LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology

Construction Plan Image Service for Mobile Devices

Bachelor's Thesis, Final Report
Examiners: Docent Jouni Ikonen
M.Sc Harri Hämäläinen

Risto Laine
Punkkerikatu 15 D 62
53850 Lappeenranta
p. +358 44 0522 653

Lappeenranta 03.09.2009

ABSTRACT

Lappeenranta University of Technology,
Department of Information Technology

Author: Laine, Risto
Subject: **Construction Plan Image Service for Mobile Devices**

Year: 2009
Place: Lappeenranta

Bachelors Thesis. Lappeenranta University of Technology. 48 pages, 16 figures, 1 chart and 2 appendixes.

Examiner: Dosent Jouni Ikonen
M.Sc. Harri Hämäläinen

Keywords: Web Service, API, CAD, BIM, Construction management, Tekla Structures

The forthcoming media revolution of exchanging paper documents to digital media in construction engineering requires new tools to be developed. The basis of this bachelor's thesis was to explore the preliminary possibilities of exporting imagery from a Building Information Modelling –software to a mobile phone on a construction yard. This was done by producing a Web Service which uses the design software's Application Programming Interface to interact with a structures model in order to produce the requested imagery.

While mobile phones were found lacking as client devices, because of limited processing power and small displays, the implementation showed that the Tekla Structures API can be used to automatically produce various types of imagery. Web Services can be used to transfer this data to the client. Before further development the needs of the contractor, benefits for the building master and inspector and the full potential of the BIM-software need to be mapped out with surveys.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto,
Tietotekniikan osasto

Tekijä: Laine, Risto

Nimi. **Kuvapalvelu rakennusmallinnusohjelmasta mobiililaitteille**
Vuosi: 2009
Paikka: Lappeenranta

Kandidaatin työ. Lappeenrannan teknillinen yliopisto. 48 sivua, 16 kuvaa, 1 taulukko ja 2 liitettä.

Tarkastajat: Dosentti Jouni Ikonen
DI Harri Hämäläinen

Hakusanat: Web Service, API, CAD, BIM, rakennusteollisuus, Tekla Structures

Vallankumous digitaalisen median käyttöönotoksi paperin sijaan rakennustyömailla, tiedon hallinnan ja tehokkuuden parantamiseksi, vaatii uusien tiedonhallintavälineiden kehittämistä. Tämän kandidaatintyön tarkoituksena oli kartoittaa alustavia mahdollisuuksia kuvainformaation välittämiselle rakennuksen tietomallinnus ohjelmistosta työmaalla sijaitsevaan matkapuhelimeen. Tämä toteutettiin kehittämällä Web Service, joka tuottaa visuaalista materiaalia käsittelemällä rakennuksen mallia suunnitteluohjelmiston ohjelmointirajapinnan kautta.

Matkapuhelimet todettiin vajavaisiksi loppukäyttäjänä vajavaisen prosessointitehon ja pienten näyttöjen takia. Toteutettu palvelu kuitenkin osoitti että Tekla Structures:n ohjelmointirajapintaa voidaan käyttää automaattisesti tuottamaan hyvin monenlaista visuaalista materiaalia rakennuksen mallista. Web Service on toimiva ratkaisu tämän tiedon siirtämiseen. Jatkokehitystä varten rakennuttajan tarpeet, rakennusmestarille ja tarkastajalle tulevat hyödyt ja ohjelmiston todellinen potentiaali tulisi kartoittaa.

INDEX

1 Introduction 3
2 Building blocks of the service 5
2.1 Building Information Modelling 5
2.2 .NET 6
2.3 Web Service 8
2.3.1 Infrastructure 8
2.3.2 Simple Object Access Protocol (SOAP) 10
2.3.3 Web Service Description Language (WSDL) 12
2.3.4 Universal Description, Discovery and Integration (UDDI) 14
3 Fitting mobile and modelling 16
3.1 Mobile Device 16
3.2 Mobile Networks 17
3.3 Tekla Structures 18
4 Potential types of imagery 19
4.1 Screenshot 19
4.2 Partial and full model 20
5 Implementation 24
5.1 Requirements 25
5.2 Operation 26
5.2.1 Communication 28
5.2.2 Program flow 29
5.2.3 Macros 31
5.3 Problems, observations and solutions 34
6 Demo system 37
6.1 Unable to connect to Tekla Structures API 37
6.2 Observations about the demo system 38
7 Discussion 40
8 Conclusions 42
9 Bibliography 43
Appendix 1: WSDL interface description 46
Appendix 2: SOAP 1.2 request and response 48

