:schema targetNamespace="http://www.myservice.com/myservice"
      xmlns="http://www.w3.org/1999/XMLSchema/">
      <xsd:complexType name="myType">
        <xsd:element name="elementOne" type="xsd:String"/>
        <xsd:element name="elementTwo" type="xsd:int" />
      </xsd:complexType>
    </xsd:schema>
  </types>

  <message name="Incoming">
    <part name="partOne" type="tns:myType">
  </message>
  <message name="ReturnMessage">
    <part name="returnValue" type="xsd:int">
  </message>

  <portType name="MyService_port">
    <operation name="myOperation">
      <input message="Incoming"/>
      <output message="ReturnMessage"/>
    </operation>
  </portType>

  <binding name="MyService_Binding" type="MyService_port">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="myOperation">
      <soap:operation soapAction="urn:MyService" />
      <input>
```

```

        <soap:body
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
namespace="urn:MyService" use="encoded" />
    </input>
    <output>
        <soap:body
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
namespace="urn:MyService" use="encoded" />
    </output>
</operation>
</binding>
</definitions>

```

In addition to the WSDL file describing the *interface* there is also need for a WSDL implementation file. This is a summary of the WSDL file and is used for publishing the Web Service. It contains the same <definitions> element except that it refers to a different namespace – the implementation file (instead of the interface). There also need to be an <import> element referring to the interface file, and finally there is a <service> element with the logical name of the service. Within this tag is also a <port> element that refers to the SOAP binding created in the interface file.

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

While WSDL provides the consumers with all the information necessary to interact with the Web service, the consumers still needs to be able to locate the services they need. This is the job of the discovery layer on the Web service technology stack.

UDDI<sup>9</sup> is a cross industry initiative to create a standard for web service description and discovery together with a registry facility that simplifies the publishing and discovery processes. It can be seen as the “yellow pages” of Web Services. As with traditional yellow pages one can search for a company that offers the services one needs, read about the service offered and contact someone for more information.

The core component of UDDI is the UDDI business registration, an XML file used to describe a business entity and its Web services. The XML file has three parts, the “white pages” describing the company offering the service, the “yellow pages” include industrial categories based on classification standards, and the “green pages” describe the interface to the service detailed enough for someone to know how to access the Web Service.

Using the UDDI discovery services businesses register information about the Web services they expose for use by others. This information can be added to the UDDI registries via a Web site or by using tools. The UDDI registry is a logically centralised, physically distributed service, consisting of multiple root nodes that replicate data on a regular basis [9]. Once a business register with a single instance of the business registry service, the information is automatically shared with the other UDDI root nodes, and is now freely available to be discovered by anyone looking for services exposed by a given business.

##### Technical overview

The UDDI specifications consist of an XML scheme for SOAP messages, and a description of the UDDI API specification.

The core information model used by UDDI registries is defined in an XML schema. This schema defines four main types of information that provide the kind of information a technical person would need in order to use someone’s Web services. The four types are:

- Business information
- Service information

---

<sup>9</sup> <http://www.uddi.org>

- Binding information
- Information about specifications for services

### **Business information**

The main elements for supporting publishing and discovering information about a business, the UDDI business registration, are contained in a structure called *businessEntity*. This structure serves as the top-level information manager for all the information about a particular set of information related to a business unit. The overall *businessEntity* information includes support for “yellow pages” categorization so that searches can be performed to find businesses within a specific trade or geographical region.

### **Service information**

The “green pages” data, technical and business descriptions, are contained within substructures of the *businessEntity* information. Two such structures are defined: *businessService* and *bindingTemplate*.

The *businessService* structure is a descriptive container that is used to group a series of related Web services based on either a business process or a category of service, e.g. shipping or purchasing services. Within each *businessService* live one or more technical Web service descriptions. These contain information (like the address of the Web service) relevant for application programmers that need to connect to and communicate with a remote Web service.

### **Binding information**

The information required to actually invoke a Web service is described in the *bindingTemplate* information element. A *bindingTemplate* element contains a special element that is a list of references to information about specifications. These specifications are concerned with issues such as what message format the Web service supports, what protocols it uses, what responses to expect etc.

The reference to the specification is called the *tModel* and is used to access the specification. The *tModel* is metadata about a specification, including its name, publishing organization and the URL pointers to the actual specifications themselves. The *tModel* does not contain the actual specification. Instead, UDDI defines a framework for utilizing URL’s and web servers, giving individual organisations the possibility to maintain centralised specifications. This again makes it possible to locate services whether or not they refer to certain specifications.

### **Programmer’s API**

As mentioned earlier the UDDI specifications include definitions for Web service interfaces that allow programmable access to the UDDI registry information – the programmer’s API. The API is divided into two logical parts: the *Inquiry* API and the *Publishers’* API.

The Inquiry API is further divisible into two parts – one used for creating programs that lets you search and browse information found in a UDDI registry, and the other used if it should occur that a Web service invocation fails.

The Inquiry API consists of two types of call that quickly lets a program discover candidate business Web services and specifications, and dig deeper into more specific details of the information initially provided by these calls. The *find\_xx* APIs gives the caller a wide range of registration data, based on different search criteria. The *get\_xx* APIs can be used if the actual keys of specific data are known, and let the caller retrieve copies of a particular structure (e.g. *businessEntity*, *businessService*, *bindingTemplate*, *tModel*).

The Publishers’ API consists of four *save\_xx* functions and four *delete\_xx* functions, one for each of the four key UDDI data structures (*businessEntity*, *businessService*, *bindingTemplate*,

*tModel*). An individual party can register any number of *businessEntity* or *tModel* information sets, or change previously published information, as long as the party is authorized. Only the same individual who initially created the structure can change it.

### **The UDDI invocation model**

Each individual advertised Web service is described in a *bindingTemplate* structure, and the invocation of a Web service is usually performed based on cached *bindingTemplate* data. The following can be seen as a recipe of on how to write a program that accesses a Web service:

1. Using the UDDI business registry the programmer locates the *businessEntity* for the appropriate business advertising the required Web service.
2. The programmer can investigate into more detail about a *businessService* or request a full *businessEntity* structure. This structure holds all information about the advertised Web Service. The programmer selects a particular *bindingTemplate* and saves it for later use.
3. The program is prepared based on knowledge of the specifications for the Web service. This information can be obtained using the *tModel* key information from the *bindingTemplate* for the service.
4. At runtime the program invokes the Web service using the cached *bindingTemplate* information.

The calls to the remote service will execute successfully assuming the calling program and the remote Web service each implement the required interface conventions, as defined in the specification referenced to by the *tModel*.

### **Failure and recovery**

A main concern both for those publishing and those seeking Web services is the ability to detect and manage communication problems and other failures. UDDI addresses these worries in the cached *bindingTemplate* convention mentioned above. If a failure occurs, the cached information is refreshed with a copy of the current version in the UDDI Web registry. The approach is as follows:

1. The program is prepared for accessing a Web service by caching the *bindingTemplate* data for use at run-time.
2. When the remote Web service is to be called the cached *bindingTemplate* data is used.
3. If the call fails the *bindingKey* value and the *get\_bindingTemplate* API call is used to obtain a fresh copy of the *bindingTemplate* for the Web service.
4. The new information is compared with the old and if it is different the call is retried and the cached *bindingTemplate* data is replaced with the new data.

This approach is called retry on failure and is more efficient than getting a fresh copy of the *bindingTemplate* data prior to each call.

More details about the UDDI schema and the UDDI Programmer's API specification are available at the [uddi.org](http://uddi.org) Web site.

## **4.2 Web Service Frameworks and Architectures**

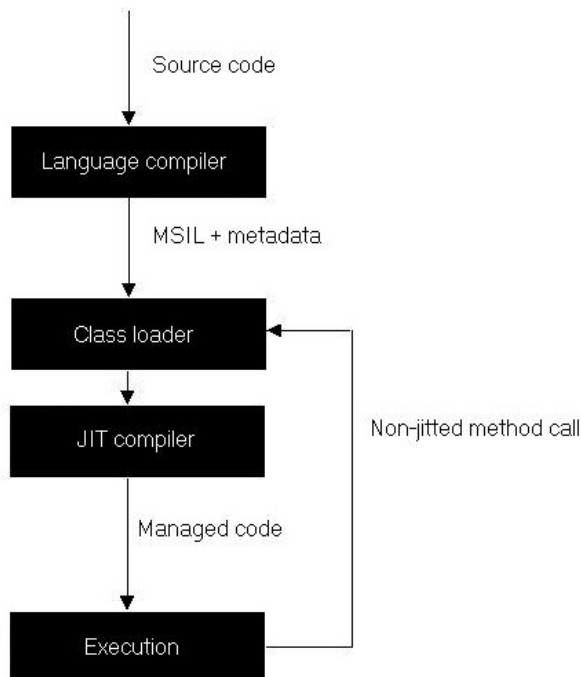
This section presents some frameworks and architectures that can be used to develop Web services. The ones that will be presented are the .NET Framework from Microsoft, the Java 2 Enterprise Edition, and the Sun Open Net Environment, both from Sun Microsystems.

## 4.2.1 Microsoft .Net Framework (.NET)

The Microsoft .Net Framework is a platform for building integrated service-oriented applications for the Internet [19]. This does not just include Web services but also console applications, Windows GUI application, ASP.NET applications and Windows services. The foundation of .NET is the common language runtime. It is first and foremost responsible for managing code from .NET programs at execution time. This code management includes mechanisms such as thread management, memory management and strict type safety. Code targeted for the common language runtime is known as managed code. This means that managed code require the common language runtime to execute, whereas code that does not require the common language runtime is known as unmanaged code.

The common language runtime is independent of any particular programming language as long as the language compiler can produce managed code. When source code is compiled the code generated by the compiler is a Microsoft Intermediate Language (MSIL). This language is higher level than the machine code normally produced by compilers and is not tied to any specific CPU. This means that any executable file containing MSIL can run on any CPU platform provided that the operating system running on that CPU platform host the .NET common language runtime.

The MSIL instructions cannot be executed directly, so the common language runtime engine must compile the MSIL instructions into native CPU instructions. The MSIL code is not translated all at once but rather as functions are being called, so-called just-in-time compilation or jitting. Figure 8 shows the .NET execution model.



**Figure 8. The .NET execution model**

When source code is compiled the compiler emits MSIL plus metadata. The MSIL is further translated by the JIT compiler to native code, which executes on the target platform. There are different variants of the JIT compiler. The default JITter works as described earlier by translating routines as they are called. The economy JITter, EconoJit is a variant targeted for devices with smaller memory, and allows developers to specify how much memory space is used for jitting and decide what policy is used to remove a jitted routine to make room for a new one. The last option is PreJit, which translates the MSIL once for the whole application.

The common language runtime implements the Common Type System (CTS), which is the formal specification of how the type system is implemented by the common language runtime. The CTS defines a standard set of objects (called types) and rules for creating new types. In addition it contains rules for type inheritance, virtual functions, object lifetime etc. Also on top of the common language runtime lie the Common Language Specification (CLS). The CLS is used to inform compiler vendors of the minimum set of features their compilers must support to target the common language runtime.

The .NET component model is based on the notion of assemblies, where an assembly is a collection of resources and types deployed as a unit (e.g. like an .exe file). Assemblies are self-describing components so no other files are required to use them. The assemblies are described using metadata, information that supplements the executable code. The metadata is contained within the assembly; no additional files are needed. Metadata for the assembly is emitted at compilation and is called the assembly manifest. The manifest includes information such as a list of types and resources available outside the assembly and also the dependencies to other assemblies.

The other major part of the .NET framework is the .NET framework class libraries. The class libraries are a comprehensive object-oriented collection of reusable types [17] that can be used in the development of the various .NET applications. The reusable types are tightly integrated with the common language runtime. Being object-oriented the developers can derive functionality from the types in their own managed code.

The class libraries can be split into two categories: One base set of class libraries providing output/input, string and numerical classes etc. In addition these base framework libraries provide classes to access operating system services such as graphics, networking, threading and cryptography.

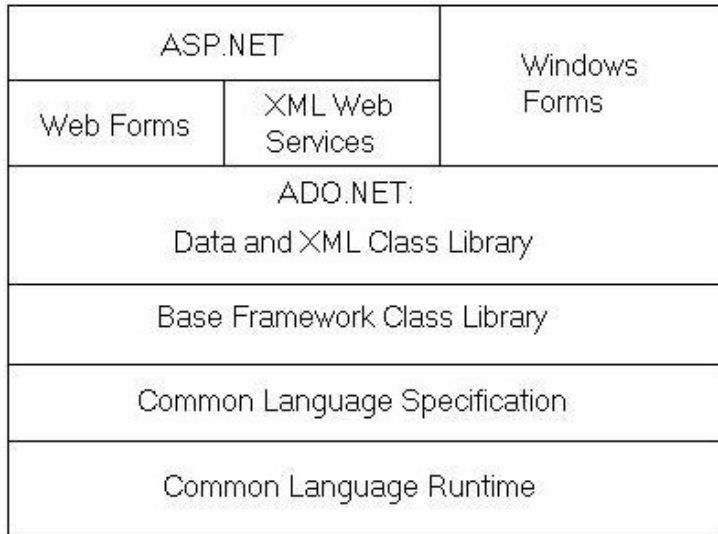
The other part of the .NET framework class libraries is the data and XML classes. This is provided with the ActiveX Data Objects .NET (ADO.NET) class library. This library support persistent data management and include SQL classes for manipulating (query, update etc.) persistent data through a standard SQL interface. ADO.NET also include XML classes that enable XML data manipulation and XML searching and transformation. The .NET framework provides a specific transformation component that supports the W3C XSL Transformations specification<sup>10</sup>.

On top of the .NET framework sits two application models: the Windows application model and the Web application model. Developers wanting to write client applications for Windows use the Windows Forms application model. The Windows Form classes are designed to be used for GUI development and allows the developer to create command windows, buttons menus, toolbars etc, in addition to support for ActiveX controls.

The application model for Web application development is the Active Server Pages .NET (ASP.NET) model, which can be used to publish both XML Web services and Web Forms. ASP.NET comes with a set of Web Forms controls that are used to generate the user interface, typically in the form of HTML. Web Forms also has support for developing interactive Web pages. The ASP.NET Web services model uses a set of classes, provided by the .NET framework, that conform to the Web service standards, such as SOAP, XML and WSDL. Figure 9 shows how the .NET Framework is made up:

---

<sup>10</sup> <http://www.w3.org/TR/xslt>



**Figure 9. The Microsoft .NET framework**

The figure shows that it all starts with the CLR at the bottom. The CLS governs the use of the CLR by describing the features needed for a compiler to target the CLR. Next follow the .NET framework classes, which is split in two: The base class library providing system-level functionality such as I/O-operations and numerical classes, and database manipulation classes provided by ADO .NET. On top of these is the various application programming models that provide higher-level components and services for developing both Windows and Web applications, as described above.

#### **4.2.2 Java 2 Enterprise Edition (J2EE)**

Sun's Java 2 Enterprise Edition is a Java platform designed for the purpose of reducing the cost and complexity in the development of multi-tier services for enterprises. J2EE has traditionally been an architecture for building server-side deployments in the Java programming language [26], but has recently been extended to include support for building XML-based Web services as well.

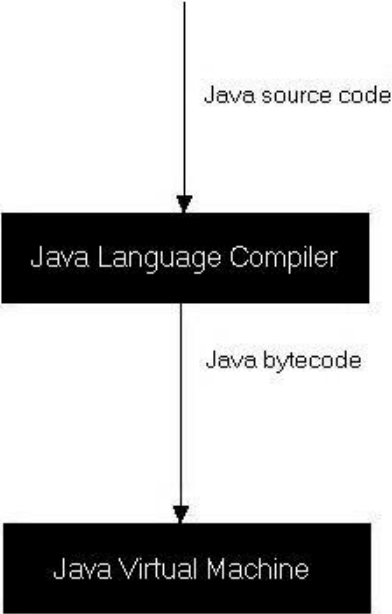
The J2EE defines a standard architecture consisting of four elements [24]:

- J2EE Platform Specification
- J2EE Application Programming Model
- J2EE Compatibility Test Suite
- J2EE Reference Implementation

The J2EE Platform Specification defines the specific Java APIs that are needed in order to develop enterprise multi-tier services.

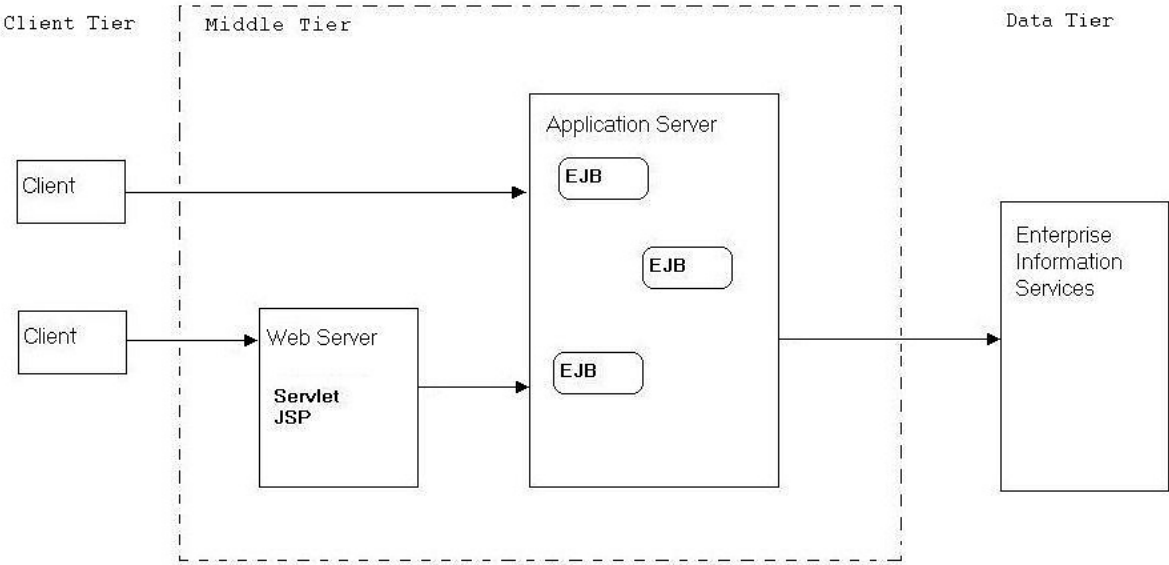
The J2EE Application Programming Model provides an architecture that supports the developer in implementing services as multi-tier applications that can run on the J2EE platform. The model partitions the work required to create multi-tier services into two parts: the business and presentation logic that must be implemented by the developer, and the standard system services provided by the J2EE platform. The J2EE Application Programming Model builds on the Java programming language and the Java Virtual Machine (JVM). The Java language consists of numerous classes and interfaces that are used to write Java applications. A Java language compiler compiles the programming language into a platform-independent executable called Java bytecode. The Java bytecode is dynamically

compiled/interpreted by the JVM into platform-specific instructions. This process is shown in Figure 10.



**Figure 10. The basics of the Java programming language.**

The application model also includes the JavaBeans component model. The Enterprise JavaBeans (EJB) technology defines a model for developing and deploying reusable Java server components, and it defines a standard programming interface for Java application servers. All middle-tier business functions are implemented as EJB components, whereas the middle-tier functionality is presented to clients using the Java Server Pages (JSP) and servlets technology. These components are not standalone applications, they run within another application environment called a container. The container provides an application context for one or more components and provides management and control services for the components. This is illustrated in Figure 11 with the J2EE architecture.



**Figure 11. The J2EE architecture.**

The figure shows the basics of the J2EE multi-tier environment, where the most central part is the middle tier. It provides the runtime environment for components that interact on both the client side and the data side.

Vendors use the J2EE Compatibility Test Suite to verify that their J2EE platform implementation complies with the J2EE Platform specification.

The J2EE Reference Implementation is an implementation of the J2EE platform specification. This is used to provide an operational definition of the J2EE platform, which can be used to demonstrate the capabilities of the platform or test J2EE applications.

The secret of building XML Web services using J2EE is a set of Java APIs, the JAX\* APIs. The APIs are as follows:

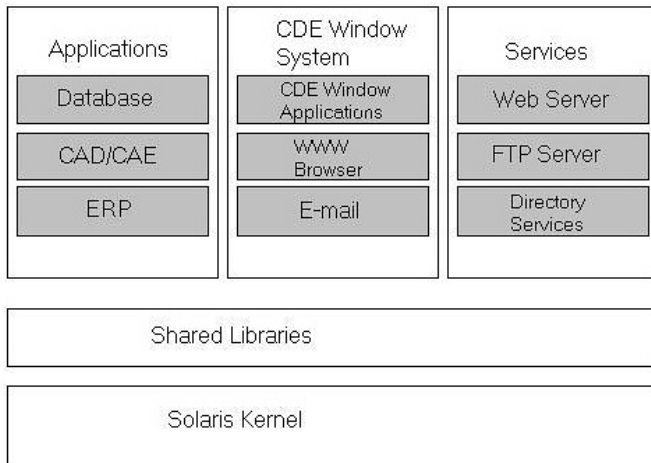
- Java API for XML Parsing (JAXP): This is a native Java interface to the industry standard XML parsing APIs SAX (Simple API for XML) and DOM (Document Object Model), in addition to a pluggable interface to an XSLT (XML Stylesheet Language Transformation engine), which is a mechanism for converting an XML document from one schema to another. These standards form the basics for parsing and processing XML documents.
