Improving reusability with Web Services

Anders Persson
Anders Persson

Improving reusability with Web Services

Master’s Thesis

Datalogi

2006
Abstract

This master’s thesis discusses the evolution of DataPartner’s application "Inventera". Inventera is designed to do inventory of stock on handheld computers and integrates with SPCS Administration. Inventera is enhanced with a wireless connection and the SPCS API is wrapped with Web Services. The theoretical part of this thesis examines different possibilities to increase reusability by using web service technology when developing software. A case study of the SPCS API is used in order to study differences between using no Web Services at all, static web services or dynamic web services.

Acknowledgments

This master thesis has been written at the School of Mathematics and System Engineering (MSI) at Växjö University. I would like to thank all the staff at DataPartner in Helsingborg for giving me the opportunity to carry out this thesis and all the great support and idea’s. A special thank goes to Peter Blonér who acted as my tutor at DataPartner. I would also like to thank my tutor at MSI, Jesper Andersson, for good ideas and great support during this thesis.
# Contents

## List of Figures

## List of Tables

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Problem statement</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Delimitations</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Method</td>
<td>2</td>
</tr>
<tr>
<td>2 Supporting Technologies</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Web services</td>
<td>2</td>
</tr>
<tr>
<td>2.2 SOAP</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Microsoft .NET Compact Framework</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Web Services orchestration</td>
<td>7</td>
</tr>
<tr>
<td>2.5 BPEL4WS</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Software reuse</td>
<td>9</td>
</tr>
<tr>
<td>3 Web service framework</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Use case description</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Design</td>
<td>15</td>
</tr>
<tr>
<td>3.3 Implementation</td>
<td>16</td>
</tr>
<tr>
<td>4 Evaluation</td>
<td>17</td>
</tr>
<tr>
<td>4.1 Scenario 1: No Web Services</td>
<td>18</td>
</tr>
<tr>
<td>4.2 Scenario 2: Static web services</td>
<td>19</td>
</tr>
<tr>
<td>4.3 Scenario 3: Dynamic web services</td>
<td>20</td>
</tr>
<tr>
<td>4.4 Demonstrator - Lessons learned</td>
<td>20</td>
</tr>
<tr>
<td>4.5 Conclusion</td>
<td>21</td>
</tr>
<tr>
<td>5 Conclusion and future work</td>
<td>22</td>
</tr>
<tr>
<td>5.1 Future work</td>
<td>22</td>
</tr>
<tr>
<td>A References</td>
<td>23</td>
</tr>
</tbody>
</table>
List of Figures

2.3 Typical structure of a SOAP message 5
2.4 A SOAP message requesting the service bookflight 6
2.5 Showing the complexity the business services accessing each other directly 7
2.6 Showing the orchestration service with a single entry point 8
2.7 Implementation of an orchestration engine using knowledge engineering 8
2.8 A java class to picture information hiding 12
3.1 The Use Cases in the Web Service framework 13
3.2 The class diagram for the Documents and Document class 16
3.3 The class diagram for the Inventory class 16
3.4 Architecture diagram of the implementation of the demonstrator program 18
4.1 Workflow when using the SPCS Administration API without webservices 19
4.2 Workflow when using the SPCS Administration API with webservices 20
List of Tables

2.1 Description of the different metrics for software reuse according to Poulin [18] ................................. 11
3.1 Description of the web methods ........................................ 17
4.1 Summary of the evaluation metrics. .................................. 21
1 Introduction

In order to understand the background and purpose of this master’s thesis a brief explanation of the problem will be given in this chapter.

The goal of this thesis is to develop software that wraps SPCS Administration API to make it accessible via Web Services technology.

It will consider the possibility to make it easier to develop software that uses the SPCS Administration API using web services. Finally, it will study if reuse potential of software increases by the use of Web Services technology. SPCS Administration is Sweden’s most sold administrative program for smaller SMEs (1-20 employees). SPCS Administration includes, among other features, the following:

- Accounts
- Budget
- Offer
- Order
- Stock
- Invoicing
- Accounts payable ledger, and much more

SPCS Administration is a product line. The simple SPCS Administration, at one end, contains only Accounts and at the other end, SPCS Administration 2000 Network a multi-user system which includes most functions a small company needs from an administrative perspective. The API enables external applications to integrate and offers secure access to the SPCS Administrations database, meaning that everything that is done via the API is validated the same way as the data administration suite. By improving the usability, thus making it easier to develop software that utilizes the SPCS API, time and money can be saved. Today, similar code is rewritten for every program that uses the SPCS API.