Abbreviations

API	Application Programming Interface
BIM	Building Information Modelling
CAD	Computer Aided Design
CIL	Common Intermediate Language
CORBA	Common Object Requesting Broker Architecture
CLI	Common Language Infrastructure
CLR	Common Language Runtime
DWG	AutoCAD format: ‘drawing‘
EDGE	Enhanced Data rates for GSM Evolution
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
JPEG	Joint Photographic Experts Group, image format
JVM	Java Virtual Machine
PDA	Personal Digital Assistant
RAM	Random Access Memory
RFID	Radio Frequency Identification
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
TCP/IP	Internet Protocol Suite: Transfer Control Protocol / Internet Protocol
UDDI	Universal Description, Discovery and Integration
UI	User Interface
URL	Uniform Resource Locator
QIM	Quality and Inspection Management
W3C	The World Wide Web Consortium
WLAN	Wireless Local Area Network
WSDL	Web Service Description Language
XML	Extensible Markup Language
XML-RPC	XML based Remote Procedure Call –protocol

1 Introduction

Visual information with properties and placement of the construction elements is needed on all stages of a construction project. Previously such information has been carried on construction yards as paper documents [LEU08]. Integration of information technology and construction engineering presents new possibilities for more effective information management; the large drawings which tear and smear in the harsh environment can be replaced by digital versions to be viewed on computers and handheld devices enhancing information management and thus boosting performance. Extra devices need to be carried along on the construction yards to handle this digital information.

The research problem for the thesis is to explore the preliminary possibilities of exporting digital imagery from a construction design software to a mobile phone. This will be done by developing an extension for the construction design software. The service is a part of Mobilding project [MOB07] in which the utilization of mobile technologies for construction engineering is researched. The elements on the construction yard have been marked with barcodes or electronic identification tags. These tags are read with a suitable scanner which is attached to the mobile phone. The phone uses mobile networks to query for imagery based on the identification code from a server. The server will relay the query to the service which will then interact with the construction design program, locate the element and return requested image (see Figure 1).

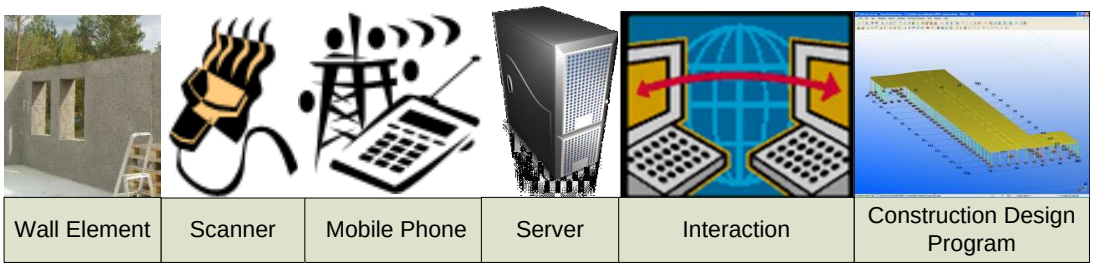

Figure 1: Overview of the Mobilding [MOB07] system

The work does not include information about the Radio Frequency Identification (RFID)-tags or the methods of inserting or reading them (for more information see [KAL06]) or the development of the software needed to view the modelling files such as DWG on a mobile phone. It will only offer few observations about the requirements of such software. Neither will it offer information about security concerns as the service will be implemented in a closed environment. It does not interact directly with the mobile phone as an existing module handles client communication. It only needs features to handle service queries and interact with the design program.

The second chapter starts by presenting Building Information Modelling (BIM) as a key concept of the construction design software Tekla Structures, which was used during the development. The development framework .NET will then be introduced. It will continue on reviewing the basic features, infrastructure and protocols of Web Services, which were used to produce the service functions. The third chapter will focus on discussing special features the environment bestows upon the project. The possibilities for the type of imagery to be returned are discussed in chapter four. The implementation of the service is covered in the fifth chapter describing the operation and going through the development stages reviewing observations, problems and solutions. The implementation chapter is followed by a preview of the demo setup. The last chapters five and six will discuss the key points and draw conclusions of the project.

2 Building blocks of the service

Building Information Modelling is a key concept of Tekla Structures, which is the construction design software used during the development of the service. Tekla Structures API (Application Programming Interface) is based on Microsoft's .NET framework. The basics of the frameworks operation needs to be opened in order to understand Web Services which are the base of service functionality. The infrastructure and basic protocols of Web Services will be reviewed.