- Java API for XML Binding (JAXB): JAXB is a specification for transforming an XML document into a Java object and vice versa. Back-end EJB layers can then handle the Java objects.
- Java API for XML Messaging (JAXM): This is a specification for interacting with XML messaging standards such as SOAP and ebXML. The API handles all the envelope header content, such as routing information, separate from the body of the envelope, thereby letting the developers focus on interacting with the message content and not the administrative issues.
- Java API for XML based RPC (JAX/RPC): JAX/RPC is used for sending and receiving method calls using XML-based protocols such as SOAP. The intention is relieve the developers of having to interact directly with the XML representation of the method call. Using JAX/RPC a servlet can receive for example a SOAP request, perform the business processing and return the result.
- Java API for XML Registries (JAXR): JAXR can be used to access registries such as UDDI.

These APIs constitute an interface layer responsible for translating incoming XML data to a format suitable for processing by the business service and translating the results back to an XML format that can be sent to the requester.

### **4.2.3 Sun Open Net Environment (Sun ONE)**

Sun Open Net Environment is an open framework that supports the development of Services on Demand [22]. Sun uses the term “Services on Demand” to encompass local applications, client/server applications, Web applications and Web services. The Sun ONE platform is targeted mainly for enterprise usage and the architecture is created to reflect the processes of modern day businesses.

The Solaris 8 Operating Environment, designed to support multi-user, multi-platform network computing, is the foundation of Sun ONE. The Solaris Operating Environment begins with a small kernel that only contains the core features required by applications and also has support for the addition of new devices, software libraries and even file systems. The other parts of the Solaris Operating Environment are the shared libraries and the CDE windowing system, as shown in Figure 12.

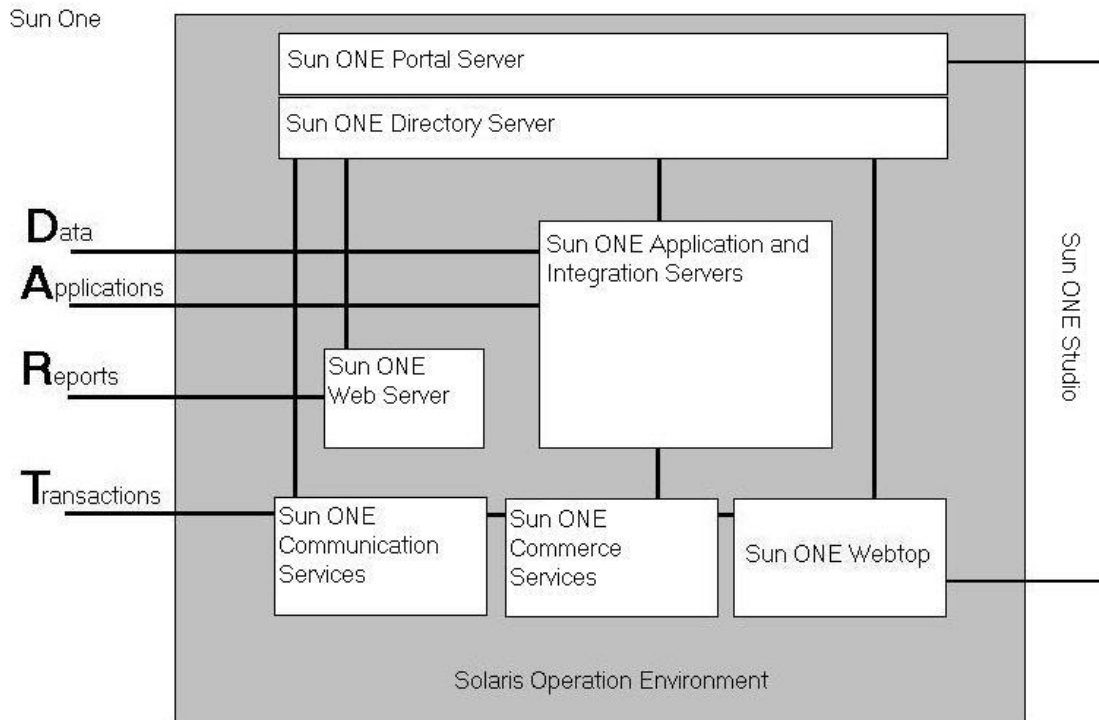


**Figure 12. Solaris Operating Environment.**

There is a clear separation of the shared libraries and the kernel, keeping the instruction-set and platform specific code isolated and easily changed. The kernel consists of modules of processor-specific code that handle platform-specific features such as access to I/O devices. Having separated the shared applications from the kernel an error in the shared library can crash applications using the library but never the Solaris environment itself. The Solaris environment supports multiple shared library version, thereby removing the risk of application failures due to incompatible library versions. For more detailed information on the Solaris Operating Environment refer to [27].

The basic elements of the Sun ONE architecture are: data, applications, reports and transactions (DART). The DART model is specifically engineered to support the information and processes of modern enterprises. The Data part is concerned with the presentation and aggregation of data that is meaningful to the organisation. This also includes user data and personalisation of content using portals. The Application part handles the ability to run already existing databases and application on the Sun ONE platform. Reports are concerned with keeping track and providing feedback of use and value of the organisation's service. The Transactions encompass useful operations on the information assets, such as buying, selling, billing or product trade, both inside and outside the enterprise.

The Sun ONE infrastructure software (formerly known as the iPlanet product line) is very central to the Sun ONE architecture as it provides servers and services that fill different roles. Figure 13 shows how the different products relate to constitute the Sun ONE framework.



**Figure 13. The Sun Open Net Environment Platform**

The Sun ONE Directory Server contains the enterprise's information assets, and sits between the community (the different kinds of users) and the data stores. The software suite of the Sun ONE Directory Server provides functionality for collecting, distributing, managing and protecting the information assets of the enterprise.

The idea of giving each user centralised and controlled access to services via a portal is a central part of the Sun ONE platform. A portal provides an aggregation of information and personalised access to this through user authentication and group memberships. On Sun ONE this is achieved using the Sun ONE Portal Server.

The Sun ONE Application Server enables access to business applications and databases and also provides a scalable platform for Java technology-based business logic. The scalability is horizontal, thus managing multiple Java runtime environments and multiple CPUs as one. The Sun ONE Integration Server enhances the Sun ONE Application Server by adding the possibility of aggregating business workflows across applications.

Reporting is achieved using the Sun ONE Web Server. It provides a platform for assembling and displaying information related to public Web sites, and in addition has built-in functionality such as load balancing.

There are two kinds of Web service transactions [22] – person-to-person and machine-to-machine. The Sun ONE Communication Services, which includes Sun ONE Messaging Server (Web mail interface) and Sun ONE Calendar Server (personal and business calendars), handles the person-to-person transactions. The Sun ONE Commerce Service, providing commerce functionality such as marketplaces and billing, handles machine-to-machine transactions.

Sun ONE Studio<sup>11</sup> (formerly known as Forte Tools) offers an integrated development environment for the Java, C, C++ and FORTRAN languages. One of its features is the automatic loading of just the capabilities the developer needs, so that different developers are able to use the same tool even though they have different preferences.

<sup>11</sup> [http://www.sun.com/software/product\\_family/forte.html](http://www.sun.com/software/product_family/forte.html)



# 5 Evaluation of technologies

This chapter contains an evaluation of the frameworks and architectures presented in the previous chapter. The evaluation is also based on the scenario in chapter 3 and how well suited the different technologies are for developing a solution that fits the scenario. Based on the descriptions of the architectures from the previous it is hard to make direct comparisons. Rather they must be evaluated with regards to some evaluation criteria. Before going in to the evaluation of the architectures, the criteria upon which the evaluation is based are presented.

## 5.1 Evaluation criteria

There are many factors to be considered in an evaluation of the architectures for Web services. Some are more important than others and the importance of each factor is also relative to the intended use of the service. This evaluation has as a goal to distinguish the various architectures and show both their benefits and drawbacks. Therefore as many as factors as possible should be chosen, without biasing towards any form of ranking of the factors. Most of the criteria are directly related to the requirements from section 3.7 since parts of the evaluation will be based on the scenario.

The following factors have been chosen for this evaluation:

### Security

For many this is one of the most important criteria, particularly in cases where personalised access is required. Issues related to this criterion are how one can protect the service from unauthorised access, how to recognise that the users are who they claim to be, encryption when sending privacy details etc. Another important thing to consider is related to how to prevent the Web service from doing anything it is not supposed to do, like accessing, changing or removing anything on the users' computers.

In the scenario the main areas for security are the user authentication for the portal, the teacher being the only one allowed to maintain the course service and restricting access to services by unauthorised users (students not registered or any non-paying external users).

### Mobility

This criterion concerns what devices the architecture can produce Web services for. Included in this are the constraints, if any, on what can be achieved on the different devices.

For the scenario the things to consider are what information can be accessed using different devices and how should the different user interface services be developed.

### Supported platforms

This criterion covers both the platforms on which the architecture can be employed and for what platforms Web services can be developed with that architecture. If there are any requirements to the platforms these are also brought into consideration here.

### Supported protocols

This regards the transport protocols that are supported by the architecture (HTTP, Jabber, SMTP etc.).

### **Component definitions**

WSDL describes a Web service with respect to the operations it provides and what any eventual parameters and responses of these are. Describing exactly what an operation does is not the job of WSDL so the architecture should support this.

### **Legacy system integration**

Companies who wish to expose some of their business functionality through Web services should be able to do so without having to change anything in their existing legacy systems. For example, the contents of a database should be presented without having to change the structure of the database. It is therefore of interest to see how this can be achieved in the different Web service architectures.

### **Shared context**

The idea behind shared context is that users of Web services only needs to supply information such as username, password, credit-card information etc. only once, and that this information is available to all Web services chosen by the same user. This is particularly interesting in situations where a Web service uses other Web services that also require a user authentication.

### **Interoperability**

The Web service concept promises increased system interoperability across languages and platforms. It is therefore interesting to look at how well the architectures provide this interoperability and what technologies are used to achieve this. The W3C standards are important in this context. Any restrictions on what degree of interoperability can be achieved must also be considered.

### **Ease of use**

Even though this often is a subjective measurement some indication can be provided on the ease of use, based on what is expected from the developer applying the specific architecture. Any pre-defined functionality or framework (similar to for example Microsoft Foundation Classes, MFC) that the developer can build on is also part of this.

It would for example be easier (and faster) for the developers at the university if the architecture provides some framework for developing the user authentication service.

### **Scalability**

The ability to scale up the Web service is important because one can never tell exactly who and how many will use the service. The scalability is related to how performance is affected by an increase in users and the possible impact of adding more resources.

If for example the university allows for the entrance of a larger number of students and expand the number of courses, they would perhaps need to add a new database server to handle the increased amount of information. This should preferably be done without affecting the response time of the portal service.

### **Deployment**

This is related to issues surrounding the publishing of the Web Service and how the architecture supports this. One such issue could for example be if the architecture has some sort of connection to UDDI.

### **Modelling tools**

A variety of modelling tools for software development exist already, like UML and OMT. The question is whether some of these relate to the architecture or if the architecture perhaps provides some form of modelling tool in itself.

If the developers at the university are accustomed with using UML for software modelling then it is their interest to be able to apply this to development using the selected architecture.

### **Tool support**

This criterion applies to what tools that are supported by the architecture, like development tools, servers, databases and libraries.

Knowing what tools are supported by the architecture can have a great impact on the choice of architecture in the scenario. The university does not want to have to spend a lot of money on new technologies; they would rather prefer to use what they already have without any additional cost.

### **Billing**

If one has plans of creating Web services that requires payment to be accessed, there needs to be some way of billing the user of the service. If there is any such support within the architecture the transaction method used is also of interest.

External consumers of the course Web service in the scenario must be billed for their use, either as a one-time registration fee or a payment based on the frequency of use.

### **Availability and price**

When the time has come to invest in an architecture it is useful to know where one can acquire it and at what price. There are many places to find software these days. If it is not available at any retailer it can be ordered from the store or from the Internet, and very often it can be downloaded directly from the Internet.

Associated with the price are most often one or more licences. So one cannot just focus on the price, one must also look at what value for money one gets. This includes the price of future related releases and upgrades to the product.

## **5.2 Evaluation**

Having presented the criteria for evaluation, it is now time to move on to the actual evaluation. The evaluation will be performed one criterion at a time, discussing how each Web service platform satisfies that particular criterion.

### **5.2.1 Security**

Security spans a wide field, from the authentication of users to ensuring that all code executes harmlessly without any chance of corrupting the executing environment. To get a better overview of the security concepts provided by the different architectures this section is structured according to the following security concepts:

- **Authentication and Authorization:** Authentication is concerned with identifying users and confirming that the users are who they claim to be, and how the architectures support this. Authorization relates to how the developer may specify what functionality the various users will experience and what kinds of resources they may access.
- **Encryption/Secure transfers:** This is a means of ensuring the users that sensitive data (passwords, credit card numbers etc.) communicated across applications cannot be read and understood by any intruders.

- Code security: This prevents malicious code from performing unexpected and unwanted actions, such as illegal operations or taking up resources. It is particularly in cases where code from unknown or semi-trusted sources are involved that one wants this kind of security.

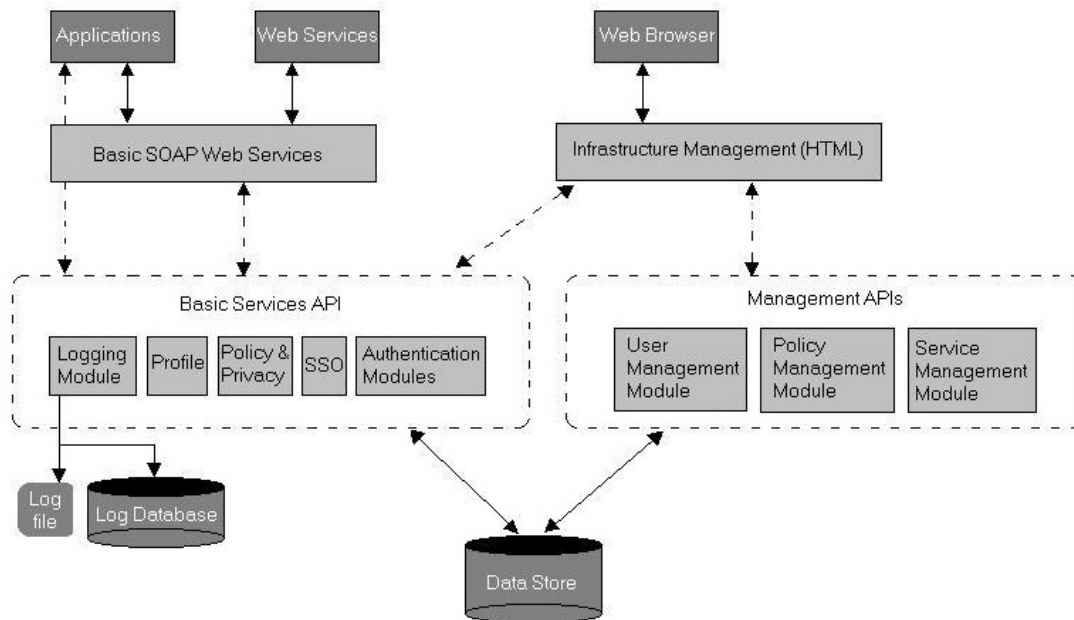
But before these issues are discussed a brief introduction of the overall security principles of each architecture:

### J2EE

The J2EE architecture does not support any particular security policy; this is rather the responsibility of the individual implementations. Still, J2EE intends to facilitate the implementation of these policies by specifying some requirements on security issues.

### Sun ONE

In Sun ONE the Identity and Policy Services provide the infrastructure for managing user identity, roles, and security. They also provide fundamental basic services such as Web service discovery using UDDI, authentication, single sign-on (SSO), and policy evaluation. The overall architecture of the Identity and Policy Services is shown in Figure 14.



**Figure 14. The Identity and Policy architecture.**

The architecture can broadly be divided into three categories:

- Infrastructure management components
- Core components
- Web services.

The infrastructure management modules provide interfaces to manage users, policy and services via a Web browser.

The User Management component creates, modifies and deletes user identities. The Policy Management component provides interfaces to create, modify and delete policy rules that define the service and access privileges for users. The Service Management component provides interfaces to manage Web services using UDDI and interfaces to manage service configuration parameters.

The Core components provide two kinds of interfaces:

- APIs that can be accessed directly by applications written in the Java programming language.
- Web services that can be accessed via SOAP over HTTP.

The following Core components are provided:

- Authentication
- Single sign-on (SSO)
- User personalisation or preferences
- Policy Evaluation and Privacy
- Security using PKI and Kerberos
- Logging

The Identity and Policy components are provided by Sun ONE Directory Server Access Management Edition and Integration edition.

### **Microsoft .NET**

The security architecture of the .NET framework is composed of the following elements [28]:

- Evidence-based security
- Code access security
- The verification process
- Role-based security
- Cryptography
- Application Domains

Most of these security elements are realised by the managed code architecture together with the CLR, and also by many of reusable classes in the base framework class library.

#### **5.2.1.1 Authentication and Authorization**

##### **J2EE**

With respect to user authentication it is required that all J2EE Web server must maintain a login session for each web user, and it must be possible for a login session to span more than one application. The login session ensures that a user only has to log in once.

The authentication of users must done using one of three required login mechanisms:

- **HTTP Basic Authentication:** This is the authentication mechanism supported by the HTTP protocol, and is based on a username and password. The authentication request is from a web server to a web client, the web client obtains the username and password from the user, and sends it to the web server where the authentication is performed. This is not a very secure authentication mechanism but it is included because it is very widely used in form based applications.
- **SSL Mutual Authentication:** The J2EE specification requires that SSL 3.0 and the means to perform mutual (client and server) certificate based authentication is supported.
- **Form Based Login:** The appearance of the login screen cannot be edited using the Web browser's built-in authentication mechanism. The Form Based Login enables the developer to control the design of the login screen. The authentication procedure is much like the one for HTTP Basic Authentication, and is therefore not very secure since the user password is sent as plain text and the authentication server is not authenticated.

The authorization model of the J2EE architecture consists of these requirements:

- Code authorization: A J2EE product may restrict the use of certain J2SE (Java 2 Standard Edition) classes and methods to secure proper operation of the system. The minimum set of permissions that a J2EE product must grant a J2EE application is listed in the J2EE specification [36] in section 6.2.
- Caller authorization: A J2EE product must enforce the access control rules specified at deployment time and more fully described in the EJB and servlet specifications. EJB and servlets use authorization constraints, which are a set of security roles. All resources that are associated with an authorization constraint require that a user belong to at least one of the listed roles to be allowed access to the resource.
- Propagated caller identities: A J2EE product must be configured so that in a chain of calls to enterprise beans all calls are associated with the same user identity.
- Run As identities: J2EE products must support the Run As capability that allows the developer to specify an identity under which an enterprise bean or web component must run.

The deployment requirements of the J2EE specification say that all J2EE products must implement the access control semantics described in the EJB, JSP and servlet specifications.

Resources within an enterprise often follow their own security policies, different from those of the application. It is therefore required that the J2EE product provides a means to authenticate in the security policy domain of the resource. The J2EE implementation is therefore required to support at least the following:

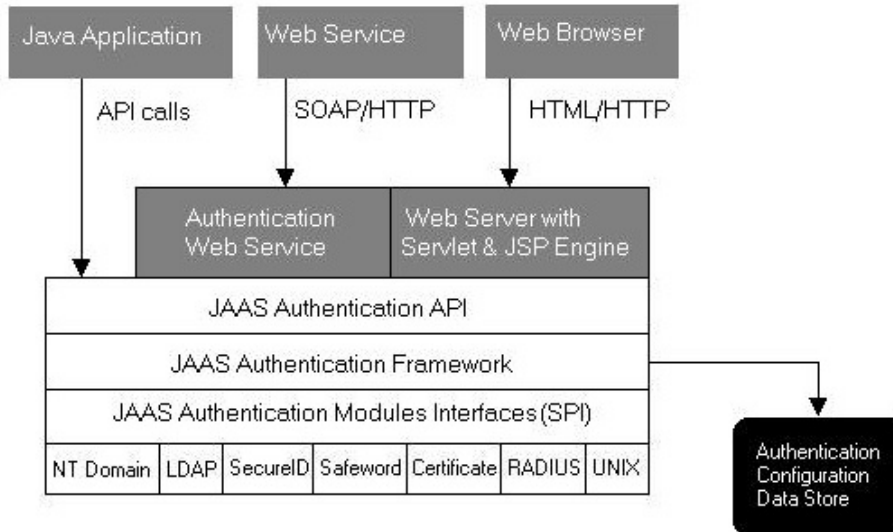
- Configured Identity: A J2EE container must be able to authenticate for access to a resource using the identity of the entity requesting the resource and authentication data specified at deployment time.
- Programmatic Authentication: The J2EE product must provide for specification of the user and authentication data for a resource by the application component at runtime using appropriate APIs.

### **Sun ONE**

The Authentication component manages the authentication modules of Figure 14, and uses the Java Authentication and Authorization Service<sup>12</sup> (JAAS) software as its framework. Figure 15 shows the architecture of the Authentication component.

---

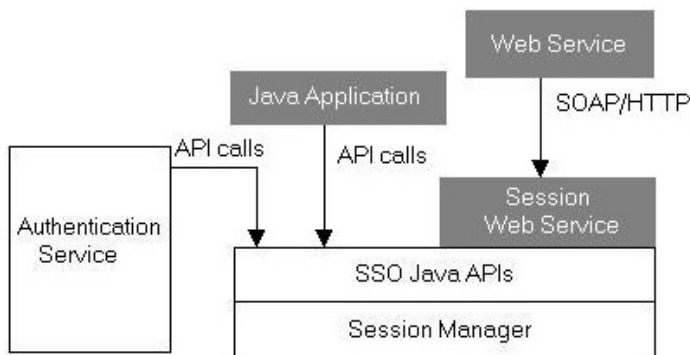
<sup>12</sup> <http://java.sun.com/products/jaas>



**Figure 15. Authentication Component Architecture.**

JAAS provides the client authentication APIs, the authentication framework, and plug-in Service Provider Interfaces (SPIs). Programmers developing application in the Java programming language can use the authentication APIs to authenticate users. The Authentication component also provides a SOAP-based Web service to authenticate users in a client-service environment and a HTML-based user interface that can be used by Web applications.

The authentication modules provided by the authentication component are RADIUS, Certificate Based Authentication (PKI), Safeword, SecureID, UNIX, NT Domain, and LDAP. The SSO component provides interfaces that can be used by Web services and applications to maintain a user's authenticated session across multiple Web services and applications, without having to re-authenticate the user. The architecture of the SSO component from Figure 14 is shown in Figure 16. It contains SSO Java technology APIs and a Session Manager.



**Figure 16. SSO architecture.**

The SSO APIs provides methods that can be used in the Java programming language to create an SSO session for a user and to build an SSO service that can be accessed by Web services. The Session Manager handles SSO session information such as time of authentication, idle time, and maximum session time.

The SSO architecture enables a tight integration with the Authentication component. The Authentication service calls SSO APIs to construct an SSO session for an authenticated user. The SSO component responds by providing an SSO token for the user.

The SSO component also provide a Session Web service that can be accessed by Web services to validate SSO tokens and obtain session related information.

The SSO solution is intended to operate in a single domain where participating Web services and applications work in a trust relationship environment. Enabling SSO across multiple domains is an issue addressed by the Liberty Alliance Project. This is further discussed in section 5.2.7 Shared context.

Sun ONE provides identity management, which gives a common way for the administrator to manage user information. In addition it provides interfaces for users to self-register and maintain their preferences, using the Profile API.

The Policy component provides two kinds of interfaces:

- An interface for obtaining and setting policy definitions such as who can access which resource and what the conditions for accessing it is.
- An interface to evaluate policies. For example if a particular user is allowed to perform a certain action on a given resource.

The interfaces both come as Java APIs and Web services. The Policy component follows the OASIS standards XACML and SAML for policy definition and evaluation.

The Policy framework keeps a data store to persistently store policies. In addition it provides infrastructure components that allow applications and servers to register policy schema, services to obtain policy evaluation results, and for administrators to set up policies for users as well as applications.

Privacy is a user-defined policy that determines who can access information and what can be done with it.

## **Microsoft .NET**

Role-based security defines the way that the .NET framework handles authentication as well as authorization. The most common authentication routines are available to .NET framework applications through various authentication providers, which are code routines that verify credentials, create the proper Identity and Principal object and attach it to the context of the request. A Principal object represent a security context where the code is running, whereas the Identity object represents the identity of the user associated with that security context.

Some of the authentication providers supported by .NET include:

- Forms-based (Cookie) authentication: This provider redirects unauthenticated calls to a specified HTML form using client redirection. Here the user post relevant logon information that is sent back to the server. If the server authenticates the user, ASP.NET issues a cookie that contains the user information needed to reacquire user identity. Any subsequent requests are issued with the cookie in the request header so that no manual authentication is needed.
- Passport authentication: Passport is a centralized authentication service provided by Microsoft that offers a single logon facility and membership services for all participating sites. Passport will be discussed further in section 5.2.7 about shared context.
- IIS: Microsoft's IIS server provides several built-in authentication mechanisms. These can be used to provide authenticated identities to IIS-hosted applications. Supported authentication mechanisms include HTTP Basic Authentication, NTLM, Kerberos, Digest Authentication, and X.509 Certificates (with SSL).
- Windows authentication: Windows support a number of authentication mechanisms that can be used by applications (via the SSPI subsystems), including Kerberos, NTLM and X.509 Certificates.

The architecture for authorization to application code is provided by ASP.NET in two methods:

- File authorization: the request location is mapped to a physical file, granting or denying access by matching the file's Access Control Lists (ACLs) with the identity making the request.
- URL authorization: access is granted or refused specifically by mapping users and roles to pieces of the URI namespace.

Another authorization method is the Windows-based impersonation. This is a method that keeps an identity tied to one user account throughout the flow of an application instead of having to periodically hand over control to the process under which the application runs.

Authentication and authorization is also handled in the evidence-based security. The key elements of the .NET framework evidence-based security are policy, permissions and evidence. The policy defines what resources code in executing assemblies may access. The policy is only accessible to system administrators and is installed automatically on every machine for every user account or it can be deployed across Windows domains via Group Policy.

Permissions describe one or more resources and associated rights, and implement methods for requesting and granting access. Using permission requests within assemblies developers have some ability to control how their code reacts to permissions granted by policy. There are three kinds of permission requests: Minimal, Optional and Refuse. For example, if a policy does not grant an assembly everything listed in the "minimal" set, then the assembly will fail to load and will not run.

The evidence of an assembly is used at runtime by the CLR to determine which permissions can be assigned to that assembly. Evidence can come from a variety of sources within the assembly either as a cryptographically sealed namespace, a software publisher identity (Authenticode) or code origin (URL, site etc.).

Working together the policy matches permissions based on evidence. If all resources involved are covered by sufficient permissions and properly configured policies and the developer uses managed code to access them, then the required evidence-checking and policy enforcement is handled transparently by the .NET framework.

Developers are also free to write their own authentication and authorization code.

### **5.2.1.2 Encryption/Secure Transfers**

#### **J2EE**

As mentioned in the beginning of this section on Security, J2EE does not dictate any specific security technology, so the choice of encryption technology is entirely up to the ones who implement the J2EE specification.

#### **Sun ONE**

Security is realised using a public-key infrastructure (PKI) and Kerberos. PKI consists of protocols, services and standards supporting applications of asymmetric and symmetric key cryptography. PKI has mechanisms for:

- Encryption and decryption of data.
- Digital signing and verification of documents
- Building a trust hierarchy based on public keys using Certification Authority (CA) and Registration Authority (RA)

Kerberos is a network authentication protocol designed to provide strong authentication in a client-server environment using secret-key cryptography. User authentication with Kerberos is achieved using a Kerberos authentication module in conjunction with JAAS.

### **Microsoft .NET**

Just as Sun ONE, the .NET framework uses Kerberos as one of its authentication mechanisms (mentioned in 5.2.1.1). Cryptographic primitives are available from managed code libraries for encryption, digital signatures, hashing and random number generation.

### **5.2.1.3 Code security**

#### **J2EE**

The Java language compiler and runtime system implement layers of defence against potentially incorrect code [31]. The Java language environment has as its starting point that no code is to be trusted, and proceeds according to this. As a result of this “no trust” policy, all incoming code is subject to bytecode verification. The bytecode verifier scans bytecodes, constructs the type state information, and verifies the type of the parameters to all bytecode instructions.

The next line of defence is in the class loader. All classes are partitioned into namespaces, one for the classes that reside locally, and one for each network source. Classes imported from across the network are placed into the private namespace associated with its origin. When a class references another class, it is first looked for in the namespace for the local file system and then in the namespace of the referencing class. With this scheme there is no way that an imported class can impersonate a built-in class. Built-in classes can never accidentally reference a class in an imported namespace; these can only be referenced explicitly.

#### **Sun ONE**

Since the Sun ONE platform is based on the Java language platform, the same code security principles apply as the ones for the J2EE platform.

### **Microsoft .NET**

Code access security is an enforcement engine that ensures that assembly code does not exceed its granted permissions while executing. When managed code assemblies are loaded they are associated with a corresponding set of permissions. Using a technique called stack walk, any attempts of sneak attacks from untrustworthy code via another assembly are prevented. This happens when any of assemblies in the call-chain do not have permission to access the requested resource. When using .NET framework classes and libraries where policies and permissions are already defined, this work is handled behind the scenes.

The final step in ensuring runtime safety of managed code is the verification process. This occurs during JIT compilation where the CLR verifies all managed code to ensure type safety. This eliminates the risk of code performing unexpected actions, such as uncontrolled pointing to memory locations, access to private data fields or accessing an object not yet initialised.

The .NET framework offers a way to segregate parts of applications through something known as application domains. Normally this kind of isolation is provided by the operating system by running each application in a different process with different address spaces, thereby preventing them to directly interfere with each other. In a large scale this can have a huge effect on performance. Using the type-safety of verified managed code the CLR can provide a large degree of isolation within the process. A single process can contain multiple application domains without the danger of any harmful interference between them.

The .NET security relies heavily on managed code and the CLR. Unmanaged code is not constrained by the security measures of the CLR and can therefore be capable of performing unauthorized access to resources. The .NET framework class libraries implement managed code wrappers that can be used for any unmanaged code methods. These wrappers take care of verifying the caller permissions and parameters and call the appropriate unmanaged code. Nevertheless, unmanaged code should be avoided as far as possible and whenever used extreme caution is required.

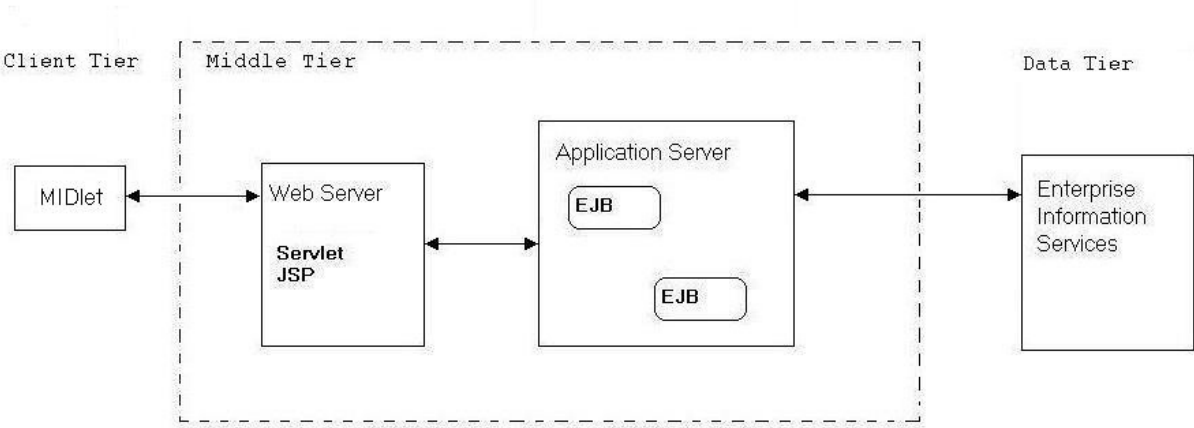
**5.2.2 Mobility**

When delivering Web services to Web-enabled devices other than regular PC’s or servers some extra considerations has to be made. Devices such as PDA’s, Web-enabled mobile phones or appliances and machines all suffer from limited processing capabilities, limited memory and unreliable periodic Internet connectivity. Also, there are numerous different devices with a wide variety of operating system versions, physical capacities (e.g. screen size) and access protocols. Mobile environments provide a different kind of user interface environment that is limited to interaction with the documents and very simple document creation.

**J2EE**

To create Java-technology based solutions for wireless devices, J2EE can be used in conjunction with the Java 2 Platform Micro Edition (J2ME). Internet-ready devices can be programmed using the Mobile Information Device Profile (MIDP), and a set of Java APIs that together with the Connected Limited Device Configuration (CLDC) provide a complete Java runtime environment [32].

MIDP applications, or MIDlets, can be developed using the J2ME Wireless Toolkit and be installed on any MIDP-compliant device. The MIDlet constitutes the client tier and provides the user interface on the mobile device. The MIDlet communicates with the Java servlet in the Web Server of the J2EE Middle Tier as illustrated in Figure 17.



**Figure 17. The J2EE-J2ME Application Architecture**

HTTP provides the bridge between MIDP and the J2EE Platform, since this protocol is supported on both the J2EE and J2ME platform.

**Sun ONE**

Delivering Services on Demand through Sun ONE to mobile devices is done with the Web client model. The model is applications built with Java technology delivered over the Web to a Web-enabled device.

The architecture for the Web client model assumes local storage in message queues, caches and device-specific stores, and storage is done in XML. This enables disconnected devices to present information collected while online and then update server-side elements when reconnected.

Full-scale Java Virtual Machine (JVM) software and the reduced-capability Kernel Virtual Machines (KVMs) are available on both traditional workstation-like machines and the more limited portable devices. JVM and KVM provide a common application and application library execution environment, which offer secure operations for accessing the underlying device-specific capabilities. The Sun ONE architecture support numerous Java APIs for these operating environment, listed among else in section 10.2 of the Sun ONE Architecture Guide [30].

Java Web client applications written for mobile devices execute on top of the device's profile. The application uses the data access, presentation, user interaction, and communication features provided by the profile. Because of the simple user interface interaction capabilities access to services must be set up using facilities for the service provision point for the device. This again requires that the device must have the basic infrastructure for interaction, consisting of reliable messaging, security, synchronisation, and access to local storage. The queuing of interaction for periodic connections is provided using the Java Message Service (JMS) APIs. With the JMS technology, multiple sources and targets can be identified with a common name instead of keeping a list of users who subscribe to the information that is published.

### **Microsoft .NET**

Developing mobile Web applications in the .NET framework is achieved using mobile Web Forms, which are an extension to ASP.NET. The Web Forms controls adapt the rendering of their output to fit to the different devices using the appropriate markup language. The mobile Web Forms controls support the Wireless Markup Language (WML) version 1.1, HTML version 3.2, and compact HTML (cHTML).

Microsoft Mobile Internet Toolkit<sup>13</sup> (MMIT) provides the technology and tools to build, deploy and maintain applications for the mobile environment. It is tightly integrated with the Visual Studio .NET IDE so a developer can create mobile applications in the same way he would create any other application. MMIT takes advantage of Web Forms so that correct markup is generated for the requesting device.

A recent release from Microsoft is the .NET Compact Framework, a subset of the .NET Framework, which is the smart device development platform for the .NET initiative. It brings managed code, XML Web services and all other features of Visual Studio .NET to PDAs, cell phones etc. This also means that existing code can easily be transferred to the mobile environment, and the developer can continue to use his Visual Studio .NET developer skills without having to learn to master a new mobile programming platform.

### **5.2.3 Supported platforms**

#### **Microsoft .NET**

As one could expect, the platforms supported by the .NET framework are the different Microsoft Windows versions. .NET client applications are required to support any of the Windows 98, NT, ME, 2000 or XP versions that are available. For .NET server applications the requirements are narrowed down to Windows 2000 Advanced, Server or Professional Edition, or Windows XP Professional.

---

<sup>13</sup> <http://msdn.microsoft.com/vstudio/device/mitdefault.asp>

## **Sun ONE**

With the exception of Windows XP, Sun ONE supports the Microsoft platforms mentioned above, in the client and server environments. The Solaris SPARC platform versions 2.6, 7.0 and 8.0 are supported, with the 8.0 version supported throughout the entire Sun ONE infrastructure software and the two former versions are supported by most of this software. A more detailed overview, including library and patch compatibility requirements, can be found at <http://www.sun.com/software/sunone/starterkit/platforms.html>. This overview also includes requirements for other platforms that are supported, including HP-UX 11.0, Linux 6.2, IBM OS/390 and Compaq.

## **J2EE**

The J2EE Software development kit lets you develop applications for the J2EE platform on Solaris, Windows NT and 2000, and Linux. The J2EE specification does not mention any restrictions on which platforms the J2EE platform can be implemented.

### **5.2.4 Supported protocols**

The Microsoft .NET Web Services currently support three Internet transport protocols: HTTP GET, HTTP POST and SOAP. These are all standard protocols for the Web.

The Internet transport protocols supported by J2EE in the J2EE specification are HTTP 1.0 and SSL 3.0.

Sun ONE has support for SOAP, HTTP and SSL.

### **5.2.5 Component definitions**

There seems to be no additional ways to describe the components of a Web service other than WSDL. The WSDL describes what the Web service does by describing the operational information related to the Web service, such as method names and number of parameters required.

To further describe the Web service, for example how it operates behind the scenes or if any other Web services are involved, the best approach probably is to add this information in a text-file that can be accessed via a Web service method.

### **5.2.6 Legacy system integration**

It was mentioned in section 2.5 how Web services can be used to integrate applications both within the enterprise borders and across these borders. By wrapping legacy system applications in to Web services these applications can communicate independent of origin and implementation.

This section will explore what other options the different architectures offer for legacy system integration, besides the Web service wrapping.

## **J2EE**

Legacy system integration on the J2EE platform can be achieved in a variety of ways, depending on what kind of legacy system is involved.