1.1 Background

This master’s thesis was initiated by DataPartner in Helsingborg. They offered me to continue the development of their product Inventera. Inventera is currently used for inventory of stock with asynchronous server communication. A sync function and a direct connection to SPCS Administration is used. Today, DataPartner works with Delphi, components when developing SPCS Administration API applications. Delphi components enable some reuse but DataPartner would like to know if it is possible to develop a framework that wraps the API and open ups for reuse. To demonstrate the capabilities of the framework an application that uses the framework and a wireless connection, providing synchronous server connection will be developed. Especially they would like to investigate which benefits one could make in terms of improved reuse by employing Web Services technology.

1.2 Problem statement

- Will reusability be improved when the SPCS API is Web Service enabled?
• How should a framework which enables static composition of web services be designed and implemented?

• What is required to perform dynamic composition of the web services?

1.3 Delimitations

DataPartner has until today developed all their applications using Borland Delphi. Because of this, the demo application will be developed using Borland Delphi. Since the .NET version of Delphi does not support the .NET Compact Framework yet, the application for the handheld computer has to be developed with Microsoft Visual Studio .NET 2003.

Due to marketing reasons the demonstrator application will not exactly follow the framework that is developed in this thesis. The framework is supposed to work directly against SPCS Administration, but most users want to manually verify the information before it enters SPCS Administration. This is resolved with the introduction of an intermediate database.

1.4 Method

First we will start with studies of techniques that can be used to solve the problems above, namely .NET Compact Framework and Web Services. Making use of some small hands on tests, We will see what is possible to do and what is not. We will also look at software reuse, and investigate metrics for the evaluation to demonstrate whether reusability increases or not when the Web Service enabled the SPCS API. Then the design and analysis of the SPCS Administration API wrapper will begin. After that I will look at the requirements for dynamic web service composition. Finally, the implementation of the demonstrator application will start.

2 Supporting Technologies

This chapter describes the technologies that will be used to solve the problems above. The presentation is supported by the following scenario.

"Anna wants to purchase a vacation package using an online travel agent. To locate the best prices for airline tickets, hotels and car rentals the agency has to ask multiple companies. They all probably use different, incompatible applications for pricing and reservations". One solution to this problem is to use web services. These are described in the next sub section using the example above.

2.1 Web services

In this section an introduction to web services will be given. It will not contain any information about XML or standard Internet protocols since it is out of scope for this thesis. A definition of a web service is:

"A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [2]
A Web Service is not a single technology but rather a collection or framework of techniques that, when combined, support applications, over a network. The framework is divided in three parts:

- Communication protocols
- Service descriptions
- Service discovery

Looking back at our example, if the travel agent’s partners all expose their services as web services, the agent would be able to access all partners in a standardised way. This would make requests for pricing and reservations much easier. The aim of Web Services is to simplify processes like the one in the example. According to Peltz [17], senior information technology executives ranked "connecting to customers, suppliers or partners electronically" as the top global IT management issue. Web Services offer standard-based mechanisms that address this issue. The core standards of the Web Service framework are:

- Simple Object Access Protocol(SOAP)
- Web service Description Language(WDSL)
- Universal Description Discovery and Integration(UDDI)

These standards are depicted in figure 2.1 as components of the web service architecture stack. The stack shows how the techniques are interrelated and their layering. WSDL

*Figure 2.1: The web service architecture stack. Source: Web Services Architecture [2]*
[24, 5] describes an XML format and a model to describe web services. A separation between the description of their abstract functionality from the concrete details, such as "how" and "where", is offered. WSDL describes messages that are exchanged between a service provider and a requester. The messages are abstractly described and bound to a specific network protocol and format. They consists of a collection of typed data items. This means that the WSDL document provides a standard way to describe a Web Service interface, which is detailed enough to allow a user to build a client application that access the web service.

Third protocol that is considered to be a core protocol for Web Services is UDDI [22, 6]. UDDI specifies a way to publish and discover information about web services. The core component in UDDI is the business registration component, an XML file which describes a business entity and its Web Services. The business registration consists of three sub components: "white pages" including address and contact, "yellow pages" containing industrial categorization using standard taxonomies, and "green pages" holding technical information about the Web Services exposed by the business. Green pages contain references to Web Service specifications. UDDI uses standard SOAP messages for communication. The core information in UDDI is defined in an XML schema. This schema defines the SOAP messages and provides a description of the UDDI API. The four core types of information in the schema are business, service and binding information as well as information service specifications about services. This information is sufficient for a technically trained individual to discover and use a service.

A typical session accessing a Web Service, using a discovery service, is illustrated in figure 2.2.