2.1 Building Information Modelling

BIM is the process of generating and managing building data during its life cycle. It is a 3D object-oriented approach to computer-aided architectural design enabling data for manufacturer's details to be imported right into project design, and to be presented with 3D models of products in place in buildings. It provides an ability to add detailed imagery and information to everyone involved in the building process including element producer, steel manufacturer, architect, contractor, constructor, etc. [EAS07].

In CAD (Computer Aided Design) technology the 3D objects represented are machine readable, meaning that spatial conflicts in a building model can be checked automatically. Because of this capability, at both the design and shop drawing levels, errors are greatly reduced. Because the building models are digital information and thus machine readable it is practical to also distribute this information among participants of the project. The distribution of information amongst different operators of the building project is what differentiates BIM from CAD [EAS07]. CAD is designing and BIM is sharing the design information during the life cycle of the project. BIM is the key in reducing the amount of manual paper work during construction projects by removing the paper drawings and replacing them with centralized digital representations on digital devices.

2.2 .NET

.NET was a main concept during the implementation phase of the thesis. It provides ground for understanding Web Services in terms of middleware providing transportability. The .NET framework is multilingual and portable development framework which provides support for developing cross platform and distributed services. Currently .NET is only available for Windows environments, but there are open source projects for transporting it to other platforms. The application itself can be coded with various different languages including C# and Visual Basic. The framework itself consists of a Common Language Runtime (CLR) and .NET framework class library [MSDN].

CLR is the foundation of the .NET as it manages the code at execution time. The application code is compiled into a platform neutral Common Intermediate Language (CIL) which the CLR then interprets to machine readable code that can be executed on the current platform giving it the ability to be portable to wherever the .NET framework is present (see figure 2) [MSDN].