The J2EE Connector architecture defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EISs). The Connector architecture achieves close coupling between the application and the EIS by minimizing the number of layers between them. EIS vendors or other third-party vendors specialising in enterprise application integration use the Connector architecture to develop standard resource adapters for different EIS types. A resource adapter is a Java programming language library that is specific to the EIS. Because these resource adapters conform to the Connector architecture

specification they can be plugged into any J2EE-compliant service container and provide connectivity between the EIS and the middle tier.

The collaboration between the service container and resource adapter is transparent to the application accessing the EIS. This is achieved using two kinds of contracts defined by the Connector architecture:

- A system-level contract defines a “pluggability” standard between the EIS and the service container. The EIS-vendor implements its side of the system-level contract in the resource adapter. When the resource adapter is plugged into a service container, the container’s functionality is extended to include connectivity to the resource adapter’s EIS.
- An application-level contract is a contract between an application component and a resource adapter. The contract defines the client APIs the application component can use for EIS access.

The JDBC API is an industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API makes it possible to establish a connection with a database or access any tabular data source, send SQL statements, and process the results.

The Java Naming and Directory Interface (JNDI) technology provides a unified interface to an enterprise’s naming and directory services. A directory service provides access to a variety of information about users, machines, networks, services and applications. This information is categorized into namespaces by a naming facility in the directory service. JNDI provides directory and naming functionality to Java applications, independent of any specific directory service implementation.

### **Sun ONE**

The Sun ONE architecture has three facilities for integration with EIS applications:

- The J2EE Connector Architecture
- Asynchronous reliable messaging
- The Sun ONE architecture’s native support for Web services

The J2EE Connector Architecture extends the service container within the Sun ONE platform to include the functionality of an EIS.

As mentioned in the J2EE part above, the Connector architecture defines a tightly coupled integration between the service container and the EIS. In cases where a more loosely coupled integration is desired, the Sun ONE architecture provides the standard Java Message Service (JMS) API for asynchronous reliable messaging. This enables the integration of Sun ONE applications with an enterprise’s Message Oriented Middleware (MOM) environment [30].

This facility allows Sun ONE applications to exchange messages with the EIS system. The Sun ONE architecture’s support for Web services extends the two aforementioned facilities by exposing the functionality of the application accessing the EIS as a Web service. The Sun ONE Integration Server is used to integrate with existing legacy applications and databases. It ships with a set of pre-built resource adapters for integration with EIS applications from numerous vendors, and has a core messaging technology consisting of XML over HTTP (for Web service communication) and JMS (for asynchronous reliable messaging).

### **Microsoft .NET**

The data access model in the .NET framework is ActiveX Data Objects .NET (ADO.NET). ADO.NET is an extension of ADO, with the most noticeable change being that ADO.NET is

entirely based on XML. As a result of this XML foundation a Web page accessing a database with ADO.NET can display the content of a database table without any manual intervention. To achieve legacy system integration beyond regular databases requires one of Microsoft's .NET Enterprise Servers. These work tightly with the .NET Framework development environment and include support for XML Web services. The Microsoft Enterprise Server family include, among else:

- BizTalk Server: which can be used to build and deploy XML-based business processes across applications and organisations.
- Commerce Server: for building e-commerce solutions.
- Exchange Server: for messaging and collaboration.
- Host Integration Server: provides integration of data, application and network to extend Windows to other systems.

### 5.2.7 Shared context

Both Microsoft and Sun support shared context but have different approaches to achieve this, through Microsoft .NET Passport and Liberty Alliance respectively.

#### Microsoft .NET Passport

Microsoft .NET Passport is an online service providing a common Internet authentication across Web sites.

When registering for a .NET Passport all the user is required to provide is an e-mail address or a phone number. All other information is optional, even first and last name. During registration the user can make choices regarding what information he/she wishes to share with participating Web sites. The user credentials specified at registration are linked to a .NET Passport Unique Identifier (PUID) assigned by the .NET Passport service.

Also an optional feature is the .NET Passport wallet, where user can store credit card information and billing and shipping addresses. The wallet is used in the .NET Passport express purchase, which enables users to transfer this information securely to a merchant in order to transact an online purchase.

Sites become participating .NET Passport sites by implementing the .NET Passport authentication service called the .NET Passport single sign-in (SSI). To implement the SSI the participating sites must install the .NET Passport Manager, which is a COM object that decrypts .NET Passport cookies, manages authentication and profile access, caches the user's authentication and profile information in cookies on the user's browser, and re-verifies the cookies as the user moves from page to page around the site. Figure 18 shows how this scheme works.

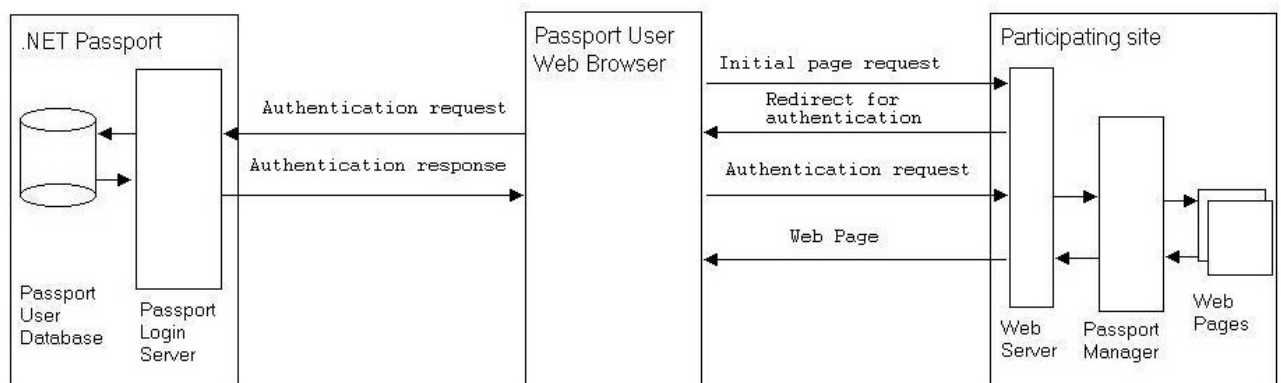


Figure 18. .NET Passport authentication flow.

When users sign in to a site, they are redirected to a secure .NET Passport Login server. .NET Passport first verifies that the site requesting the authentication is a valid participating site. Then it displays a page that asks users for their credentials. When .NET Passport verifies that the credentials correspond to a valid .NET Passport user, the user is authenticated. The user's PUID is sent to the site in a ticket encrypted using a key specific to the site. The user's .NET Passport password is never sent to participating sites.

There are currently three levels of security provided by .NET Passport. Standard sign-in is intended to be used by sites that do not require a high level of security. SSL is used only when username and password is sent to the .NET Passport Login servers.

With secure channel sign-in security is improved because all communication is end-to-end SSL protected.

The most secure sign-in mechanism is the strong credential sign-in, where the solution for protecting secure content is to have a two-stage sign-in process. The first step is identical to the secure channel sign-in. Step two involves a second sign-in page where the user is required to enter a four-digit security key.

.NET Passport is also available on mobile devices and support browser and devices like Nokia WAP browsers, PocketPC 3.x OS and WinCE 3.x. Some features, such as strong credential sign-in are not available on the .NET Passport version for mobile devices.

### **Liberty Alliance**

The Liberty Alliance Project<sup>14</sup> is an organisation formed to create an open federated single sign-on identity solution for any device connected to the Internet. A federated identity system ensures that the use of sensitive personal information is managed and distributed by the appropriate parties, rather than by a central authority. This can be seen as a reaction to the .NET Passport concept, where all user details are stored in .NET Passport database managed by Microsoft.

A federate identity system enables the development of federated commerce. With federated commerce businesses are allowed to maintain ownership of their customer directory, and extend their directory more effectively to participate in closely related marketing and partnerships with other businesses. This in turn will provide end user with more convenience, choice and control of their identity.

The Liberty Alliance Project is a fairly new organisation. It was established in September 2001, on the initiative of several large corporations, including AOL Time Warner, General Motors, American Express, Hewlett Packard and Sun Microsystems and many more. The three main objectives of the Liberty Alliance Project are:

- Allow individual consumers and businesses to maintain personal information securely.
- Provide a universal open standard for single sign-on with decentralized authentication and open authorization from multiple providers.
- Provide an open standard for network identity spanning all network-connected devices.

No formal specification exists of yet from the Liberty Alliance Project but future version of the Sun ONE architecture will require interfaces to the standards proposed by the Liberty Alliance.

---

<sup>14</sup> <http://www.projectliberty.org>

### 5.2.8 Interoperability

The Sun ONE architecture's support of XML, SOAP, WSDL and UDDI has been demonstrated to be interoperable with the .NET implementations of the same standards [30]. Therefore the Sun ONE architecture is able to consume .NET Web services and provide Web services that may be consumed by the .NET environment.

Both the Visual Studio .NET and Sun ONE Studio IDE may be used to develop applications that consume .NET as well as Sun ONE Web services.

Early February this year saw the foundation of the Web Service Interoperability Organization, WS-I<sup>15</sup>. WS-I is an open industry organization that aims to promote Web service interoperability across platforms, operating systems and programming languages. Founders of WS-I include Microsoft, Oracle and IBM, but quite surprisingly Sun has not yet joined in but are expected to do so in the near future.

The WS-I will not be working to develop standards but will rather endorse and encourage the use of standards promoted by the likes of the W3C, IETF and OASIS. The WS-I has announced three goals:

- Provide implementation guidance and education to help customers with Web services adoption.
- Promote consistent and reliable interoperability among Web services across platforms, applications, and programming languages.
- Articulate and promote a common industry vision for Web services interoperability to ease customer decision making, grow industry adoption of Web services and ensure the continued evolution of Web services technologies.

To achieve these goals the WS-I plan to create a set of interoperability testing tools and publish an architectural roadmap that will try to identify the future directions of Web service evolution.

WS-I also plans to introduce the concept of Profiles [35]. A Profile is a named group of Web service specifications at specific version levels along with conventions on how they work together. The first Profile proposed by the WS-I is the WS-I Basic Web services, which consist of the four specifications that provide the basic functionality of Web services, XML Schema 1.0, SOAP 1.1, WSDL 1.1, and UDDI 2.0. One or more WS-I working groups will prepare the conventions and best practises associated with this Profile.

The number of additional Profiles depends on the evolution of Web service standards, both existing and upcoming.

Whether or not the WS-I will prove to be a successful initiative remains to be seen. Its work is still in the start-up phase and to have multiple vendors agree on what is a best practice for Web services could easily become a tedious task.

### 5.2.9 Ease of use

It is hard to judge the ease of use for each architecture, since this a criterion very specific to each individual. One can only try to make some general suggestions as to how easy it is to work with the different architectures.

### J2EE

The ease of use for the J2EE platform depends first and foremost and what kind of implementation one chooses. One needs to find the implementation that best matches ones needs, and from there on it all depends on what the company responsible for the

---

<sup>15</sup> <http://www.ws-i.org>

implementation offers. This can range from standard templates and wizards to drag and drop design interfaces.

### **Sun ONE**

A good user experience with the Sun ONE platform depends on how well one gets familiarized with the Sun ONE Infrastructure Products. All these products supposedly have easy to use interfaces that let the users configure, manage and utilize the functionality offered by the specific product. The Sun ONE Infrastructure Products also offer easy and tight integration between each other.

### **Microsoft .NET**

The ease of use offered by the .NET Framework can be found in the most levels of the architecture. The Framework class libraries offer functionality that can be derived, and the ADO .NET classes simplify data access. Then there are the Web Forms and Windows Forms application models that also reduce the programming burden in the Web and Windows development areas.

## **5.2.10 Scalability**

None of the technologies under evaluation mention anything particular on scalability other than saying that the architecture is scalable. No particular scalability measurements exist but some thoughts on the scalability of J2EE versus .NET is found in [26]. Here it is stated that both implementations of J2EE and .NET allows you to add additional machines to increase user load while maintaining the same response time. It is further stated that the significant difference between J2EE and .NET scalability is that since .NET supports Win32 as the underlying hardware, a larger number of machines than a comparable J2EE deployment are needed to achieve the same performance. This is mainly due to processor limitations. There is no reference to Sun ONE scalability but it should match up to the scalability of the J2EE platform since the Sun ONE Application Server is J2EE compliant and the entire Sun ONE platform is based on Java.

## **5.2.11 Deployment**

### **J2EE**

A J2EE application consists of one or more J2EE components and one J2EE application deployment descriptor. The deployment descriptor lists the applications components as modules. A J2EE module represents the basic unit of composition for a J2EE application, and each module is made up of one or more J2EE components and a module-level deployment descriptor.

The deployment descriptor for a J2EE module contains declarative data required to deploy the components in the module. The deployment descriptor also contains assembly instructions that describe how the components are composed into an application.

The J2EE application deployment descriptor is provided using an XML DTD, described in section 8.4 of the J2EE specification [36].

Deployment of Web services on the J2EE platform requires a deployment tool. What the deployment tool usually does is to validate the content of the XML deployment descriptor as a correctly assembled deployment artifact, collects binding information from the deployer, deploys the components and Web services defined within the modules, publishes the WSDL document belonging to the deployed Web services, deploys any clients using Web services, configures the server and starts the application.

## **Sun ONE**

Deployment of applications on the Sun ONE platform is done in the service container. This service container is a Web application server configured with J2EE technology. The J2EE deployment considerations from above therefore also apply to Sun ONE Web service deployment.

The Sun ONE Application Server constitutes the service container for the Sun ONE platform. It ships with tools that help ease the process of packaging and deploying applications. These are the Command Line Interface (CLI), the Deployment Tool, and Visual Café plug-in. The Sun ONE Studio tools also offer means for easily assembling and deploying applications on the Sun ONE Application Server.

## **Microsoft .NET**

There are different approaches to deployment in the .NET framework environment. The easiest approach regards deploying a stand-alone application. In this case all that is needed is to copy the file to a computer that runs the CLR. Nothing else is needed, no installation process or registry entry. The application is removed simply by deleting the file.

Component-based applications are slightly more complicated, depending on whether components are private to application, shared with related applications, or shared with potentially unknown applications.

If the components are private to the application deployment is done exactly as with the stand-alone application. Similarly, if several related applications use the same assembly, the assembly can be located in a common subdirectory.

If the application uses assemblies that are shared with undetermined applications, these assemblies must be installed in the assembly cache and must exhibit certain properties, such as unique name and versioning information. These properties are used to ensure that the CLR binds the application to the appropriate component versions.

An important feature of .NET Framework applications is the ability to maintain application configurations in plain text files. This allows an administrator to tailor an application's behaviour on a particular computer without involving developers.

The simplified deployment scheme of the .NET framework is a result of the fact that components are self-describing, with all necessary information contained within the assembly. No separate files are needed for class definitions; all definitions for the component can be obtained by inspecting the metadata that are included within the assembly. The self-describing property of an assembly also leads to that no explicit registration with the operating system is required.

### **5.2.12 Tool support**

#### **J2EE**

If one is interested in a large selection of tools to develop with, then the J2EE architecture is the one to choose. J2EE implementations come from a multitude of vendors, such as IBM, Oracle and BEA. The integrated development environments (IDE) contained in the J2EE product portfolio include Web Grain's Visual Café, IBM's VisualAge for Java, Borland's JBuilder and more.

The Java Web Service Developer Pack Early Access 2 (WSDP EA2), available from Sun, contains all the necessary technologies for developing Web services on the J2EE platform.

The technologies include:

- The Java XML Pack, which consist of the JAXM, JAXP, JAXR and JAX-RPC APIs (described in 4.2.2).

- The Java Server Pages Standard Tags Library (JSTL), which encapsulates core functionality common to many JSP applications. This implies, among else, that a single tag can be used on multiple JSP containers.
- Apache Tomcat 4.1-dev Container, which is a Java servlet and JSP container for deployment and execution of Web services.
- Registry Server, which implements version 2 of UDDI and offers mechanisms for registering and discovering Web services.
- `Deploytool`, which is helps developers package, configure and deploy Web services on the Tomcat server.

### **Sun ONE**

Sun ONE Studio is the IDE for Sun ONE. The Sun ONE Studio IDE is based on the NetBeans<sup>16</sup> software IDE, which is open source, modular and standards-based. It is written in the Java programming language and can run on any platform with a Java Virtual Machine that is compliant with the J2EE platform. The Sun ONE Studio IDE extends the NetBeans software to provide a richer set of functionality that enable the developer to produce Services on Demand for the Sun ONE architecture.

The modularity of the Sun ONE Studio IDE enables developers to add modules that provide additional functionality, such as editing, debugging and error highlighting not just for the Java programming language, but other languages as well. In addition to Java the IDE works with C/C++, Fortran, UML, IDL, XML and others.

The primary components of the Sun ONE Studio IDE provide direct support for numerous standards and products that are defined by the Sun ONE architecture. These include:

- J2EE platform support, which includes EJB, JSP, Java Servlet and support for the J2EE Connector Architecture.
- Support for the Java APIs for XML interactions as described in section 4.2.2.
- Tools that enable the use of SOAP, WSDL and UDDI for applications and services.
- Legacy integration using JDBC, the J2EE Connector Architecture or Sun ONE Integration Server.
- Solaris Operation Environment support, which include Forte IDE Developer Compilers for C, C++ and Fortran.