Figure 2.2: Accessing Web Service using a discovery service like UDDI Source: Web Services Architecture [2]

2.2 SOAP

Simple Object Access Protocol SOAP [10, 11, 6, 21] is a standard XML-based protocol for messaging and remote procedure invocation over the Internet or any other network. SOAP provides means for communication between applications written in different languages, running on different operating systems. SOAP works on top of existing transport protocols such as HTTP, SMTP. SOAP was initially created by Microsoft and later developed in co-operation with Developmenttor, IBM, Lotus and Userland. Initially, it was
intended as a protocol for accessing objects over the internet, but since then many things have changed and SOAP has become interesting to a broader audience. Because the focus of the specification quickly moved away from objects towards a generalized XML messaging framework, this created a problem with the "O" in SOAP, but the working group has decided to keep it because the name has become so popular. It is interesting to note that the working group does not even mention objects in the official definition of SOAP 1.2. The core structure in SOAP is very simple since it consists of an XML envelope element that wraps the user defined messages. W3C approved version 1.1 of SOAP in July 2001 and since then W3C has been working with the specification of version 1.2. This was adopted in June 2003. The two most important design goals for SOAP are simplicity and extensibility. A SOAP message consists of three parts, all of which will be described below. Figure 2.3 shows the typical structure of a SOAP message.

Figure 2.3: Typical structure of a SOAP message

```xml
<SOAP:Envelope
    xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/
    <SOAP:Header>
        <!-- content of header goes here-->
    </SOAP:Header>
    <SOAP:Body>
        <!-- content of body goes here-->
    </SOAP:Body>
</SOAP:Envelope>
```

2.2.1 SOAP Envelope

A SOAP Envelope element is the root element of a SOAP message. This makes it easy for applications to identify SOAP messages. The SOAP Envelope must contain a SOAP Body element and may also contain an optional SOAP Header, both described below. The SOAP Envelope has to be in the namespace "http://www.w3.org/2003/05/soap-envelope".

2.2.2 SOAP Header

SOAP Header is an optional child element of the SOAP Envelope located in the SOAP namespace containing control information. It may consist of any number of elements from any namespace. These elements are called header blocks. Header blocks contain information about the payload processing. Header blocks can also be equipped with an attribute (mustUnderstand) from the SOAP namespace to indicate whether or not a receiver has to understand the header block before processing the message.

2.2.3 SOAP Body

A SOAP Body element is a mandatory child element of the SOAP Envelope element. The body represents the message payload. To represent the payload the header, just like the body, contains any number of elements from any namespace. Every inner body element under the body must be qualified by a namespace.
2.2.4 SOAP Fault

A SOAP Fault represents errors within the body element. This element is important. Without it every application would have to invent their own representation to distinguish between when things go wrong and when they don’t. The fault element contains an error code followed by a fault string element providing a human readable representation of the fault.

Looking back at the example in the beginning of this chapter our traveling agent knows of a service `bookflight` when she wants book her flight ticket. She then constructs a SOAP message and send it to the service. Table 2.4 displays the SOAP message.

![Figure 2.4: A SOAP message requesting the service `bookflight`]

```xml
<SOAP:Envelope
  xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:ex=http://www.example.org>
  <SOAP:Body>
    <ex:Ticket>
      <ex:passengerName first="Anna" last="Smith"/>
      <ex:flightInfo airline="SAS" flightNr="1234"
                      departureDate="2004-06-01"/>
    </ex:Ticket>
  </SOAP:Body>
</SOAP:Envelope>
```

2.3 Microsoft .NET Compact Framework

The .NET compact framework (.NET CF) [16, 19, 23] was released in March 2003. Since then, .NET CF has quickly become the de facto platform for Pocket PCs and by the end of 2003 it became available for Smartphones (Smartphones and Pocket PCs refers to products by Microsoft). Simply, one can say that the .NET CF is a slim version of the .NET Framework for desktop computers. Just like the .NET framework the .NET CF works with what they call managed code, i.e. instead of compiling the source code to native code that is understood directly by the processor it is compiled to managed code and then at runtime transformed by the .NET CF to native code that works with the current processor. In this way one can execute applications on all systems supported by the .NET framework. Thus you do not have to compile one version of your application for MIPS based device and one for the ARM based one. Support for web services in .NET CF is somewhat limited since you cannot create any web services, only use them. Microsoft created an abstraction layer Platform Adaptation Layer (PAL) for the underlying operating systems specific APIs. In the future the .NET CF could easily be ported to other operating systems just by creating the corresponding PAL. On top of the PAL .NET CF implements a Common Language Runtime (CLR). This CLR tells the Just in time (JIT) compiler how to do the transformation from managed to native code. The .NET CF supports all processors that the Windows CE operating system supports, which includes StrongARM, MIPS, x86, SH4, XScale to mention a few. Just like the .NET Framework, .NET CF is language independent, so any language that is .NET CF compatible will do. At the moment there are only two, namely C# and Visual Basic .NET. But others are working on releases that will be compatible with .NET CF, e.g. Borland Delphi .NET.
2.4 Web Services orchestration