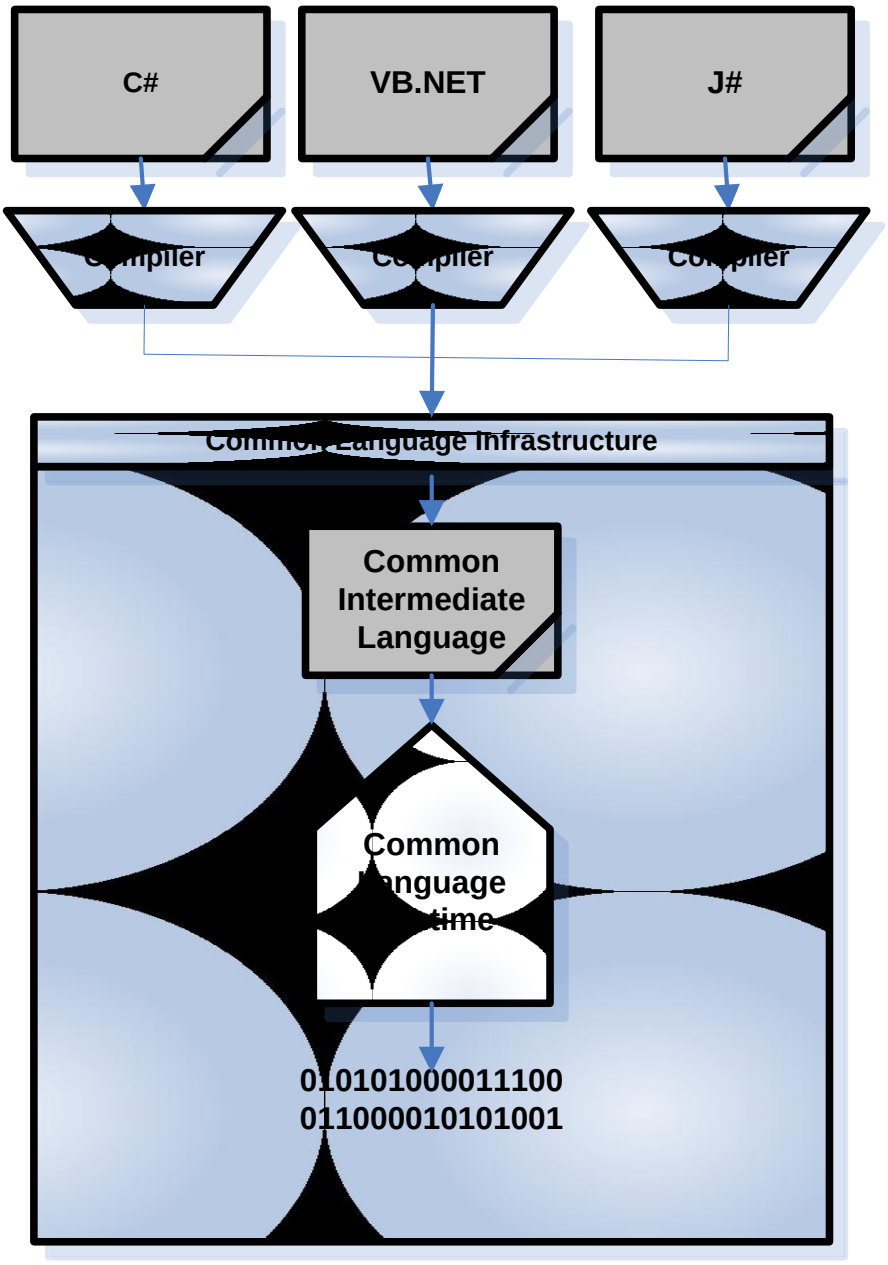

Figure 2: Common Language Infrastructure (CLI) [MSDN]

The class library implements a large number of common functions, such as file reading and writing, graphic rendering, database interaction, and XML (Extensible Markup Language) document manipulation, among others. It is a comprehensive, object-oriented collection of reusable types that can be used to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on ASP.NET, such as web forms and XML Web Services [MSDN].

The CLI shown in figure 2 and .NET languages such as C# and Visual Basic have many similarities to Sun's JVM (Java Virtual Machine) and Java. Both are based on a virtual machine model that hides the details of the computer hardware on which their programs run. The service could have been developed with other frameworks also if not for the .NET API of Tekla Structures.

2.3 Web Service

Web services provide a standard way of operating between different software applications, running on a variety of platforms and/or frameworks [W3C04]. They are an attempt to solve the current complexity problems of application integration by standardizing middleware systems [ALO04], where middleware means a software that functions at an intermediate layer between applications and operating system or database management system, or between client and server [EBU]. The technology evolved to provide interoperability to support the move to coherent distributed architectures. It is a move towards service-oriented architectures [ALO04].

As a short term Web Services are going to solve interoperability problems with application integration. As a long term, Web Services are going to be the foundation for a semi-automated or fully automated infrastructure for e-business, e-commerce and application integration [ALO04].

Overall Web Services are not a revolutionally new method of creating distributed systems. They have great similarities for the older RPC (Remote Procedure Call) technologies, but they have matured a lot faster. This is because they are standardized, based on accessible technologies and their interoperability has been a primary concern from the start of development [PHIL]. All this does come with a cost, but what is lost in performance is gained in availability and accessibility.

2.3.1 Infrastructure

In order to understand the infrastructure of the Web Services we will now go through the stages of building a Web Service from ground up. First we need a common syntax – the grammatical rules we are going to use in our communication. In case of

Web Services this syntax is XML which is a well known and proved technology for flexible way to describe data structures and formats [KIM06].

Then we need to know how to use the syntax provided. SOAP (Simple Object Access Protocol) is used commonly as a communication protocol between the Web Service requester and provider. It is a standardized way to encode different protocols and interaction mechanisms into XML documents and to exchange them over the Internet [ALO04].

We also need to describe our Web Service in a standard way so that users can read the instruction on how to invoke our service. WSDL (Web Service Description Language) is supported by W3C (The World Wide Web Consortium) as a standard way for describing the interfaces of Web Services.

If the Web Service is wanted to be made available for public it needs a place where the requesters can find it. A platform independent UDDI (Universal Description, Discovery and Integration) handles the role of name and directory service for Web Services. It describes the place where the service can be discovered (URL [Uniform Resource Locator]) and the interface how to invoke it. In a case of closed environment Web Service which is developed for this thesis there is no need for this method.

The overall resulting infrastructure can be seen on figure 3. We now have a Web Service (provider) and a client for the service (requester). The requester can find the service by using the discovery service UDDI where the Web Service has been published. Requester finds out the location (URL) and the interface of the Web Service described with WSDL. It then composes a message with XML to the Web Service and enclosed it in a SOAP envelope. The envelope can then be transferred using compatible transfer protocol such as HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol) or TCP/IP (The Internet Protocol Suite).