There are numerous ways to creating services with the Sun ONE Studio. One approach is to use the Java Web Service Designer (JWSD). With JWSD existing Java class libraries, Beans and EJB can be exposed as SOAP-based Web services via the creation of XML Operations (XOP).

When dealing with other languages than Java, the Native Connector Tool (NCT) and the Native Connector Architecture (NCA) can be used to expose existing native libraries written in C, C++, Fortran or even Microsoft CLR-compliant languages as either Java platform-based classes/beans or directly as SOAP-based Web services.

There is also an XML Designer tool that developers can use to build adapters for existing application components.

The Sun ONE Studio also enables the integration of services created on other platforms such as .NET. This is achieved using UDDI for dynamic discovery of services and SOAP/XML for communication.

---

<sup>16</sup> <http://www.netbeans.org>

## Microsoft .NET

For the .NET platform Visual Studio .NET has been launched as the IDE. It supports all languages previously supported by Visual Studio, with the exception of Java, which has been replaced with C# and J#, two new object oriented programming language from Microsoft. Any programming language can be used to produce code for the .NET platform, provided that the language compiler can produce managed code targeted for the common language runtime. Visual Studio .NET contains a complete toolset for building, consuming, discovering, registering, testing, deploying and maintaining Web services.

Building a Web service can be as easy as just attaching the attribute “<WebMethod()>” to an existing component, and thereby transforming it to a Web service. A wide variety of templates and wizards based on the ASP .NET application model are also available, simplifying development even further.

Developers can incorporate existing Web services into their own solution simply by adding a *Web Reference*. With this reference in place, developers can program against the Web service as if it were any other local component within their project.

Microsoft has incorporated a direct interface to UDDI in Visual Studio .NET. This gives direct access to UDDI registers for publishing and discovering Web services. This is achieved using two HTML-based pages, the *Find a Service* page for searching UDDI and the *Register Service* page for obtaining an UDDI account and publish the Visual Studio .NET Web service.

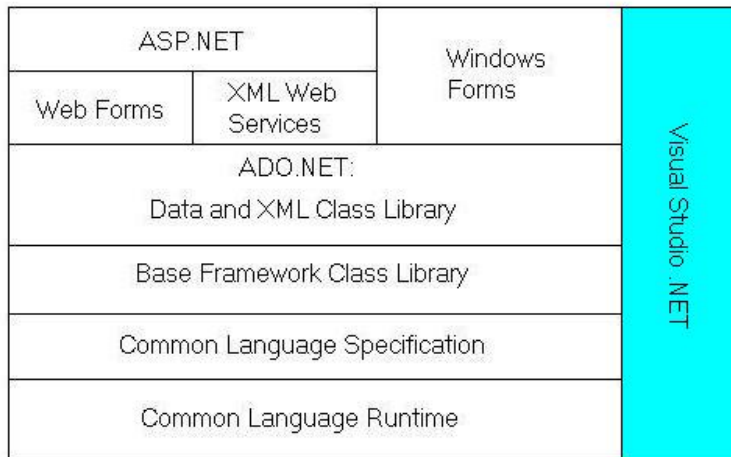
The Application Center Test (ACT) is an integrated feature of Visual Studio .NET that developers can use for performance or functional testing, gathering metrics and status information within the development environment. The ACT offers several methods for creating tests for Web services. One of the highest-level test methods is the *Browser Record Test*, which quickly and easily creates a test by recording HTTP requests sent to the Web service. The more detailed test methods involves stress tests that are created within Visual Studio .NET using scripting languages such as VBScript or Perl.

Visual Studio .NET provides access to a wide variety of 3<sup>rd</sup> party .NET hosting services where one can deploy Web services. The principle behind this is called “One-Click Hosting” and involves an interface that enables a developer to select a hosting service, set up an account and begin deploying Web services at the touch of a button.

In addition to Web service development Visual Studio .NET is also an IDE for client and server-side development. Client development is based on a unified programming model so that the same IDE can be used to develop applications for desktop, the Web or mobile devices. This means that developers do not need to learn a new tool each time they change client environment.

Visual Studio .NET comes with a collection of tools for server side rapid application development (RAD), including the Server Explorer, Component Designer and a toolbox of server side components.

Standard based software modelling is also a part of Visual Studio .NET, enabling users to develop various types of UML diagrams, such as sequence-, use case-, state chart-, and activity diagrams etc.



**Figure 19. The .NET Framework with Visual Studio .NET.**

Figure 19 shows Visual Studio .NET as an integral part of the .NET Framework. It utilises all aspects of the .NET Framework, from the application programming model of ASP .NET and Windows form, the ADO .NET data access model, the base classes functionality and all the way down to the CLS and the CLR.

The toolset provided by the Java community does provide a lot more functionality than the tools provided by Microsoft. They do, however, not originate from a single vendor and cannot be said to be 100% interoperable. It can be a very tedious task to work with a mix of tools and none of the Java tools alone offers as much as Visual Studio .NET.

### 5.2.13 Modelling tools

Rational software has released the Rational XDE Professional tool, where XDE stands for eXtended Development Environment. The tool can be fully integrated into both the Java and the Microsoft .NET IDE. Rational XDE works inside the IDE, meaning that all its features show up within the program's menus and toolbars. The tool currently works with Visual Studio.NET (for the .NET platform) and IBM WebSphere Studio Application Developer (for the J2EE platform).

Rational XDE includes all the features of the traditional Rose products, such as generation of UML from code and the possibility of building visual designs of applications and turning them into code.

The XML Metadata Interchange Format<sup>17</sup>, XMI, is a standard, developed by the Object Management Group (OMG). XMI has as its purpose to enable an open interchange model for application components and related business assets [40]. XMI gives developers working with object technology the ability to exchange programming data over the Internet in a standardized way.

With XMI as the standard different application development tools can interchange their information. These can be design tools like Rational Rose, development tools like VisualAge for Java, database tools like IBM DB/2 or source code like C, C++ and Java. All that is required is that the vendor adds XMI support to the product.

The main functionality of XMI is the ability to generate XMI DTDs from UML models. Such an XMI-generated DTD for UML allows interchange of object-oriented UML models and

<sup>17</sup> <http://www.omg.org/technology/documents/formal/xmi.htm>

class definitions. Other information that can be exchanged includes database information, C++ and Java class definitions, IDL, Enterprise Java Beans and Electronic Commerce. What makes XMI interesting in this context is the fact that it uses XML for information exchange, just like Web services. This means that the XMI DTD that describes for example an UML model can be placed inside the body of a SOAP envelope for transmission. XMI can be found in Microsoft's modelling tool Visio, as an XMI export component that exports a Visio UML model to an XML file compliant with the XMI standard. Visio is a component of the Visual Studio .NET Enterprise Architect version. For the Sun ONE platform the Sun ONE Studio offers support with tools that implement the XMI standard. This is achieved using the Metadata Repository, which is part of the NetBeans software that is included in the Sun ONE Studio IDE. For the J2EE platform one can for example choose the XMI toolkit from IBM<sup>18</sup>. The XMI toolkit is a Java component that converts Java to Rational Rose and UML models and vice versa.

### 5.2.14 Billing

There are various approaches to billing service consumers on the Internet. Gisolfi [38] presents five potential revenue models for the Service provider of the service-oriented architecture (Figure 3, page 5):

- Transactional
- Membership/subscription
- Lease/license
- Business partnership
- Registration

The Transactional model is based on the pay-per-click or fee-for use model. This is the most primitive of all the models, and the charge per transaction is possible using payment instruments like credit cards. There are two approaches to this model. The revenue method may lie at the service interface, charging the user for each access. Alternatively, a relationship between the provider and consumer is established beforehand, and the provider can audit the usage of the service and bill for it on a periodic basis.

The Membership or Subscription model refers to a revenue model that regards an established user account with specific terms of usage. A user may register for service access for a certain period, or for a given quantity of usage. Just as with the Transactional model the revenue method may reside at the service interface, or the terms of usage can be established offline. The Lease or License model concerns a revenue model more suited for larger business-to-business relationships, which involves high usage frequency and a more customised agreement. The service provider may charge by volume of transactions or perhaps the number access points from with the service requester. The relationship for this kind of revenue model is established offline.

The Business partnership model is a new concept, aimed at B2B relationships, for example where two businesses collaborate to present a total solution to the customer. The relationship can be based on exchange of services or percentage sharing of the revenue collected from the service requester.

The Registration model refers to a revenue model that would apply more readily to a UDDI Gateway or Green Pages business. Collection of revenue is based on a pay to be seen concept. And the idea is that if a service provider wants to be seen they will be willing to pay a registration fee.

---

<sup>18</sup> <http://www.alphaworks.ibm.com/tech/xmitoolkit>

None of the architectures in question here mention any thing specific about billing. Which of these models and how they should be implemented is entirely up to the developers. The one thing that is fair to assume is that all architectures one way or another support some kind of transaction logic that will aid the developer during implementation.

### **5.2.15 Availability and price**

#### **J2EE**

On <http://java.sun.com/j2ee/download.html> you can find the specifications and downloads that are relevant for the J2EE platform, free of charge:

- The Java 2 Software Development Kit (SDK): this includes a binary version of the J2EE reference implementation, plus related development tools and documentation that enable developers to create distributed applications for the J2EE platform.
- Java Enterprise Blueprints: The Java BluePrints Program<sup>19</sup> helps developers create robust, scalable, and portable applications by providing guidelines, patterns, and code that illustrate best practices on how to build end-to-end applications using Java technology.
- The J2EE Platform Specification: Any use or implementation of this specification is subject to license, which among else states that the whole specification must be implemented as stated by Sun without any subsetting or supersetting.
- The J2EE Connector Specification: The use and/or implementation of this specification follow the same license agreements as the J2EE Platform Specification.

Implementations of the J2EE platform are available from vendors such as IBM, BEA, Borland and many more. Prices vary according to range of products and functionality offered.

#### **Sun ONE**

Everything a developer needs to get started with developing Services On Demand for the Sun ONE platform is found in the Sun ONE Starter Kit. It includes Forte for Java, both Community and Enterprise Edition (Enterprise Edition is a 60 day evaluation licence, full price is \$1,995), and the Sun ONE Application, Web, Portal, Integration and Directory Servers. The servers are provided under right-to-develop licenses. These are binary code license agreements (BCL) that generally last for a year and permit the end user to develop applications using these products. Licensing restrictions do however apply and terms may vary so the user should review the applicable BCL. The latest edition of the Starter Kit also includes the Java Developer Kit, J2EE 1.3, and the currently available parts of the JAX Pack. The Starter Kit can be ordered on CD and DVD, and can be ordered either as a one time shipment of the current version or as a year's subscription, which includes four shipments. Prices for a one-time shipment are \$19,95 for DVD and \$29,99 for CD, whereas the one-year subscription cost \$49,95 and \$99,99 respectively.

The Sun One Starter Kit can be installed on Solaris, Linux and Windows platforms.

#### **Microsoft .NET**

.NET connected software from Microsoft encompass a large family of products, all from clients that power smart devices, to servers, services and tools designed to support XML and incorporate industry Internet standards.

Client software that utilize the .NET framework include Windows XP for PC's, and Windows CE .NET and Windows XP Embedded for smaller devices such as PDA's. Windows XP for PC's come in two flavours: Home Edition and Professional. Both the Home and Professional

---

<sup>19</sup> <http://java.sun.com/blueprints>

Edition can be purchased at a retail store or of the Internet and prices are around \$199.00 and \$299.00 respectively, but one can also buy an upgrade package if one already runs a previous Windows version (98, NT, 2000 or Millennium Edition). These cost approximately \$100 less. With respect to Windows CE .NET and XP Embedded no particular pricing is mentioned. To purchase any of the two involves buying a licence and the price depends on the number of licences.

If one is interested in one of the Microsoft servers, most of these also come in two versions, Standard and Enterprise, with the Enterprise versions as the most expensive. For example, the BizTalk Server costs \$24,999 pr. processor for the Enterprise Edition and \$4,999 for the Standard Edition. For the Commerce server prices are at \$19,999 and \$6,999 respectively, but this also require the Windows 2000 Server and the SQL server. The choice of edition mainly depends on the size of the business wishing to incorporate a Microsoft server.

The .NET Framework Software Development Kit (SDK) is available for free at the Microsoft home page<sup>20</sup>. The .NET Framework SDK includes the .NET Framework in addition to everything else needed to write, build, test and deploy .NET Framework applications. The SDK runs on Windows NT, 2000 and XP Professional.

The .NET Framework Redistributable is another free download from Microsoft that includes all you need to run .NET application, including the CLR, the .NET Framework class libraries and ASP .NET. It works with Windows 98, NT, ME, 2000 and both XP editions.

Visual Studio .NET is available for purchase in three versions: Professional, Enterprise Developer, and Enterprise Architect. Of these it is the Enterprise Architect edition that has the most features, whereas the Professional Edition has the least. For example, the two Enterprise Editions both come with all of Microsoft's server technologies (with the exception of BizTalk, which only comes included with the Enterprise Architect edition). The costs of these editions are as follows (version upgrade price in parentheses):

- Visual Studio .NET Professional Edition: \$1,079 (\$549)
- Visual Studio .NET Enterprise Developer Edition: \$1,799 (\$1,079)
- Visual Studio .NET Enterprise Architect Edition: \$2,499 (\$1,799)

## 5.2.16 Summary

As the evaluation comes to an end it is time to sum up the results. Table 2 below shows the evaluation criteria related to the different technologies with a short description of the results found.

<b>Criterion</b>	<b>Java 2 Enterprise Edition</b>	<b>Sun Open Net Environment</b>	<b>Microsoft .NET</b>
Security: Authentication and Authorization	<i>Authentication with HTTP Basic, SSL and Form Based Login. Code and Caller authorization. Resource authentication and authorization</i>	<i>Authentication Component provides NT Domain, LDAP, SecureID, Safeword, Certificate, RADIUS, and UNIX authentication modules. SSO component. Authorization with the Policy component follows XACML and SAML</i>	<i>Authentication with Form Based Login, Passport, IIS and Windows authentication. File and URL authorization. Evidence-based security: policy, permissions and evidence</i>

<sup>20</sup> <http://www.microsoft.com>

Security: Encryption	<i>No specific security technology, depends on implementations</i>	<i>PKI Encryption Kerberos cryptography</i>	<i>Kerberos included in IIS authentication. Cryptographic primitives available in class libraries</i>
Security: Code access	<i>The Java Runtime Environment provides the bytecode verifier, which ensures type safety, and the class loader, which separates classes into namespaces according to origin.</i>	<i>Same as for J2EE</i>	<i>Permission associated with assemblies. “Stack-walk” prevents sneak attacks. Code verification during compilation ensures type safety. Application domains isolate application parts.</i>
Mobility	<i>J2ME that uses MIDP to create MIDlets that access J2EE over HTTP</i>	<i>The Java Web Client model. Applications for the JVM and KVM</i>	<i>Mobile Internet Toolkit together with Web Forms. .NET Compact Framework</i>
Supported platforms	<i>No restrictions mentioned. Development for Solaris, Windows NT and 2000, and Linux</i>	<i>Windows, Solaris SPARC, Linux, IBM OS/390, HPUX 11.0, Compaq and more</i>	<i>Windows</i>
Supported protocols	<i>HTTP, SSL</i>	<i>HTTP, SSL, SOAP</i>	<i>HTTP, SOAP</i>
Component definitions	<i>None</i>	<i>None</i>	<i>None</i>
Legacy system integration	<i>J2EE Connector Architecture JDBC</i>	<i>J2EE Connector Architecture JDBC Asynchronous reliable messaging</i>	<i>ADO .NET .NET Enterprise Servers</i>
Shared context	<i>None in particular but likely to follow recommendations from the Liberty Alliance Project</i>	<i>Coming recommendations from the Liberty Alliance Project</i>	<i>.NET Passport</i>
Interoperability	<i>Nothing mentioned</i>	<i>Interoperates with .NET Web Services</i>	<i>Interoperates with Sun ONE Web Services</i>
Ease of use	<i>Depends on implementations</i>	<i>The Sun ONE Infrastructure Products offers ease of use and tight integration with each other.</i>	<i>Functionality can be derived from class libraries. Easier development with Web Forms and Windows Forms</i>

Scalability	<i>Additional machines can be added without affecting performance</i>	<i>Approximately the same scalability as J2EE</i>	<i>Does not scale as well as J2EE as the number of additional machines grow</i>
Deployment	<i>Applications are required to have an XML deployment descriptor, which contains declarative data for deploying components of the application. Requires a deployment tool</i>	<i>Same deployment descriptor as J2EE. Tools for deployment are the Command Line Interface (CLI), the Deployment Tool, and a Visual Café plug-in</i>	<i>All deployment information contained in the assembly's metadata. Deployment = copy files to destination computer/folder.</i>
Tool support	<i>Numerous tools from vendors such as IBM, BEA, Oracle etc. Java Web Service Developer Pack EA2</i>	<i>Sun ONE Studio</i>	<i>Visual Studio .NET</i>
Modelling tools	<i>Rational XDE for IBM WebSphere. XMI toolkit</i>	<i>XMI support in Sun ONE Studio</i>	<i>Rational XDE and XMI support with Visual Studio .NET</i>
Billing	<i>No particular support</i>	<i>No particular support</i>	<i>No particular support</i>
Availability and price	<i>Download J2EE SDK for free</i>	<i>Sun ONE Starter Kit can be ordered on CD or DVD</i>	<i>Download .NET Framework SDK and Redistributable for free Visual Studio .NET available for purchase</i>

**Table 2. Summary of the evaluation.**

### **5.3 Evaluation related to the university scenario**

The next step in this evaluation is mapping the criteria just evaluated to the scenario in chapter 3. In this way the effects of the different technologies will hopefully be made clearer and a decision can be made as to which technology is best suited for realising the scenario.

#### **Security**

With respect to security the most central aspects of the scenario were the authentication of user and authorization of the different actors in the scenario. The technologies are quite similar on authentication issues. They all maintain user sessions so that once a user is logged on he/she remains logged on until the session is closed. The most common authentication mechanism, the form based login where the user enters username and password via a user interface, is also supported by all technologies.

The one thing that does stand out is the SSO and Authentication components on the Sun ONE platform. These components do not just expose APIs for programmers to add to their code, they also provide Web service interfaces. This implies that applications as well as Web services can easily utilize authentication and session logic. Related to the scenario, this is a