Web Services offer a set of standard protocols to allow applications to communicate independently of the underlying operating systems and programming languages. This is, though, not enough to fully support business processes that have additional requirements, like transaction management, exceptions etc. One way to create an integrated system is to let the services communicate with each other, but according to Zhu [24] this easily gets out of control. Looking back at our example, the travel agency might access different services for hotels, flight reservations and car rentals, see the illustration in fig 2.5. As the figure shows, this approach tends to become very complex, even with only a few services. Web Services that calls other Web Services has to know the interfaces of the called Web Service, due to this. A large system tend to be very complex and ends up with the creation of multiple dependencies and the creation of multiple entry points in a system, this reduces manageability. This will also make it harder to change the logic in the system, since every service is dependent on every other service in the worst case. If one part of the system changes the change has to be done in many services. It is also harder to track down problem sources.

Figure 2.5: Showing the complexity the business services accessing each other directly.

The solution to this problem, according to Zhu [24], is to factor out the integration procedures into a dedicated web service called orchestration service. An orchestration service controls the integration as a whole. It calls the involved business services and then moves on to the next step which may include calling other business services. In this way the orchestration service becomes a single entry point for the services, as illustrated in fig 2.6. With this approach our travel agency accesses a single service and the services behind may change independently of its end usage. In this context, a travel agency is not the best example since the competing companies are not interested in co-operation.

According to Zhu [24], there are mainly two ways to implement the orchestration engine. One way is hardcode all integration business logic and business function calls into the engine. This will lead to a quick implementation but a static one which is difficult to change. The problems might not occur immediately but may show when business logic changes or when new business services are added. This may lead to a lot of work and recoding. The other path to follow is to separate the orchestration engine, the knowledge of the business services and the integration rules. This will lead to a more flexible and robust implementation which is easier to change and update, without the necessity to recode the
Figure 2.6: Showing the orchestration service with a single entry point.

Figure 2.7: Implementation of an orchestration engine using knowledge engineering.

entire orchestration engine. This technique is called knowledge engineering. To achieve an implementation like this, it is necessary to create a service registry and an integration rule knowledgebase. The local service registry could be a local UDDI service which stores information of the contributing business services. The integration knowledgebase stores the business logic, which specifies how to respond to each type of request. When receiving a request the orchestration engine queries the integration base to find the right type of services for the request. Then the engine searches the service registry to find the services needed to fulfill the request. This is illustrated in figure 2.7. In this way a certain request is never bound to a certain service. The orchestration engine relies instead on the service registry to find the right service.

According to Peltz [17] orchestrated web services must be able to handle errors and time-out constraints. Traditional ACID (Atomicity, consistency, isolation and durability)
transactions are not sufficient for long time running transactions, since they cannot lock resources in these kinds of transactions. To solve this problem, a kind of compensating transactions can be used. Compensating transactions provide a way to undo an action if a user cancels the action or if an error occurs.

2.5 BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS) [1] or BPEL for short, Web Services coordination (WS-Coordination) [4] and Web Service transactions (WS-Transaction) [3] provide support to create robust business compositions. According to Cubera et al. [7], BPEL supports a process-oriented form of service composition: each BPEL4WS composition is a business process or workflow that interacts with a set of Web Services to achieve a certain goal. BPEL compositions are called processes and the services that the process interacts with are called partners. Partners can be clients to the process or service providers to the process. Partners are defined through partner links, roles and the WSDL port types associated with a role. The process is represented by any Web Service that supports a set of WSDL interfaces that enable it to exchange messages with its partners. A process can also define fault handlers, or compensating handlers. Compensating handlers are used to undo an actions that has already been finished. Fault handlers provide a model to deal with errors occurring within a process. These are similar to try and catch blocks in Java. Compensating handlers are closely tied to fault handlers in BPEL.

WS-Coordination is a framework for implementing specific coordination types for example a coordination that requires a shared context. WS-Transaction defines two particular coordination types, one for short-running atomic transactions and one for long-running business activities.

2.6 Software reuse

2.6.1 History

The definition of software reuse is according to Schach et al. [20] as follows. Software reuse refers to the utilization of a software component C within a product P, where the original motivation for constructing C was other than for use in P. Component C may be any software component including, but not restricted to, a specification, plan, contract, design or code module or part of such component. The underlying idea of software reuse is that we can produce software faster, cheaper and work fewer defects. According to Poulin [18] many believe that software reuse provides the key to enormous savings and benefits in software development; the U.S. Department of Defence alone could save $300 million annually by increasing reuse as little as 1%.

2.6.2 Modules, Components and Frameworks

In this subsection a review of what distinguish modules, components and frameworks will be given. These three techniques have in common that they are all leverage reuse.

When a program is modularised the related parts of the program are collected together and considered as a single module. A typical example of a module is a class when using an object orientated paradigm. Modules can be reused within a project or in other projects. Modules contribute to a relatively low degree of reuse.