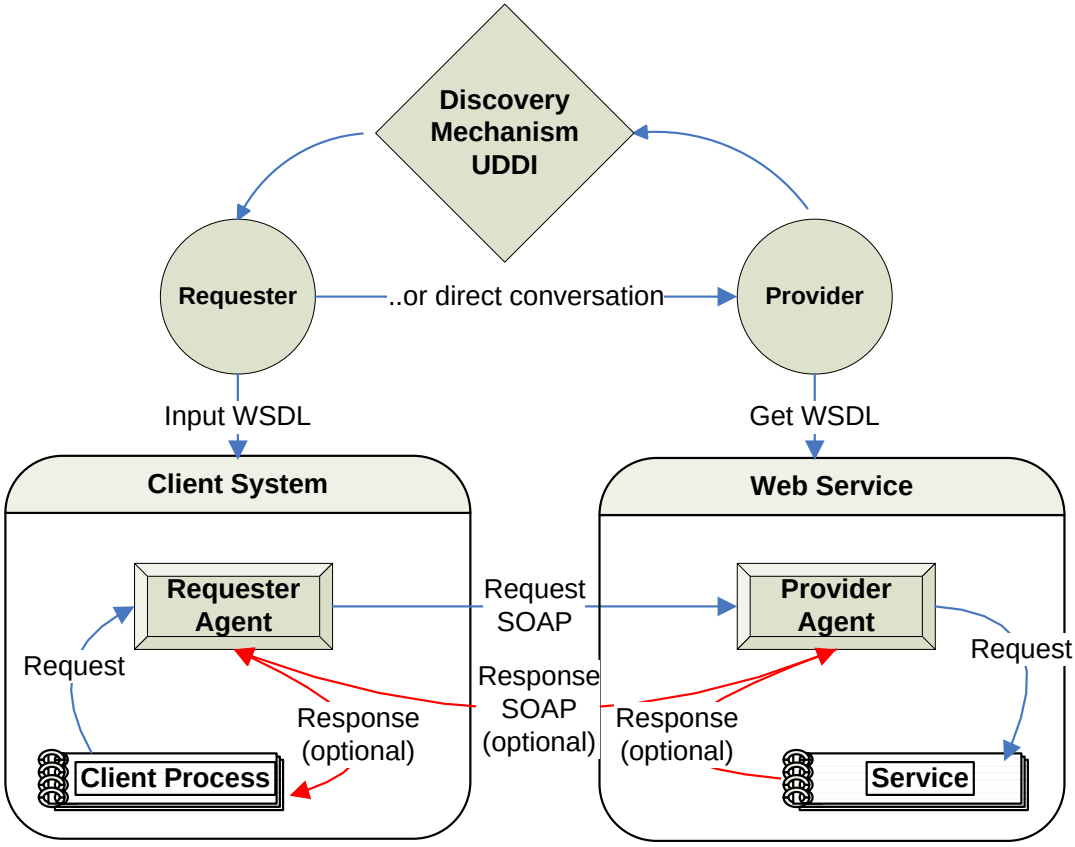

Figure 3: Web Service Infrastructure [PHIL]

The Web Service will thus provide a standard means of interoperating between different software applications coded in various programming languages, running on a variety of platforms and/or frameworks and which are connected to the Internet.

The basic protocols of Web Services will now be examined in more detail. SOAP and WSDL will be gone through in depth while only the surface of UDDI will be covered, because it is not implemented on the service.

2.3.2 Simple Object Access Protocol (SOAP)

The most common communications protocols used on Web Services are SOAP and XML-RPC. While XML-RPC aims for simplicity, SOAP adds features to further define the content of the messages and description of the service. This chapter will concentrate on SOAP as it was used on the implementation of the Web Service.

SOAP is an application layer protocol that provides the foundation layer of Web Services protocol stack, providing a basic messaging framework upon which Web Services can be built. In the W3Cs note for SOAP 1.1 [W3C00] it is described as a lightweight protocol for exchanging of information in a decentralized, distributed environment. 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, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols.

The basic structure of a SOAP message can be seen in figure 4. The message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The envelope defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory. It can also contain namespace declarations as well as additional attributes.

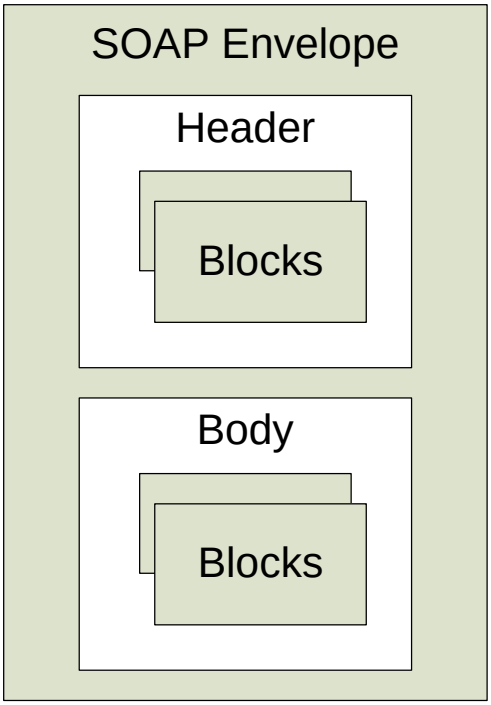

Figure 4: SOAP message structure [ALO04]