solution that fits in well with the Web services concept of the scenario. A course service can authenticate user logged on to the portal as well as those accessing it from outside the university, using the Web service interface of these two components.

To achieve this with the .NET framework would imply the development of a similar authentication Web service or use .NET Passport. Nevertheless this does not mount up since the first approach requires extra work and the Passport approach means that the service provider would have to sacrifice some control to Microsoft.

All the technologies in question also offer satisfying authorization mechanisms that help ensure that no unauthorized access takes place. The only insecure factor is the fact that .NET security relies on the managed code and the Common Language Runtime, so if you work with any unmanaged code you lose many of the security features of the CLR unless certain precautions are taken.

### **Mobility**

All the presented technologies contain more than sufficient support for developing Web services for mobile platforms. The question is not how to develop Web services for mobile devices, but rather how to modify mobile applications for access to Web services.

The technology that seems to promise the most is the .NET Framework, thanks to the Mobile Internet Toolkit and the .NET Compact framework. Using these a programmer can create applications for the mobile environment just as for any other development environment, and even transfer existing code to the mobile platform. The mobile Web forms is the key application programming model that renders the appropriate output for the device in question.

### **Platform support**

Choosing Microsoft .NET will restrict development to the Windows platform. J2EE and Sun ONE will not impose any restrictions since they support all of the most common platforms.

### **Component definitions**

None of the technologies in question offer any descriptions of Web services beyond WSDL. If the developers of the scenario are interested in further describing the services this must be provided for example via “read me” information accessible through the Web service interface.

### **Legacy system integration**

The best integration with legacy systems can be achieved using the J2EE Connector architecture. It is available on both the J2EE platform and the Sun ONE, and offers connectivity to any EIS that expose a resource adapter for the J2EE Connector architecture. For the scenario however, the EIS consist merely of a database with all relevant student information. It would therefore be sufficient to apply either the ADO .NET data access model or the JDBC API for the .NET Framework, and the J2EE and Sun ONE platform respectively. Both these models offer database interdependent access, and the only significant difference is that ADO .NET is entirely based on XML.

### **Shared context**

The only technology that offers anything concrete within the shared context area is the .NET Framework with .NET Passport. For the Sun ONE platform the work of the Liberty Alliance will be crucial to what directions the platform will take on issues surrounding shared context. For the scenario shared context would among else enable a tighter integration between universities that share courses. If two or more universities implemented the scenario and wanted to include courses from each other, it would be in everyone’s interest if the student

only has to log on to the portal at his/hers university and be able to access courses from other universities without having to re-authenticate.

This can be realised using .NET Passport, but it requires that every student registers for a Passport, and that the universities would be willing to let Microsoft handle this information instead of keeping it internally.

### **Interoperability**

In theory, Sun ONE Web services should be able to communicate with .NET Web services and vice versa. The same should also count if one throws Web services for the J2EE platform into the equation. If it works totally problem free in practice is however harder to assess. In addition there are a wide variety of SOAP tools that are used for developing Web services. With so many different SOAP implementations some interoperability issues are bound to arise.

The work of the WS-I could turn out to be crucial in this area, particularly if they manage to come up with a set of tools for testing interoperability.

For the scenario it is hard to decide whether or not the Web services provided by the university are interoperable with other external Web services, at least as long as these are implemented using different technologies.

### **Ease of use**

The J2EE platform's ease of use depends on the implementation of the platform, whereas Sun ONE and .NET offer ease of use on two different levels. Sun ONE can be said to offer ease of use on a higher level than .NET through the Sun ONE Infrastructure Products that ships with user-friendly interface and offers easy and tight integration.

.NET however, offers ease of use on a lower, more functional level, through the pre-defined functionality that can be inherited from the .NET Framework class libraries, and the Web and Windows Forms application models.

The developers of the university scenario would probably prefer the .NET Framework with respect to ease of use, since it allows them to quickly develop the various Web services without having to concern themselves with all the details of SOAP, WSDL and smaller pieces of functionality.

### **Scalability**

As was pointed out, J2EE offers the best scalability since .NET requires a larger number of machines than a similar J2EE deployment to maintain performance when the number of additional machines grows.

For the scenario, however, this will probably have little effect. The Web services that will be implemented are not large applications that require much space on the servers and the numbers of databases that are required to store user data will not be many. It is therefore not likely that this scenario will reach the number of additional machines where the difference in scalability is noticeable.

### **Tool support**

Even though the J2EE platform offer the largest selection of tools, Visual Studio .NET offers the most complete solution for developing Web services. This does not imply that one should automatically choose Visual Studio .NET for developing Web services.

First of all, the .NET framework does not support Java. Even though Microsoft's newest releases on the programming language front, C# and J#, have many similarities to Java, it still is different. The Java programming language has gone through several stages of evolution, mainly in the Java Virtual Machine, and is now the preferred language of many developers.

It is therefore fair to assume that some programmers would abandon the .NET framework simply because of its lack of support for Java. Businesses with a high degree of Java competence would also probably look somewhere else than the .NET framework. For Java and the J2EE platform, the benefit is that there are such a wide variety of tools to choose from. One just needs to find the appropriate tool for the task at hand. For the developers of the considerations to make concerning development tools are simply that if Java is the preferred language, then J2EE or Sun ONE should be chosen. If not, the .NET framework offers compilers that target the CLR for many different languages.

### **Billing**

With respect to billing none of technologies offer anything concrete. Instead one should consider the different revenue models that can be used to charge users of Web services. For the scenario, there are several options with regards to billing. If the university chose the transactional model, users would be charged for each time they accessed course web services, either upon entry or as periodic charge based on number of entries. This model should only be chosen for external users, since students pay a one-time registration fee to the university and the portal service should be included in this registration. Therefore the Membership or Subscription model is more suited. Students register when they start at the university and each new term they pay a subscription fee for access to courses and the portal. External users also register and pay for access to courses for a limited time. If an external user for example is a company wishing to incorporate some course web service into their Intranet solution, the Lease or License model could be suitable. Another university that offers some courses from the service providing university could also use this model. Between universities the Business partnership model could also be a solution, where two or more universities agree on one common portal for access to all courses and the incoming fees are distributed among the universities according to some agreement. The Registration model would not apply to this scenario.

## **5.4 Conclusion**

The purpose of this evaluation has been to present a comprehensive unbiased overview of the capabilities of the .NET framework, J2EE and Sun ONE. The evaluation has been conducted with respect to discovering how each technology fulfils the list of criteria, without suggesting that one technology offers more than the others. All three technologies are well suited for Web service development and deployment. Results of the evaluation, summed up in table 2, can then be used to determine which technology is best within a certain context, where some criteria are more important than others.

If the context for choosing Web service development framework is a scenario where the most important requirements are high scalability, support for numerous platforms, offers large-scale legacy systems integration and has a wide variety of development tools, results of the evaluation suggest the use of the J2EE platform. The summary in table 2 showed that the J2EE platform offers the best scalability when the number of additional machine grows and that there are no strict limitations on platform support. The J2EE Connector Architecture can be used to integrate with any EIS that implements a resource adapter.

If the scenario instead of a large selection of tools required a tightly integrated product family the Sun ONE platform is recommended, due to the Sun ONE infrastructure software. The Sun ONE platform uses a J2EE conformant service container for Web service deployment and therefore inherits many of the capabilities of the J2EE platform, as shown in table 2 in areas such as scalability, deployment, legacy system integration and code access security.

However, if the requirements lean towards a platform that offers a comprehensive and easy to use tool, the opportunities of shared context, an easy deployment scheme and a promising

development environment for mobile devices, then the Microsoft .NET platform is the best to choose. Visual Studio .NET is a tool that lets developers rapidly and easily create Web services thanks to templates, wizards and the .NET Framework class libraries. The .NET Compact Framework is a promising initiative for development for mobile devices, in addition to the already existing Mobile Internet Toolkit. With the .NET Framework developers have access to an easy deployment model, which in most cases simply involves copying the Web service to its destination machine. .NET is also the only technology in this evaluation that offers anything concrete within shared context, with the .NET Passport, as long as the Liberty Alliance has nothing concrete to show to as of yet.

With respect to security all technologies offer sufficient support, which include different kinds of mechanisms for authentication, authorization, encryption and code protection. All technologies also support the necessary protocols for Web services and claim to be interoperable with each other.

A benefit of the J2EE and .NET platforms that is also worth mentioning is the fact that one can download free software development kits for these platforms. In this way one can test some of the ability of the technology before making any major investments.

Relating the evaluation to the university portal scenario also showed that selecting the proper technology would depend on what criteria the developers consider being the most important. It is therefore at this point not clear what technology is most suitable for realising the university portal scenario, but this will be clarified in the next chapter when discussing the overall design of the scenario.



# 6 Overall design

Having investigated how the different architectures fulfil the evaluation criteria, both in general and with respect to the scenario, it is now time to turn the attention to the design of the scenario and how the technologies can be used to realise the university portal. Before looking into the specifics of the architectures it is best to first review the general design considerations.

## 6.1 General design considerations

No matter what architecture is chosen the functionality and interfaces of the different Web services remain unchanged. So does the interaction between the services. This section will look at what functionality the Web services of the scenario will expose and how, in addition to looking at the interaction patterns between them. The design is based on the scenario description in chapter 3.

### 6.1.1 Interactions

Based on the use cases of section 3.2, the service description of section 3.5 and the overall architecture of Figure 5 in section 3.6, the interactions that may take place between the services are those shown in Figure 20.

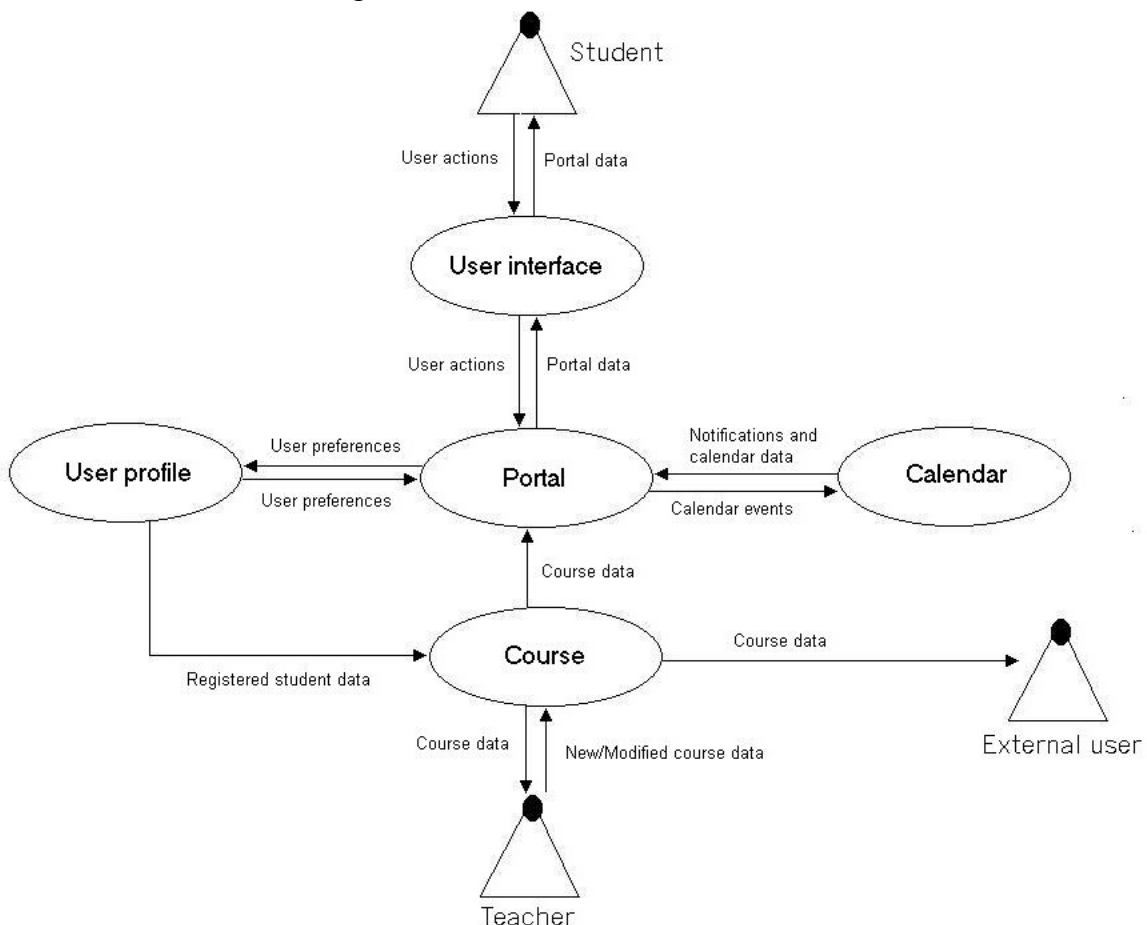


Figure 20. High-level model of the interactions in the university scenario.

The arrows in the figure show in which directions information/data is sent with a short description of what kind of data is sent next to the arrow.

### 6.1.2 Course Web service

As shown in Figure 20 the Course Web service sends its data to the Portal, the teacher and any external users. The data that is sent is general information about the course, lecture times, exam dates, exercises, curriculum, progress plan, messages related to the course, and information about students registered for the course. The Web service must therefore implement methods for retrieving this information.

Data for the Course Web service comes from teachers and the User Profile Web service. The teacher adds, removes and updates (which is a combination of remove and add) information related to the course he/she is teaching. The User Profile Web service sends information regarding registered students and also authentication data for external users. It is thus necessary to implement and/or call methods that retrieve this data.

The “implement and/or call” phrase means that there are two ways to get data; either through a method call to the source of information or by implementing a method that the information source uses to send information using the parameters of the method. For example, the Course Web service calls a method in the User Profile Web service, but implements methods that the teacher uses to manipulate course information.

Below is Table 3 that summarizes the methods that must be implemented for the Course Web service interface, together with the parameters for each method and a short description on what the methods do. Methods that operate behind the scenes are not shown here.

Method name	Parameters	Purpose
GetCourseDescription	None	Returns a description of the course
AddCourseDescription	Description text	Adds a description of the course
DeleteCourseDescription	None	Removes the description of the course
GetLecture	None	Returns the day, time and place of lectures
AddLecture	Day, time, place	Adds the day, time and place of lecture
DeleteLecture	Day	Removes the lecture for given day
GetExercise	None or id	Returns a description of the most recent or a specified exercise and deadline for delivery
AddExercise	Exercise id	Adds an exercise with the date for delivery
DeleteExercise	Exercise id	Remove specified exercise
GetCurriculum	None	Returns the curriculum
AddCurriculum	Plain text	Adds a curriculum for the course
RemoveCurriculum	None	Removes curriculum plan
GetProgressPlan	None	Returns the progress plan
AddProgressPlan	Plain text	Adds a progress plan for the course
RemoveProgressPlan	None	Removes the progress plan
GetMessage / GetMessages	None or message id	Returns the most recent or all messages related to the course
AddMessage	Message text and id	Adds a message
RemoveMessage	Message id	Removes a message
GetStudentInfo	None	Returns the students that are registered for the course

**Table 3. Overview of the methods for the Course Web service.**

The course description is plain text that describes the course and there is only one description for the course so there is no need for any identification parameters.

A course can have lectures on more than one day but only one per day, so the *GetLecture* method returns a list of all lectures for the course during the week. This can be implemented

by having a *lecture* datatype and return a table of such *lecture* elements. The *Add* and *Delete* methods for lectures must have a day parameter to identify the appropriate lecture.

A course often has more than one exercise so the *GetExercise* method with no parameter returns the most recent exercise. A student should also have access to older exercises and these can be accessed by specifying the exercise number as a parameter. This parameter is also used for adding and removing exercises. Exercises are described in plain text together with a date for delivery.

The curriculum and progress plan is also plain text and there is only one for each course. Plain text is also the case for a message but it must also contain an identifier. This is used to specify particular messages. The *GetExercise* method with no parameter returns the most recent message and the parameter can be used to retrieve a specific message, whereas the *GetMessages* method returns all messages related to the course.

The *GetStudentInfo* calls methods from the User Profile Web Service to retrieve a list of all students that are registered for the course.

Methods that are concerned with passing data to the course without any data return should implement a status return, telling the requester whether or not the method was successfully executed. As an example if a teacher wants to remove a lecture from the course he/she has to send as parameter the day on which the lecture is held. If the teacher then accidentally specifies a day where no lecture is held a return message should be sent to notify the teacher of this. A message should also be sent to teacher if the operation was executed successfully. Below is a short extract of how the WSDL file for the Course Web service looks. It focuses on the part of the WSDL file that concerns lectures. The structure for the remaining parts of the course would look approximately the same.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="Course"
  targetNamespace="http://www.university.com/course-interface"
  xmlns="http://www.schemas.xmlsoap.org/wsdl"
  xmlns:soap="http://www.schemas.xmlsoap.org/wsdl/soap"
  xmlns:tns="http://www.university.com/course"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <types>
    <xsd:schema targetNamespace="http://www.university.com/course"
      xmlns="http://www.w3.org/1999/XMLSchema/">
      <xsd:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        <xsd:complexType name="Lecture">
          <xsd:element name="Day" type="xsd:String" />
          <xsd:element name="Time" type="xsd:time" />
          <xsd:element name="Place" type="xsd:String" />
        </xsd:complexType>

        <xsd:complexType name="ArrayOfLecture">
          <xsd:complexContent>
            <xsd:restriction base="soapenc:Array">
              <attribute ref="soapenc:arrayType"
                wsdl:arrayType="tns:Lecture[]" />
            </xsd:restriction>
          </xsd:complexContent>
        </xsd:complexType>
      </xsd:schema>
    </types>
    ...
    <message name="SendLectures">
      <part name="Lectures" type="tns:ArrayOfLectures">
    </message>
    <message name="NewLecture">
      <part name="Lecture" type="tns:Lecture">
```

```

</message>
<message name="TargetLecture">
  <part name="Day" type="xsd:String">
</message>
<message name="OperationStatus">
  <part name="status" type="xsd:String">
</message>
...
<portType name="Course_port">
  <operation name="GetLecture">
    <output message="SendLectures">
  </operation>
  <operation name="AddLecture">
    <input message="NewLecture">
    <output message="OperationStatus">
  </operation>
  <operation name="DeleteLecture">
    <input message="TargetLecture">
    <output message="OperationStatus">
  </operation>
...
</portType>
...
</definitions>