According to Johnson [12] the ideal reuse technology provides components (modules grouped together) that can be easily connected to make a new system. The idea of software components was proposed by McIlroy [15] in 1967 and in 1970 an application
of this idea was demonstrated by Lanergan and Paynton [13] with great success. They identified code and standard structures that could be used in many of their applications. The idea is simple: reuse one component in many software projects. An example of a technique that is supposed to support the development of components is JavaBeans. JavaBeans is the component architecture for the Java programming language by Sun. The goal of JavaBeans is "Write once, run anywhere, reuse everywhere". This sentence describes pretty well the goal of software reuse.

There are two types of reuse. Deliberate reuse occurs when the component was originally constructed for reuse and accidental reuse occurs when the component was originally constructed for a specific product and later reused in another one. The software developer does not have to know how the component is implemented, and the specification of the component is easy to understand. The resulting system will be efficient, easy to maintain and reliable. A typical example from the real world is the electrical system: anyone can connect a toaster from one store and a television from another, and they will work at both your home and office, without you knowing Ohm’s law.

A framework is according to Johnson [12] as follows "a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact". According to Erich Gamma [9] a framework is: "A set of cooperative classes that makes up a reusable design for a specific class of software. A framework provides architecture guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes." Today frameworks is a widely used technique, some examples are the .NET framework by Microsoft and Java’s graphical interface framework Swing.

Fayad et al. [14] has listed the following four primary benefits of using frameworks

- **Modularity** is enhanced by frameworks by hiding implementation details behind stable interfaces. Framework modularity also contributes to software quality by making the penetration of design and implementation changes local. Due to this localization the effort to modify and maintain existing software is decreased.

- **Reusability.** The interfaces provided by the framework helps improve reusability by defining generic components that can be reused to create new application rapidly. In this way prior domain knowledge can be reused making requirements and software design in new applications easier. Reusability also affects the programmer productivity, enhance the quality, performance and interoperability of the software.

- **Extensibility.** By providing hook methods applications are able to extend the stable interfaces of a framework. Hook-methods can be used to decouple the stable interfaces providing possibility to make various changes to framework that might be required by the application in some particular context. Framework extensibility is fundamental to make customization of new software timely.

- **Inversion of control** is what characterizes the run-time architecture of a framework. This architecture enables the framework to invoke event handlers via a dispatching mechanism. When events occurs the dispatcher invokes the pre registered handler methods. The inversion of control allows the framework rather than the application to settle which set of application specific methods to be invoked when events occur.
2.6.3 Measuring Software Reusability

Measuring software reusability is not an easy task. According to Schach et al. [20] research has been conducted metric-based predictions whether existing software is a good candidate for future reuse or not. Depending on the metric, a module is either discarded or handed over to a domain expert for further analysis regarding reusability. The problem is that we do not know whether this approach works or not. That is to say, no research has been published to show that discarded modules really are not good material for reuse. Many metrics are based "on the size" of code properties, for example complexity, volume, number of comment words etc. These metrics are not very useful to our investigation because the comparison involves the API and a Web Service including same amount of code. Metrics, better suited for this thesis are metrics such as information hiding, high cohesion and low coupling and portability. The metrics are explained in table 2.1.

Table 2.1: Description of the different metrics for software reuse according to Poulin [18]

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>The component does not depend on unique hardware or operating system services.</td>
<td>This is measured by comparing the portability of the SPCS API before and after the use of Web Services.</td>
</tr>
<tr>
<td>Information hiding</td>
<td>The component hides implementation details from the user, for example, internal variables and their representation. It also clearly defines the interfaces to other operations and data.</td>
<td>This is measured by comparing the interfaces that the user have to know about before and after the introduction of Web Services.</td>
</tr>
<tr>
<td>High cohesion and Low coupling</td>
<td>The component does a specific, isolated function with minimal external dependencies.</td>
<td>This is measured by comparing the cohesion and coupling before and after the introduction of Web Services.</td>
</tr>
</tbody>
</table>

An example of portability is software that can be used with both Microsoft Windows and Linux Operating system, for instance the Apache web server.

To illustrate information hiding, a Java class is shown in figure 2.8. By using the methods setMembervalue and getMembervalue the internal member variable never has to be accessed directly and the user of the class just have to know about the methods not the variable.

Cohesion is the degree to which the responsibilities of a single component form a meaningful unit and should be high because this means that the component is more useful. According to Dhama H. [8] "Cohesion is an intramodule property that reflects the design considerations for integrating the various components of the module into one unit. It is the glue that holds a module together, and it is a measure of the logical strength of a software module. The strength and consequently the "quality" of the module increase with increasing cohesion."