The header is a mechanism for adding features to a SOAP message in a decentralized manner without prior agreement between the communicating parties [W3C00]. It defines attributes that can be used to indicate who should deal with a feature and whether it is optional or mandatory. Typical features that can be implemented in the header are authentication, transaction management, payment etc. The header is optional but if present in the SOAP message it needs to be the first element on the SOAP envelope.

The body of the message is for exchanging mandatory information intended for the ultimate recipient of the message. Typical uses of the *body* element include marshalling RPC calls and error reporting [W3C00]. Each attribute will be encoded separately. If the *header* is absent then the body will be the first element of the envelope. The *header* and *body* can be extended as *header blocks* or *body blocks* respectively.

The encoding rules also influence the structure of the SOAP message. They define how a particular data structure is represented in XML. The different ways to encode the message lead to different type of structures. Thus it is imperative that the client and server have agreed on the encoding rules of the messages.

The advantages of SOAP such as platform and language independence do not come without a price. The coding, decoding and parsing of XML from and for SOAP messages affect the performance of both sending and receiving parties of communication. This leads to SOAP being considerably slower than other middleware protocols such as CORBA (Common Object Requesting Broker Architecture). There are technologies to enhance the throughput of Web Services, but all of these trade accessibility for performance [DEJ02].

2.3.3 Web Service Description Language (WSDL)

W3Cs note on WSDL 1.1 [W3C01] describes WSDL as an XML format for describing network services as a set of endpoints operating on messages containing information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. WSDL is

extensible to allow description of network services and their messages regardless of what message formats or network protocols are used to communicate.

WSDL is utilized on Web Services to handle the need to describe the interface in a structured way amongst standardized communication protocols and message formats. WSDL defines an XML grammar describing network services providing a guideline for the use of the services.

The WSDL service specification can be seen on figure 5, where the abstract definition of ports and messages are separated from their concrete use or instance, allowing their reuse. A port is defined by associating a network address with a reusable binding, and a collection of ports (available methods) defines a service. Messages are abstract descriptions of the data being exchanged, and port types are abstract collections of supported operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding, where the operations and messages are then bound to a concrete network protocol and message format. In this way, WSDL describes the public interface to the web service [W3C01].

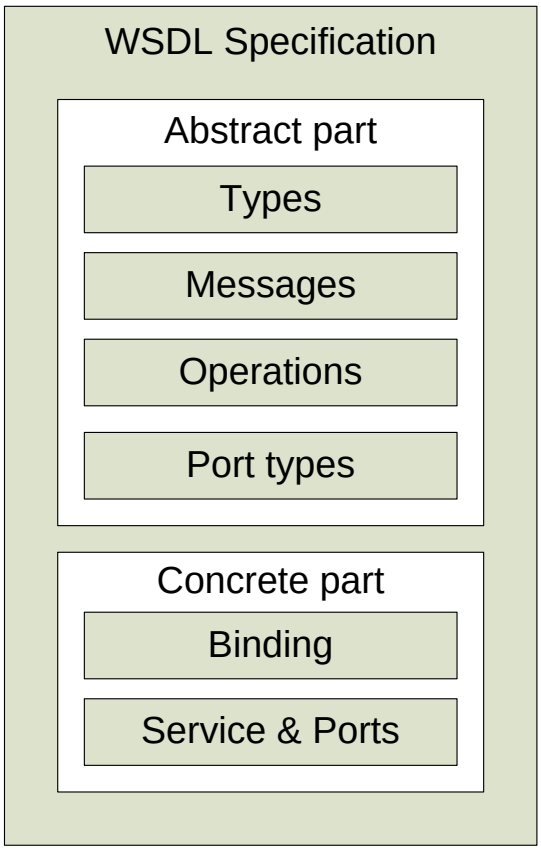

Figure 5: WSDL service specification [ALO04]

W3C does not endorse version 1.1 anymore and recommends the use of version 2.0. It offers better support by accepting binding to all the HTTP request methods (not only GET and POST as in version 1.1) and is much simpler to implement. However many software development kits don't yet support 2.0 but instead use the older 1.1 specification [DHE04]. This is also the case in Visual Studio 2005 which was used for development.