```

This example only shows the abstract definitions of the WSDL file. Since there is no concrete implementation to bind this to, the binding details are left out (it would look just like the binding example shown in section 4.1.3).

In the example the lecture is built as a complex data type and an array for sending multiple lectures is also defined. Next come the messages that act as the parameters and responses for the methods. These messages are then associated with corresponding operations when defining the `<portType>`.

The following example will show the SOAP message structure of the messages that were presented in the WSDL example. The example shows an incoming and an outgoing message.

```

<soap-env:Envelope
  xmlns:soap-env="http://www.w3.org/2001/06/soap-envelope"
  soap-env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"/>
<soap-env:Header>
  ...
</soap-env:Header>
<soap-env:Body>
  <ns:NewLecture xmlns:ns="urn:Course">
    <Lecture>
      <Day>Monday</Day>
      <Time>10:00:00</Time>
      <Place>Room 452</Place>
    </Lecture>
  </ns:NewLecture>
</soap-env:Body>
</soap-env:Envelope>

```

The above example shows the appearance of the SOAP message for adding a new lecture to the course. The message contains the day, time and place for the new lecture. The XML Schema for the day, time and place elements reside in the "urn:Course" namespace. The following example looks at the message for sending the array of lectures from the course.

```

<soap-env:Envelope
  xmlns:soap-env="http://www.w3.org/2001/06/soap-envelope"

```

```

soap-env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"/>
<soap-env:Header>
  ...
</soap-env:Header>
<soap-env:Body>
  <ns:SendLectures xmlns:ns="urn:Course">
    <Lectures soap-enc:arrayType="ns:Lecture[2]">
      <Lecture>
        <Day>Tuesday</Day>
        <Time>11:00:00</Time>
        <Place>Room 553</Place>
      </Lecture>
      <Lecture>
        <Day>Friday</Day>
        <Time>09:00:00</Time>
        <Place>Room B2</Place>
      </Lecture>
    </Lectures>
  </ns:SendLecture>
</soap-env:Body>
</soap-env:Envelope>

```

This SOAP message sends the lectures that are held for the specific course, contained in a Lectures array that hold two Lecture elements.

### 6.1.3 Portal Web service

The Portal Web service receives data from the Course, User Profile, User Interface and Calendar Web services and send data to the User Profile, User Interface and Calendar Web services.

The communication with the Course Web services involves presenting all students with the general descriptions of the courses they request, and presenting the more detailed information to the students that are registered for the course in question. The Portal uses the various *Get* methods from the Course Web service to retrieve this information. Also, the Portal must extend these *Get* methods in order for the student to be able to request this information instead of having it presented all at once.

The relationship between the Portal Web service and the User Profile Web service is concerned with authentication of users, and storage and retrieval of student data and preferences. The Portal uses the User Profile Web service to authenticate students and must maintain a user session until the students logs off. The Web service should therefore have methods for logging on and off. This will also regulate what other Portal methods the student can use. As soon as a student is authenticated the student's preferences must be fetched in order to set up the Portal appropriately, and when the student logs off any changes to the preferences must be saved. The Portal should also be able save preferences during the session, for example as the student goes on adding and removing courses. All of this happens behind the scenes, no methods need to be exposed at the Portal's interface.

The Portal Web service must take the information it receives from courses (lectures, exam dates, exercise deadlines) and send it to the Calendar Web service, and also let the user add his/hers own dates. This process may involve transforming data into an appropriate format for the messages to the calendar. In the other direction the Portal will receive notifications from the calendar upon important events related to the logged on user, and it must let the user access the calendar to find out about coming events.

The Portal interacts with the students through the User Interface Web service. This means that students request data from the Portal via the User Interface and the Portal sends data to the students via the same Web service.

Table 4 contains an overview of the methods that the Portal Web service needs to implement at the service interface.

Method name	Parameters	Purpose
SearchCourse	Name or code	Search for a course either by name or code
BrowseCourses	None	Returns a list of all available courses
AddCourse	Course code	Lets the student register for a specific course
DeleteCourse	Course code	Removes a specific course from the portal
GetCourseDescription	Course code	Returns the description for the specified course
GetLecture	Course code	Returns the day, time and place for lectures in the specified course
GetExercise	Course code (+exercise id)	Returns the most recent exercise for the specified course or a particular exercise
GetCurriculum	Course code	Returns the curriculum for the specified course
GetProgressPlan	Course code	Returns the progress plan for the specified course
GetExamDate	Course code	Returns the exam date for the specified course
GetMessages	Course code	Returns messages from the specified course
GetMessageBoard	None	Returns messages from all the courses for which the student is registered.
GetCalendarData	None	Returns the contents of the student's calendar
AddCalendarEvent	Event	Adds an event to the calendar
DeleteCalendarEvent	Event	Removes an event from the calendar
SavePreferences	None	Saves the student's preferences

**Table 4. Overview of the methods for the Portal Web service**

The methods related to courses let the student browse available courses at the university, or search for a specific course identified by name or code. The student can then add a course to his/hers Portal, at the same time registering for the course. The Portal must however check with the student's user profile to see if the course is allowed according to the study plan. If a student has a change of heart he/she can remove the course from the Portal.

The student can access information related to specific courses via the various *Get* methods for the course description, lecture, exercise, curriculum, progress plan and messages. In addition comes the *GetMessageBoard* method, which displays all messages from the course that the student is registered for.

The Portal will automatically add to the calendar lectures, exercise deadlines and exam dates from the courses for which the student is registered. Furthermore the student can add and remove personal events to the calendar.

The *SavePreferences* method can be used to save any changes made to the student's preferences during the Portal session. These preferences do not just include the courses the student has selected but also details such as read/unread messages.

The structure and appearance of the WSDL file and SOAP messages will resemble those for the course Web service.

#### **6.1.4 User Interface Web service**

The User Interface communicates with the Portal Web service in order to present information to students and let the user make requests to the Portal and modify the set-up of it. The presentation of Portal data must be adapted according to the device the student uses for access.

The functionality of the User Interface Web service involves extending the functionality of the Portal and generating appropriate output for client devices, and receiving input from students that must be translated to corresponding methods for the Portal.

The methods at the service interface should be approximately the same as those for the Portal Web service in Table 4.

### 6.1.5 User Profile Web service

The User Profile Web service interoperates with the Portal Web service to authenticate students and manage user preferences. It is also responsible for delivering lists of students that are registered for the requesting course.

In the relationship with the Portal the user profile must provide a means of authenticating students. This is done with a student specific user name and password that is checked against the student directory. Having authenticated the student the preferences for this student must be sent to the Portal for configuration of the Portal. The user profile must also be able to receive preferences in order to update these.

As mentioned for the Course Web service the teacher might be interested in finding out who is registered for the course. The Web service therefore had a method called *GetStudentInfo* that returned a list of registered students. This method must be able to access the User Profile Web service to receive the list of students, so a method for returning registered students is required.

Table 5 below summarises the methods that are required at the User Profile Web service interface.

Method name	Parameters	Purpose
AuthenticateUser	Username and password	Authentication of students
GetUserPreferences	StudentID	Returns the user preferences for a given student
SaveUserPreferences	Preferences, StudentID	Stores the user preferences for a given student
GetRegisteredStudents	CourseID	Returns the students registered for a given course

**Table 5. Overview of the methods for the User Profile Web service.**

The User Profile Web service provides all authentication logic. Upon entry to the Portal the student is then redirected to the User Profile where the student must supply a username and a password for authentication.

Dealing with user preferences involves database access. Based on the student identification (either username or a specific student ID number) the User Profile searches the database for the preferences for that particular student. When saving the preferences the User Profile receives the preferences in a tabular form, so the Web service must extract the elements of the table and store any changes in the database.

The *GetRegisteredStudents* method involves a simple database query which returns a list of all students registered for course specified in the parameter of the method.

### 6.1.6 Calendar Web Service

The Calendar Web service is from an external service provider and communicates with the Portal Web service. The Calendar must implement methods for inserting and retrieving calendar events and also methods for notifying users upon special events. The Portal must use the WSDL file from the Calendar to determine the data format for each of the Calendar's methods.

## 6.2 Microsoft .NET

To develop the scenario for the .NET platform requires the use of ASP .NET and managed components. With the ASP .NET application model one can develop Web Forms and Web

services. Developing Web services involves the use of the `System.Web.Services` namespaces. These namespaces consist of classes from the .NET framework class library and are the classes that enable one to create XML Web services.

Using Visual Studio .NET will help hide the details of ASP .NET and developers can develop Web services using the templates and wizards that come with the tool. There one can develop a Web service by creating an ASP .NET Web service project.

The appearance of the portal can be developed using Web Forms. With Web Forms one only needs to create one user interface, since the Web Forms render the output using the markup language that is appropriate for the client device. It would therefore not be necessary to have the User Interface Web service for rendering user interface that was suggested for the scenario.

This section will look at the basics of developing a Course Web service using ASP .NET.

Developing XML Web services with ASP .NET begins with the following three steps:

1. Create a file with an `.asmx` file name extension.
2. Within the file, declare the XML Web service using a directive.
3. Define the Web service methods that constitute the functionality of the Web service.

To the declare a Web service one uses the `@ WebService` directive at the top of the file with the `.asmx` file extension. The following code example illustrates this. The example uses the .NET programming language C#.

```
<%@ WebService Language="C#" Class="Course" %>
using System.Web.Services;
public class Course{
...
}
```

An additional feature when declaring a Web service is the optional `WebService` attribute. This attribute lets the developer set the default XML namespace for the Web service, as shown in the following code example.

```
<%@ WebService Language="C#" Class="Course" %>
using System.Web.Services;

[WebService(Namespace="http://www.university.com/course")]
public class Course{
...
}
```

It is recommended that one modify this default namespace, which is originally set to `"http://tempuri.org"`.

The next step is defining the Web service methods. Applying the `WebMethod` attribute to public methods does this. The following example shows how the `Course` method for setting and returning the course description could be implemented.

```
<%@ WebService Language="C#" Class="Course" %>
using System;
using System.Web.Services;

[WebService(Namespace="http://www.university.com/course")]
public class Course{
```

```

...

[ WebMethod(Description="set the course description")]
public void AddCourseDescription(string newDesc) {
    courseDescription.Copy(newDesc);
}

[ WebMethod(Description="return the course description")]
public string GetCourseDescription() {
    return courseDescription;
}

private string courseDescription;

...
}

```

These simple code examples show that it is fairly simple to expose new or existing code as Web services, using the `@WebService` directive and the `WebService` and `WebMethod` attributes.

### 6.3 Java 2 Enterprise Edition

Developing the scenario on the J2EE platform involves using the JAX\* API's presented in section 4.2.2. These APIs let programmers develop the Web services using only the Java programming language; no SOAP or WSDL mark-up is required.

The APIs come in two categories. JAXP processes XML documents using various parsers, and JAXB maps XML elements to Java classes. These APIs can therefore be said to be document-oriented. The other APIs are procedure-oriented: JAXM and JAX-RPC is used for SOAP messaging, and JAXR accesses business registries.

This section will look at how to create a course Web service with these APIs, focusing mainly on message creation with JAXM. The reason for choosing JAXM over JAX-RPC is that JAXM has some features not included with JAX-RPC, such as asynchronous messaging, routing of messages to more than one party, and reliable messaging with features such as guaranteed delivery.

To create a message with JAXM one uses a *MessageFactory* object. First of all an instance of the object must be created and instantiated, as illustrated with the following code snippet.

```
MessageFactory messagefactory = MessageFactory.newInstance();
```

The *MessageFactory* object can then be used to create a basic *SOAPMessage* object.

```
SOAPMessage message = messagefactory.createMessage();
```

All messages created by the *MessageFactory* are basic SOAP messages, meaning that they have no pre-defined headers. They do however contain the required elements *SOAPPart*, *SOAPEnvelope*, and *SOAPBody*, in addition to the optional *SOAPHeader* element.

These elements can be accessed to provide content to the SOAP message. The following code example shows how to access the different elements, with the *SOAPMessage* object as the starting point.

```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
```

```

SOAPHeader header = envelope.getHeader();
SOAPBody body = envelope.getBody();

```

The message object is used to access the *SOAPPart* which again can be used to access the *SOAPEnvelope*. Having retrieved the envelope one can access both the *SOAPHeader* and the *SOAPBody*.

Having retrieved the different elements of the SOAP message it is time add the content. The following code example shows how to create the SOAP message *NewLecture* as it was presented in section 6.1.2. It will assume that the various elements of the SOAP message described above have been retrieved.

```

Name bodyName = envelope.createName("NewLecture", ns,
"urn:Course");
SOAPBodyElement newLecture = body.addBodyElement(bodyName);

Name lecture_name = envelope.createName("Lecture");
Name day_name = envelope.createName("Day");
Name time_name = envelope.createName("Time");
Name place_name = envelope.createName("Place");

SOAPElement lecture = newLecture.addChildElement(lecture_name);
SOAPElement day = lecture.addChildElement(day_name);
SOAPElement time = lecture.addChildElement(time_name);
SOAPElement place = lecture.addChildElement(place_name);

day.addTextNode("Monday");
time.addTextNode("10:00:00");
place.addTextNode("Room 452");

```

The Java code presented here is what is required to create the SOAP message for adding a new lecture to the course. First the body is created with a local name, a prefix for the namespace being used and the URI for the namespace. To add content to the message one must have a *SOAPBodyElement* to hold the content. The actual content is added using *SOAPElement* objects. When adding elements to the message the *Name* object is required in order to identify the element.

The examples show that using the JAX\* APIs involves a bit of extra coding compared to that of .NET, but it hides all the XML details of SOAP and WSDL and lets the developer express all functionality using the Java programming language.

## 6.4 Sun Open Net Environment

On the Sun ONE platform the Sun ONE Portal Server seems ideal for realising the university portal scenario. The Sun ONE Portal Server offers aggregation, presentation, personalization and security features that transfer nicely to the features required on the university course web portal.

The Sun ONE Portal Server can be used as a single point of access to user-facing Web services, such as the course Web services in the scenario. It can also aggregate Web applications such as a calendar that also was part of the requirements for the scenario. In addition to aggregation and presentation of Web service for end users, the portal can also do the same for other Web services, thereby acting as a Web service on its own.

The Sun ONE Portal Server has personalization features that let both administrators and end users customize the portal to their needs. These features include content and desktop layout, access control policies and management of profile data. For the scenario this means that the

portal can present the user with exactly the course Web services that has been registered for and restrict access to other courses.

The portal handles user authentication with single sign on and access control policies. Every authenticated user on the portal is assigned a unique role in the domain. The role defines most aspects of what the user will experience in content as well as access control. Every application and Web service within the Sun ONE Portal Server defines a set of privileges in its application profile. These privileges define the policy for each specific role. This means that course Web services define privileges for the different types of users (e.g. registered/unregistered students, external users) and restricts access accordingly.

The majority of the Sun ONE Portal Server's platform, user and application specific data are stored in the portal server's profile database, which is the Netscape LDAP Directory Server. The Sun ONE Portal Server does not support using customer's existing LDAP directory to store portal server data. Nevertheless, the Sun ONE Portal Server supports aliasing attributes from the portal server profile to attributes in the customer's LDAP directory. For example, the university may keep IMAP passwords in their LDAP directory server. In this case the Sun ONE Portal Server profile can be pointed to the university LDAP directory to retrieve those passwords.

The fact that the Sun ONE Portal Server handles user authentication and manages profile data, makes the User Profile Web service for the scenario redundant, since the purpose of this Web service was to provide logon functionality and storage for user preferences.

A portal that is part of the Sun ONE platform, such as the Sun ONE Portal Server, must have a Java class file called Provider. The Provider is responsible for converting the content that the portal presents into to proper format for the receiving channel. The contents are delivered to client devices in one or more markup languages, such as HTML, cHTML, WML or XML. Providers for the portal can be developed using the Content Provider API or one can use one of the several pre-built Providers that are included with the portal.

The Provider has important implications for the scenario, as it is not necessary to create a user interface web service that provides user interfaces based on what device is being used to access the portal.

The Sun ONE Portal Server therefore seems to cover most aspects of what is required for the scenario: authentication of users, managing user preferences and displaying appropriate markup for accessing devices and the ability to configure the portal, both with respect to appearance and access control policies. The course Web service can be deployed on either the Sun ONE Portal Server or the Sun ONE Application Server, which can be used in conjunction with the portal server.

## **6.5 Conclusion**

Having looked at the design of the scenario and investigated how the various technologies solve some of the design issues, some general conclusions can be made regarding the use of J2EE, Sun ONE and .NET for developing the university portal.

The Sun ONE platform definitely seems to offer the most tempting solution, with the Sun ONE Portal offering all of the functionality the university portal should have. It also reduces development effort since the ones administrating the portal can easily customize it.

In addition, it removes the need to develop separate User Interface and User Profile Web services, since these kinds of services are included in the Sun ONE Portal.

With respect to developing the Course Web services the choice is not so obvious. Having already invested in the Sun ONE platform one could choose to stick with the Sun ONE Studio IDE, which is closely integrated with the rest of the Sun ONE infrastructure software.

As has been pointed out many times when discussing Web services it does not matter what programming language one uses to implement the Web service as long as it can communicate with other Web services using SOAP and WSDL. The Sun ONE platform offers interoperability with .NET Web services, and J2EE Web services can be deployed directly on the Sun ONE platform since it uses a J2EE compliant service container for deployment of Web services.

Choosing between J2EE and the Sun ONE Studio is then really a matter of preference, at least for Java developers. Developers can then use the tool they feel most comfortable with. Sun ONE Studio also offers IDEs for C++ and Fortran but those developers can also opt for the .NET platform. The .NET platform can in addition be used by a wide variety of other languages provided that these have a compiler that can produce managed code for the CLR. A benefit of using .NET is that it allows developers to easily turn existing code into Web services using the *WebService* and *WebMethod* attributes presented in section 6.2. This would be a good option to have if some code distributing course information already existed.

The important thing to remember, though, is that all the three technologies in question can be used to develop the scenario. There are many factors involved in choosing platform that are beyond the scope of this discussion, such as the time and cost of porting from the currently used platform or the demands of existing and future clients. A careful and thorough analysis should therefore be conducted before making the final choice of platform. Based on the analysis made here the recommendation is to use the Sun ONE Portal server to realise the university portal together and also gain the benefits of the user authentication and interface benefits that follow with the portal server. For the course Web services the votes are more or less tied, with a slight predominance on the .NET platform, which hides most of the details of SOAP and WSDL and also offer the rich set of reusable classes together with the ASP .NET programming model.

# 7 Conclusions and further work

This report will be finished with a summary of the points made throughout, and also a look at what can be done for further work.

## 7.1 Overall conclusions

In chapter 2 an introduction to the Web service concept was presented. Having looked at the different definitions of the term Web service, the following definition was suggested: *A Web Service is functionality, available through an open Internet standard, giving programmable access to the service, independent of its implementation.* It was also concluded that Web services differ from the traditional component approach in two areas: coupling and granularity. The presentation also showed that Web services have a wide variety of application areas, ranging from large-scale business integration service to small-scale user services such as finding the current temperature in Spain.