Coupling applies to any relationship between software components and should be low since this means that the component does not depend on other components. According to Dhama H. [8] "Coupling is a measure of the interdependence between two software
Figure 2.8: A java class to picture information hiding.

```java
public class information_hiding {
    private string membervalue;

    public void setMembervalue(string value) {
        membervalue := value;
    }

    public string getMembervalue() {
        return membervalue;
    }
}
```

modules. It is an intermodule property. Because it is desirable that the changes made in a module affect another module as little as possible, the "quality" of a module increases as module coupling decreases."

These three metrics will be used in the evaluation.
3 Web service framework

In this chapter we describe the analysis, design and implementation of the Web Service framework. Figure 3.1 shows the use-cases for our framework. There are four packages in the framework. First, the Documents package that contains the use cases that handles everything that has to do with offers, orders and invoices. Second the Articles package which contains the use cases to handle the article registry. Further, the Customer’s package containing use-cases to handle the customer registry, and finally, Suppliers package which contains use cases for the supplier registry. The first release from this project will include the use cases in the package documents and the inventory use case from articles. These use cases will be described later. Since the actor of the use cases is rather undefined, it can be everything from a web service to a user utilizing a program which in turn employs the web services. Therefore the actor is left out in the diagram.

3.1 Use case description

3.1.1 Use case Add Documents

Pre condition: The user wants to add an offer, order or invoice to SPCS Administration
Post condition: The offer, order or invoice should have been added to SPCS Administra-
**Process description:**

1. The user collects the necessary information to create a document.
2. The information is verified.
3. If the information is correct the document is added to SPCS Administration.
4. The user gets information if the operation was successful.
5. The use case ends.

**Exception, Error situation:** If the information is incorrect the user shall be notified

**Additional information:** The minimal information in 1 is for documents: A customer number and at least one article row (article number and quantity). For invoices invoice type also has to be specified.

3.1.2 Use case Get Documents

**Pre condition:** The user wants to retrieve a document (offer, order or invoice) from SPCS Administration

**Post condition:** The user received the requested document.

**Process description:**

1. The user collects information to specify what document (via a document number) to be retrieved from SPCS Administration.
2. The system tries to find the document
3. If the system finds the document the user gets it.
4. The use case ends.

**Exception, Error situation:** If the service cannot find the document the user shall be notified

3.1.3 Use case Update Documents

**Pre condition:** The user wants to change an document.

**Post condition:** The document is changed in SPCS Administration.

**Process description:**

1. The user collects information about which document (via a document number) and which parts in it to update.
2. The system validates the information.
3. If the information is correct the system attempts to update the document.
4. The use case ends.

**Exception, Error situation:** If the system cannot find the document or if the update information to update the document with is incorrect the user shall be notified.
3.1.4 Use case Search

**Pre condition:** The user wants to find a document.
**Post condition:** The user got the sought document.
**Process description:**

1. The user collects information about what to find and specifies a search order.
2. The system tries to find the requested document.
3. The use case ends.

**Exception, Error situation:** If the API cannot find the requested document the user shall be notified.

3.1.5 Use case Inventory

**Pre condition:** The user has an article that should be inventoried.
**Post condition:** The article is inventoried in SPCS Administration.
**Process description:**

1. The user collects information about what article that is going to be inventoried namely the article number and a quantity.
2. The system checks that the article exists and that it is in stock, then it adds the inventory line to SPCS Administration.
3. The use case ends.

**Exception, Error situation:** If the article does not exists in SPCS Administration or if it is not in stock the user shall be notified.

3.2 Design

In this section we explain the classes of the web service framework. The first release will include the use cases from the package documents and the use case inventory from articles package.

3.2.1 Documents

Figure 3.2 shows the class diagram for the document and documentService class. The class implements the use cases in the documents package. The document class represents a document as it is in SPCS Administration and is just a helper class to simplify the document handling. It contains a head and a row object that is of the type Dataset, and then there is method for setting and getting the head respective the rows. The class documentService contains three methods. The addDocument method adds the specified document object to SPCS Administration, the getDocument method returns the specified document from SPCS Administration and finally the search method that returns the sought document from SPCS Administration.
3.2.2 Inventory

Figure 3.3 shows the class diagram for the inventory class. The class implements the use case Inventory from the package articles. The Inventory class contains three methods. The addToArticle() adds the given number to the article inventory list in SPCS Administration, the checkArticle() method returns the total number inventoried at the given article, the getArticleInfo() method returns a dataset containing information about the article, This could be the name, barcode etcetera.

3.3 Implementation

In this section a brief description of the implementation of the demonstrator will be given. As mentioned in chapter 1 the implementation does not follow the design completely due to marketing reasons. Figure 3.4 depicts the architecture of the demonstrator system. The figure depicts a middleware database that was introduced since customers wanted to check the data before it entered SPCS Administration via the API. Customers that do not want this feature can of course disable it, enabling the changes from the handheld to enter SPCS Administration directly.