2.3.4 Universal Description, Discovery and Integration (UDDI)

UDDI is an XML-based registry, a distributed database, enabling providers to publish their services and giving requestors a way to discover them by defining how the services interact over the Internet [ALO04]. UDDI registry specifications in respect of service discovery perform two main tasks: they provide support in finding information about services (an advertisement mechanism), and offer a dynamic binding [KIM06].

UDDI is an essential part of public Web Services because of the need to have the service available to broader user base. Nonetheless I will not go into more details about UDDI because the Web Service developed for this bachelor's thesis is private network application and therefore publishing of technical interface is neither needed nor recommended. The required WSDL information will be transferred with direct conversation between the requester and provider. For more information about UDDI I recommend 'Web Services – Concepts, Architecture and Applications' by Alonso Gustavo [ALO04] and 'Introduction to UDDI: Important Features and Functional Concepts' by OASIS [UDDI04].

3 Fitting mobile and modelling

This third chapter describes the environment characteristics of the project, how they affect the development; the problems that rise and solutions for them. There are several factors that need to be taken into account in the development of the service. First and foremost the end-user device of the service will be a mobile device on a light throughput data connection. The features of mobile devices today are limited compared to that of a personal computer. Handling a fully featured model is not practical with them. Neither is the available data communication on construction yards, such as GPRS (General Packet Radio Service), sufficient enough to transfer large amounts of data. Thus we need to define an output for the service that is both satisfactory in detail and still adequately small in size for the medium to handle it in regards of response time and network costs. Secondly, the features of Tekla Structures, the building information modelling software, will greatly direct the development.

3.1 Mobile Device

The system is planned to use a smart phone as the end-user device. Even though the resources (processing power, memory) on such devices have greatly been enhanced during the last few years they still lack compared to that of a stand-alone computers. Joined by the fact that models from Tekla Structures might have tens if not hundreds of thousands of elements make even the smartest phones lacking in terms of raw processing power.

The drawings that are carried on the construction yards today range from small element drawings to massive rolled up drawings of the whole building. Considering the size of the displays on mobile devices today we find that they are way too small to be utilized for viewing these drawings. Regarding any larger drawing, where you could zoom in on the target of interest and zoom out for a bigger picture, the details would not be viewable without a properly sized display. The small displays are not

only problem with today's mobile devices, but the trend is less likely to change in the future making mobile devices less optimal choice for the task at hand.

Although the mobile devices support the viewing of basic imagery formats in general they do require specialized software to be able to view CAD-files such as DWG. The software available are few in number, platform dependant and expensive.

3.2 Mobile Networks

During the design of the system it was taken into account that the mobile phone networks are used for data transfer between the client and service. Today in Finland the 3G networks, which boast broadband data connections, with 2 Mbps on the downlink are only available in the primary urban areas. The farther off you head from the cities centres the slower your data connection will get, degrading first to EDGE (Enhanced Data rates for GSM Evolution), with speeds up to 384 kbps, and then to GPRS with just 14.4 kbps (kilobits per second) for GSM 1800 networks and 9.6 kbps for GSM 900 networks [GPRS]. The construction yards may locate far away from populated areas meaning that the faster networks will not be available and thus we need to get along with a standard GPRS connection. In order for the system to be functional outside primary urban areas the service needs to be light enough for the slowest connections.

One solution for the networking problem of the system could be situating the server on the construction yard and constructing a wireless local area network (WLAN) as the last communication link increasing the transfer speeds at the construction yard. A base link would still be needed to share the information with other project participants. Whether there is a sufficient link available is another concern. Again this would increase the number of devices needed to be installed on the sites pre constructions.

3.3 Tekla Structures

The features of Tekla Structures and its API greatly affect the development. Tekla Structures does host a vast number of features from modelling to drawing creation and scripting regarding the development. Latest version 15.0 was used during the testing and implementation.

The vast number of features conflicts with the lack of available documentary. Although only key features are needed, searching them from a full scale design program without access to the required information is a tedious task. An experienced user would be needed to show how and what can be done. Because there was no access to required information and no such professional available the help files with class descriptions were the best at hand.

Because the service uses Tekla Structures API it is an extension limiting its portability. The service cannot function without Tekla Structures and swapping the design program renders most of the code broken leaving only the basic logic portable. Tekla Structures also requires Windows and the API is based on .NET meaning that the service has to be done for Windows. This is more of a detail than a problem because the system isn't meant to be portable.