Chapter 3 concentrated on the development of a scenario for further reference. A familiar scenario was chosen: a university providing information about courses to students through a portal. It was shown that a Web service approach would be suitable, as it would enable the collection of always up-to-date information through a single point of access. The Web service approach would also open up for integration with other universities and various external users. A Web service solution also ensures interoperability with any future technologies the university may choose to adopt and also allows easy management of courses (e.g. adding new courses).

Chapter 4 looked at the enabling technologies for Web services: XML, SOAP, WSDL and UDDI. Three platforms for Web service development were presented: Microsoft's .NET platform, Sun's Java2 Enterprise Edition and the Sun Open Net Environment. These technologies were the subjects for the coming evaluation.

Chapter 5 contained the evaluation of the abovementioned platforms, based on a set of evaluation criteria. The first phase of evaluation consisted of describing how the various technologies fulfilled the criteria in general, whereas the second phase related the evaluation to the scenario in chapter 3. The evaluation resulted in an unbiased presentation of platform capabilities, and forms the basis for selecting the most suitable technology based on requirements in a specific context where some criteria are more important than others.

The J2EE and Sun ONE platform seem to offer much of the same in most of the aspects of the evaluation, particularly within scalability, legacy system integration, platform support, deployment, code access security, and most likely within shared context. It is also within the first two abovementioned criteria that these platforms seem to offer more than the .NET Framework. Scalability and legacy system integration are requirements that often have a high priority when it comes to scenarios that include large-scale business integration and business-to-business collaboration. These are also the areas where the J2EE implementations and Sun ONE are most often used, and the target customers for these platforms are mainly enterprises. Not to say that .NET does not fit for enterprise solutions, but the framework does have a wider target audience. This is reflected in the ease of use offered with the .NET Framework class libraries, the Windows and Web Forms programming models and the templates and wizards found in Visual Studio .NET, and also the simple deployment model it offers. These properties aim at letting anyone who has some knowledge in programming develop applications and Web services rapidly and easily.

Chapter 6 suggested an overall design of the university portal scenario and showed examples of how the different technologies could be used to realise the scenario. The discussion resulted in a suggestion that the Sun ONE Portal Server should be used for realising the portal

as it provides all the required portal functionality in addition to facilities for managing user profiles and user interfaces. It was also suggested that the choice of technology for implementing the course Web services was a matter of preference but that the .NET platform probably would be the best choice. This is particularly true when it comes to porting existing code to Web services since adding a few keywords to existing code can do this. .NET is also more suited for developing smaller Web services of the kind such a course Web service is than the more enterprise ready J2EE and Sun ONE platforms.

Interoperability between the Sun ONE Portal Server and the .NET Course Web service could turn out to be an issue even though both technologies claim to be interoperable with each other. In such a case testing is required and then the necessary adoptions must be done to the interfaces and message formats.

Overall this report has given an introduction to the Web service concept and technologies surrounding it. A neutral evaluation of Web service platforms for development and deployment has been presented, which can be used as a starting point for those interested in finding out more about Web service technology. The evaluation framework can also be used for evaluation of other Web service technologies, in addition to the platforms evaluated in this report, such as various development frameworks available from a wide variety of vendors (see section 7.3).

## **7.2 Limitations**

The evaluation in this report has been based solely on a theoretical study of the different technologies' capabilities. In order to assess all parts of the evaluation and find out about any other properties that might be of key interest to the final result, the comprehensive literature study approach is the best.

Nevertheless it is a limitation to the evaluation not having tested any of the subject technologies. Testing the different platforms could have given a stronger evaluation and any assumptions made during evaluation could have been confirmed. Many of the evaluated criteria could have been expanded to include test results. Testing would also enable additional criteria, for example performance and installation.

It would also have been beneficial to develop some kind of prototype for the scenario in order to demonstrate and confirm the overall design. In this way any missing or inadequate requirements could be discovered and the description of the scenario could be improved.

Limitations are mainly due to the fact that the evaluation has been conducted alone. Had two or more performed the work, this would probably have freed time to perform testing of the technologies and the development of a small prototype.

## **7.3 Suggestions for further work**

The following is suggested for further work:

- **Testing the subject technologies:** As mentioned in the previous section on limitations, there have been no concrete tests of the subject technologies. Testing these technologies would be a natural next step in the process of evaluation and should be conducted on as many of the criteria as possible. A prototype for testing and demonstration would have to be developed, such as one of the Web services in the scenario. Then various tests would have to be set up. Examples of tests include various security tests, a mobile client for mobility testing, interoperability testing and scalability tests.

One could also test and evaluate new criteria, for example performance related criteria such as response time and memory usage.

- **Developing the scenario:** A prototype of the university portal scenario should be implemented using the three technologies presented, one prototype for each platform. This would give an even more solid foundation for evaluating the technologies, at least within the context of the university portal scenario and any thinkable scenarios that might resemble the scenario used here.  
This is best done in an experimental setting, where three groups are put together, one for each platform. If only one group had developed all three prototypes one would most likely experience learning effects (the group becoming more and more familiar with Web service development) that could influence the final result.  
It is also important that the scenario is even further described in detail to ensure that ambiguity is avoided as far as possible, resulting in more or less identical prototype portals from the various technologies.
- **Evaluate more technologies:** There exist a wide variety of Web service development frameworks, which are mainly implementations of the J2EE specification. This report has evaluated J2EE in a general manner, independent of the numerous J2EE implementations that exist. Some Web service development technologies that could be evaluated are IBM's WebSphere software platform<sup>21</sup>, BEA's WebLogic platform<sup>22</sup>, the iNsign Web service development framework from Iopsis<sup>23</sup>, or CapeConnect and CapeStudio from Cape Clear Software<sup>24</sup>.

---

<sup>21</sup> <http://www-3.ibm.com/software/info1/websphere/index.jsp>

<sup>22</sup> <http://www.bea.com/products/weblogic/platform/index.shtml>

<sup>23</sup> <http://www.iopsis.com/products/iNsign.htm>

<sup>24</sup> <http://capeclear.com/>



# Appendix A: Bibliography

- [1] Karl Gottschalk. The next stage of evolution for e-business. September 2000.  
<http://www-106.ibm.com/developerworks/webservices/library/w-ovr/>
- [2] Graham Glass. Applying Web services to applications. November 2000  
<http://www-106.ibm.com/developerworks/webservices/library/ws-peer1.html>
- [3] Mary Kirtland. A platform for Web Services. January 2001  
[http://msdn.microsoft.com/library/en-us/dnwebsrv/html/websvcs\\_platform.asp](http://msdn.microsoft.com/library/en-us/dnwebsrv/html/websvcs_platform.asp)
- [4] Diana Reichardt. A Field Guide to Services on Demand and Sun™ ONE. September 2001. <http://www.sun.com/software/sunone>
- [5] Greg Flurry. January 2001. Applying Web services to the application service provider environment. <http://www-106.ibm.com/developerworks/library/ws-wsasp/>
- [6] Elliotte Rusty Harold, W.Scott Means. XML in a nutshell, A Desktop Quick Reference. O'Reilly and Associates. January 2001.
- [7] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. <http://www.w3.org/TR/SOAP/>
- [8] James Snell, Doug Tidwell, Pavel Kulchenko. Programming Web Services with SOAP. O'Reilly & Associates January 2002.
- [9] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001.  
<http://www.w3.org/TR/wsdl>
- [10] Roger Wolter. Simply Soap. 15 October 2001.  
<http://msdn.microsoft.com/library/en-us/dnexxml/html/Xml10152001.asp?frame=true>
- [11] UDDI Technical White Paper. 6 September 2000. <http://www.uddi.org>
- [12] Roger Wolter. XML Web Services Basics. December 2001.  
<http://msdn.microsoft.com/library/en-us/dnwebsrv/html/webservbasics.asp?frame=true>
- [13] A Guide to Web Services. <http://www.cbdiforum.com>
- [14] Bilal Siddiqui. Introduction to Web Services with WSDL. November 2001.  
<http://www-106.ibm.com/developerworks/library/ws-intwsdl/>
- [15] Brett McLaughlin. Excerpt from “Java and XML”, Chapter 12: SOAP. O'Reilly & Associates September 2001.  
[http://www.onjava.com/lpt/a//onjava/excerpt/java\\_xml\\_2\\_ch2/index.html](http://www.onjava.com/lpt/a//onjava/excerpt/java_xml_2_ch2/index.html)
- [16] Bertrand Meyer. .NET Is Coming. Computer, August 2001.

- [17] .NET Framework Developer's Guide. Overview of the .NET Framework.  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp>
- [18] Aaron Weiss. Microsoft's .NET: Platform in the Clouds. NetWorker, Volume 5, Issue 4. December 2001.
- [19] Jeffrey Richter. Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web. MSDN Magazine, September 2000.  
<http://msdn.microsoft.com/msdnmag/issues/0900/framework/print.asp>
- [20] Jeffrey Richter. Part 2: Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web. MSDN Magazine, October 2000.  
<http://msdn.microsoft.com/msdnmag/issues/0900/Framework/Framework.asp>
- [21] Mary Kirtland. The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework. MSDN Magazine, September 2000.  
<http://msdn.microsoft.com/msdnmag/issues/0900/WebPlatform/WebPlatform.asp>
- [22] Sun Microsystems. Getting Started With Services on Demand and the Sun™ Open Net Environment – Sun ONE. Executive Overview. <http://www.sun.com/sunone>
- [23] Anne Thomas. Java™ 2 Platform, Enterprise Edition, Ensuring Consistency, Portability, and Interoperability. Patricia Seybold Group, June 1999.
- [24] Sun Microsystems. Simplified Guide to the Java™ 2 Platform, Enterprise Edition. September 1999.
- [25] James Kao. Developer's Guide to Building XML-based Web Services with the Java 2 Platform, Enterprise Edition (J2EE). The Middleware Company, June 2001.
- [26] Chad Vawter, Ed Roman. J2EE vs. Microsoft .NET, A comparison of building XML-based Web Services. The Middleware Company, June 2001.
- [27] Sun Microsystems. Better by Design – The Solaris™ Operating Environment. October 1998.
- [28] Foundstone Inc. Security in the Microsoft .NET Framework, An Analysis by Foundstone Inc. and CORE Security Technologies.  
<http://www.foundstone.com/company/dotnet.html>
- [29] Microsoft .NET Passport Technical Overview. September 2001.  
<http://www.microsoft.com/my services/passport/passport.asp>
- [30] Sun Microsystems. Sun ONE Architecture Guide.  
<http://www.sun.com/software/sunone/docs/arch/index.html>
- [31] James Gosling, Henry McGilton. The Java™ Language Environment, A White Paper. Sun Microsystems, May 1996.

- [32] Sun Microsystems. Designing Wireless Enterprise Applications Using Java™ Technology, A Java BluePrints for Wireless White Paper. January 2002 (Revision 2)
- [33] Sun Microsystems. JNDI, Java™ Naming & Directory Interface™, Executive Summary. <http://java.sun.com/products/jndi>
- [34] Sun Microsystems. The JDBC™ API Universal Data Access for the Enterprise. <http://java.sun.com/products/jdbc>
- [35] WS-I. Web Services Profiles, An Introduction. February 2002. <http://www.ws-i.org/Documents.aspx>
- [36] Bill Shannon. Java™ 2 Platform Enterprise Edition Specification, v1.3. Sun Microsystems, June 2001. <http://java.sun.com/j2ee/download.html>
- [37] Microsoft Corporation. Visual Studio .NET Launch Reviewers Guide. Microsoft Corporation, 2002. <http://msdn.microsoft.com/vstudio/productinfo/evalguide.asp>
- [38] Dan Gisolfi. Web services architect, Part 2: Models for dynamic e-business. IBM, April 2001. <http://www-106.ibm.com/developerworks/webservices/library/ws-arc2>
- [39] Jim Knutson, Heather Kreger. Web Services for J2EE, Version 1.0. Public Draft. IBM April 2002. <http://www-3.ibm.com/software/solutions/webservices/>
- [40] S. Brodsky. XMI Open Application Interchange. IBM, March 1999.
- [41] .NET Framework Developer's Guide. Building XML Web Services Using ASP.NET. Microsoft Corporation 2001. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconbuildingaspnetwebservices.asp>
- [42] Sun Microsystems. The Java™ Web Service Tutorial. March 2002. <http://java.sun.com/webservices/docs/ea2/tutorial/index.html>



# Appendix B: Glossary

<b><u>ADO .NET</u></b>	ActiveX Data Objects .NET is the data access model for the .NET Framework. It is entirely based on XML.
<b><u>ASP .NET</u></b>	Active Server Pages .NET is the application model for Web application development in the .NET Framework, and can be used to publish both XML Web services and Web Forms.
<b><u>cHTML</u></b>	The Compact HTML is a well-defined subset of HTML, designed for small information appliances.
<b><u>CLDC</u></b>	The Connected Limited Device Configuration is the foundation of the Java runtime environment that targets small, resource-constrained devices, such as mobile phones, and together with MIDP it constitutes a complete application runtime environment for J2ME.
<b><u>Common Language Runtime</u></b>	The Common Language Runtime is the foundation of the .NET Framework, which is responsible for managing code from .NET programs at execution time. Code targeted for the common language runtime is known as managed code.
<b><u>Common Language System</u></b>	The Common Language System is used to inform compiler vendors of the minimum set of features their compilers must support to target the common language runtime.
<b><u>Common Type System</u></b>	A formal specification of how the type system is implemented by the common language runtime. The CTS defines a standard set of objects (called types) and rules for creating new types.
<b><u>DTD</u></b>	A Document Type Definition specifies constraints on the valid tags and tag sequences that can be in an XML document.
<b><u>EJB</u></b>	The Enterprise JavaBeans technology defines a model for developing and deploying reusable Java server components, and it defines a standard programming interface for Java application servers
<b><u>HTTP</u></b>	Hyper Text Transport Protocol is an Internet protocol that defines how information is exchanged between http clients and web servers.
<b><u>IETF</u></b>	The Internet Engineering Task Force is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

<b><u>J2EE</u></b>	Java 2 Enterprise Edition. Sun's J2EE is a Java platform designed for developing multi-tier services for enterprises. It has traditionally been used for building server-side deployments in the Java programming language, but now also includes support for building XML-based Web services.
<b><u>J2EE Connector Architecture</u></b>	The J2EE Connector architecture defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems.
<b><u>J2ME</u></b>	The Java 2 Micro Edition is the edition of the Java 2 platform targeted at consumer electronics and embedded devices.
<b><u>JAXB</u></b>	The Java API for XML Binding is a specification for transforming an XML document into a Java object and vice versa.
<b><u>JAXM</u></b>	The Java API for XML Messaging is a specification for interacting with XML messaging standards such as SOAP and ebXML.
<b><u>JAXP</u></b>	The Java API for XML Processing is a native Java interface to the industry standard XML parsing APIs SAX (Simple API for XML) and DOM (Document Object Model), in addition to a pluggable interface to an XSLT (XML Stylesheet Language Transformation engine).
<b><u>JAXR</u></b>	The Java API for XML Registries can be used to access registries such as UDDI.
<b><u>JAX/RPC</u></b>	The Java API for XML based RPC is used for sending and receiving method calls using XML-based protocols such as SOAP.
<b><u>JDBC</u></b>	The Java Database Connectivity API is an industry standard for database-independent connectivity between the Java programming language and a wide range of databases.
<b><u>JVM</u></b>	The Java Virtual Machine dynamically compiles/interprets Java bytecode into platform-specific instructions.
<b><u>Kerberos</u></b>	Kerberos is a network authentication protocol, designed to provide strong authentication for client/server applications by using secret-key cryptography.
<b><u>Liberty Alliance</u></b>	The Liberty Alliance Project is an organisation formed to create an open single sign-on identity solution for any device connected to the Internet.

<b><u>MIDP</u></b>	The Mobile Information Device Profile is a set of Java APIs that, together with the CLDC, provides a complete J2ME application runtime environment targeted at mobile information devices, such as mobile phones and entry level PDAs. The MIDP specification addresses issues such as user interface, persistence storage, networking, and application life cycle.
<b><u>MMIT</u></b>	Microsoft Mobile Internet Toolkit provides the technology and tools to build, deploy and maintain applications for the mobile environment.
<b><u>.NET Compact Framework</u></b>	The .NET Compact Framework is a subset of the .NET Framework for smart device development on the .NET platform.
<b><u>.NET Framework</u></b>	The Microsoft .Net Framework is a platform for building integrated service-oriented applications for the Internet.
<b><u>.NET Passport</u></b>	Microsoft .NET Passport is an online service providing a common Internet authentication across Web sites.
<b><u>OASIS</u></b>	The Organization for the Advancement of Structured Information Standards is a non-profit global consortium driving the development, convergence and adoption of e-business standards.
<b><u>PKI</u></b>	Public Key Infrastructure is a system for verifying the authenticity of each party involved in an Internet transaction. It uses digital certificates to specify key components of the user's identity.
<b><u>SAML</u></b>	The Security Assertion Markup Language is an XML-based security standard for exchanging authentication and authorization information.
<b><u>SOAP</u></b>	Simple Object Access Protocol, a lightweight protocol for exchange of information in a decentralized, distributed environment. SOAP is a standard message format for Web Services communication.
<b><u>SSL</u></b>	Secure Sockets Layer is a protocol for securing data transmission in commercial transactions on the Internet. It uses public-key cryptography to provide server authentication, data encryption and data integrity for client/server communications.
<b><u>Sun ONE</u></b>	Sun Open Net Environment is an open framework that supports the development of local applications, client/server applications, Web applications and Web services (also known as Services on Demand).

<b><u>UDDI</u></b>	Universal Description, Discovery and Integration is a standard for web service description and discovery together with a registry facility that simplifies the publishing and discovery processes. It can be seen as the “yellow pages” of Web Services.
<b><u>W3C</u></b>	World Wide Web Consortium is an organization that develops interoperable technologies, such as standards and specifications, to lead the Web to its full potential.
<b><u>Web Service</u></b>	In this report a Web Service is defined as functionality, available through an open Internet standard, giving programmable access to the service, independent of its implementation.
<b><u>WML</u></b>	Wireless Markup Language is an XML based markup language for use on WAP cell phones and other WAP-enabled mobile devices.
<b><u>WSDL</u></b>	Web Service Definition Language is an XML based schema for describing the operational information related to the Web Service, such as the interface and the endpoints. WSDL defines XML grammar describing contracts between endpoints that are to exchange messages.
<b><u>WS-I</u></b>	The Web Service Interoperability Organization is an open industry organization that aims to promote Web service interoperability across platforms, operating systems and programming languages.
<b><u>XACML</u></b>	The eXtensible Access Control Markup Language is an XML specification for expressing policies for information access over the Internet.
<b><u>XMI</u></b>	The XML Metadata Interchange Format is an open interchange model for exchanging programming data over the Internet in a standardized way.
<b><u>XML</u></b>	eXtensible Markup Language is an open standard meta-markup language for structured documents and data on the Web.
<b><u>XSD</u></b>	XML Schema Definition Language is used to define specific schemas for XML objects, providing instructions about what constitutes a proper instance of an XML object.