The implementation of the web services is coded in Borland Delphi .NET and executed on a Microsoft Internet Information Server. There are currently 9 web methods and these are described in table 3.1.
Table 3.1: Description of the web methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function getOrders() : DataSet;</td>
<td>Fetches orders ready to be picked from SPCS Administration and returns a .Net DataSet</td>
</tr>
<tr>
<td>function getOrderHead(ordernr : Double; userid: string) : DataSet;</td>
<td>Returns the order head from SPCS Administration of the given ordernumber as a .Net dataset and marks that order as being picked by the given used id.</td>
</tr>
<tr>
<td>function getOrderRows(ordernr : Double) : DataSet;</td>
<td>Returns the orderrows from SPCS Administration of the given order number</td>
</tr>
<tr>
<td>function putPicked(ordernr: Double; artnr: string; quantity: Double) : Boolean;</td>
<td>Adds the picked orderrows to the middleware database.</td>
</tr>
<tr>
<td>function putInvent(artnr: string;quantity: Double; userid: string): Double;</td>
<td>Adds an row to the middleware database Inventory with the given article number and user id</td>
</tr>
<tr>
<td>function getArtInfo(barcode : string) : DataSet;</td>
<td>Returns a .Net DataSet with information about the given article number</td>
</tr>
<tr>
<td>function searchArt(search_string : string; search_field : string) : DataSet;</td>
<td>Searches SPCS Administration for the given article and returns the article number</td>
</tr>
<tr>
<td>procedure exitProgram(UserID : string);</td>
<td>Method that is called when the user exits the program om the handheld. The method resets all locks on orders done by the user</td>
</tr>
<tr>
<td>function getInvents(artnr : string) : DataSet;</td>
<td>Returns a .Net dataset with information about all the inventory rows in the database of the given article number.</td>
</tr>
</tbody>
</table>

The handheld part for the .NET CF is written in C# using Microsoft Visual Studio .NET 2003. The handheld used is a Unitech PA950 with pocket PC 2003 and .NET Compact Framework. The application for the handheld contains two parts, one to do order picking and one for inventory. Figure ?? shows the main screen of the program.

4 Evaluation

This section evaluates how the efficiency when developing software employing SPCS Administration API is affected when web service enables the API. This will be done by describing three scenarios. The first one show how the software development was done before using web services meaning that the API was accessed directly. The second reflects the method when using static web services to access the API and the third shows how the API could be accessed when using dynamic web services. The scenarios will first be described with a workflow for the current scenario and then evaluated with the three metrics to evaluate the software reuse. The metrics are portability, information hiding and
high cohesion and low coupling.

4.1 Scenario 1: No Web Services

Before the time of web services the software had to be composed by several different modules. These modules had to be configured separately to access the required parts of the SPCS Administration API. This process could be quite time consuming and the configuration would look very similar for the different software, but had to be done for every software project. Further, the software was very tight coupled together with the SPCS Administration API. Problems would arise if the software had to be connected with another system than SPCS Administration API or if the software was supposed work apart from the SPCS Administration API such as a remote client. Next is a description to get a better understanding of how an SPCS Administration API application could look like. The connection to the API is as follows:

- First the SPCS Administration API dll file has to be located and loaded into the memory and then the database selected.
- When this is done the next thing to do is to create a data structure that corresponds to the correct table of the database which will be used in the application.
- Now we are able to either conduct a search with the data structure as a pointer and a search value that will return the structure filled with the resulting search data. Alternatively we can insert new data and then call the add function with the structure as a parameter to add a new document. The resulting data from the search can also be utilized to modify a existing database record in SPCS Administration.
A corresponding data structure has to be created for every different document type in the SPCS Administration that is going to be used.

The workflow in scenario 1 is illustrated in figure 4.1. When the SPCS API is employed in this way, it is tightly bound to the Win32 environment making the portability low. We are also forced to work with lots of details of the SPCS API that could easily have been hidden.

When looking at the three metrics to measure software reuse beginning with the portability,

- The portability is pretty low since we are forced to use a Win32 environment because that’s the only platform that handles dll files.

- When looking at the information hiding the result from this scenario shows that the information hiding is low since the user(programmer) is using pointers to data structures, hence the user have to know everything about these. These structures isn’t really that interesting for the user and could have been hidden. That gives the information hiding a minus.

- The cohesion is good because the SPCS API does what it’s supposed to do not more not less. But when looking at the coupling we see that we are tightly coupled to the dll file and because of that the cohesion and coupling gets a minus.

**Figure 4.1:** Workflow when using the SPCS Administration API without webservices.

4.2 Scenario 2: Static web services

With a Web Service facade, the process of creating software that takes advantage of the SPCS Administration API changes a bit. Instead of using different modules, a connection to the server running the web services is created. The workflow is depicted in figure 4.1 and described here.

- The first thing to do when using a web service is to import the WSDL document. This step is usually done automatically by either the Integrated Development Environment (IDE) or by an external software that generates stubs for the different method calls that the web service offer.
When the WSDL is mapped the next step is really just to call the web service when needed. The web service returns datasets and there is no need to use pointers or to manually create the data structures to be returned.

The obvious advantage in the scenario is that the process when connecting to SPCS Administration API is shortened down to only two steps: First import WSDL and second call the web service. What is most beneficial is that the risk to make mistakes is reduced since the user doesn’t have to deal with pointers.

When looking at the three metrics is as follows.

- Portability is raised a lot because now we have the possibility to move the client application to almost any platform, as long as we have a web service implementation for that platform. Moving on to the information hiding, we can see that even here the level is raised, since the user does not have to deal with weather internal structures of the SPCS Administration API nor pointers.

- Cohesion is not really affected by the web service.

- Coupling at other hand is affected that makes coupling lower since we are no longer bound to the win32 platform at the client side.

Figure 4.2: Workflow when using the SPCS Administration API with webservices.

4.3 Scenario 3: Dynamic web services

As described in the previous scenario the reusability will improve by web service enabling the SPCS API. By using a dynamic binding in the web service it would be possible to direct the incoming calls to different background systems without the user’s knowledge. The workflow in the scenario is about the same as in scenario 2, the big change in this scenario is what happens behind the curtains. When a client makes a call to the web service, the orchestration engine directs the calls to the correct system based on orchestration rules. With this solution it is possible to create one single entry point to services that could be used by all applications. Even better, a generic web service for handling say order routines could be built. In this way any administrative system could be implemented behind the curtains and this would increase reuse a lot since every program written to consume the ordering service could connect to any administrative system. Since this scenario only make a slight change to the workflow both the portability and the information hiding is not changed at all compared to scenario 2. On the other hand the coupling is lowered even more since we replace the system behind the service without the consumer’s knowledge. This gives the cohesion and coupling a double plus in our evaluation.

4.4 Demonstrator - Lessons learned

When the demonstrator was built I realised that web services is not the fastest way of communication. It takes rather long time to get an answer from the server when requesting a web method. If this solution will be implemented as a commercial solution some optimization has to be done to the communication since it is too slow as it is today.
4.5 Conclusion

To conclude, one can say that there is a big difference between the first and the second scenario when looking at the reuse aspect. For example, coupling, and portability. Web Services makes the coupling go down which is good and the portability increases. This occurs since the SPCS API, that is only possible to consume on a Win32 platform, is now possible to consume from any platform that supports the Web Service standards. The improvement is not that big when looking at the third scenario even if one can change everything behind the web services without having to change anything in the clients. However, this does not change the reusability much. To conclude the scenarios, one can say that Web Services increases reusability at least when measured with our rough metrics. This does not however conclude that it is appropriate to use web services in every application or project. As mentioned above the web services require a lot from the hardware at least when running the Microsoft .NET framework.

Table 4.1: Summary of the evaluation metrics.

<table>
<thead>
<tr>
<th>Scenario/Metric</th>
<th>Portability</th>
<th>Information hiding</th>
<th>High cohesion and low coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario1: No Web Services</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Scenario2: Static Web Services</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Scenario3: Dynamic Web Services</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>
5 Conclusion and future work

In this master thesis I wanted to investigate reusability, developing software would increased with web service technology. To do this I have web service enabled some parts of the SPCS API and developed a small client/server application which operates wireless with web services against the SPCS API. The client in this case was a handheld computer running Pocket PC and the server using Microsoft Internet Information Server running the ASP.NET web services. The client application was developed using .NET compact framework with Visual Studio 2003 and the web services were developed using Borland Delphi 8 .NET. One big problem during the development process was that both the .NET framework and the .NET compact framework had performance problems. But even worse was the very slow communication between the client and the server. Even with the small amount of data that I have been using in my tests, the response is too slow.

I have created three scenarios when looking at the increase of reuse ability. First a scenario without web services just using software modules. In the second scenario the modules where web service enabled. In the last scenario I added an orchestration engine that made it possible to replace the services behind the web service. The conclusion of these three scenarios is that when the SPCS Administration API is web service enabled the reuse increases, at least when using my three metrics. This masters thesis contains a highly subjective qualitative evaluation of improvement when using Web Services. It would have been better if the measure on "real" sourcecode, but in this case it haven’t been possible due to pratical things like time limitations.

5.1 Future work

In the future the web service framework will be further developed to support all the functions in the SPCS API. These functions will probably be supported in the client application as well.
A References