4 Potential types of imagery

One of the main problems for this thesis is defining the output of the service in view of environment restrictions i.e. what kind of models and or images can be exported from the design software and transmitted to the client? The options available can be divided into three categories increasing in size, detail and complexity respectively: fixed image, partial model and full model.

4.1 Screenshot

Screenshot stands for an image in a basic format such as JPEG. It is an image of the construction or a single element where the view can not be rotated or the model itself manipulated in any way at the client end (see Figure 6).

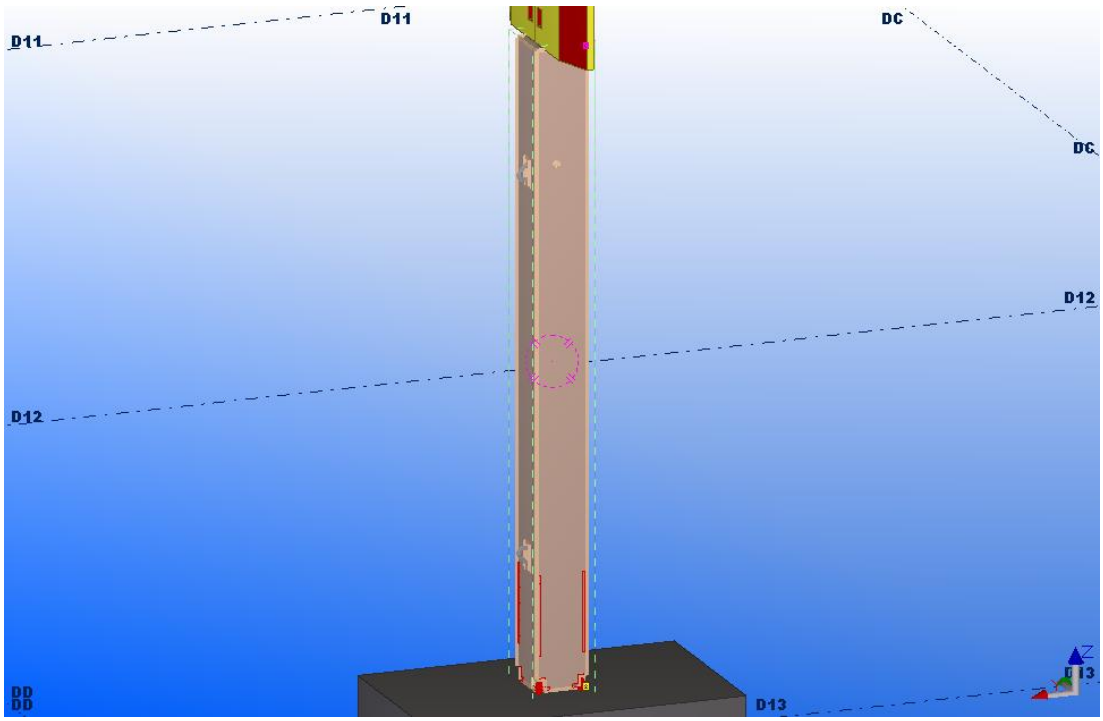

Figure 6: Screenshot of adjoining elements from Tekla Structures

The most definite advantages screenshots have over the other options are the accessibility and the relatively small size. There is not any need for additional

software to view them on the mobile devices and the small size means faster transfer over the medium and smaller processing load for the mobile devices.

Because the images are fixed, only showing a single element or from a single angle and distance, it is compulsory to fetch a new image every time different element or view is needed. This leads to increased use of the network when the client requires images of a different element or view. It also increases the workload at the service point as the service has to handle a lot more interactions.

The creation of the image has to be automated meaning that the service must have proper features to handle the model automatically. To produce a satisfactory screenshot we need: 1) methods to locate the scanned element on the construction plan and 2) to automatically adjust the view, so that the desired part of the model or element is viewed in a non-obstructed way. Whether the image is viewed from the wanted side (wall element – inside or outside?) or angle leads to numerous different variables needed increasing the complexity of the method adjusting the viewpoint. If a solution could be found implementing it with the API could prove to be very complicated and might not be in the scope of this thesis. This fact alone makes screenshots forbidding choice.

4.2 Partial and full model

Simplest solution implementation-wise would be to export the model itself as is with all the elements and details it holds (Figure 7). The service would only act as a file transfer service. The model could be viewed from any angle, zoomed in on the required component and maybe even manipulated to reflect the real life situation if differences were noticed. The network connection would only be stressed once for each location when the model is transferred to the device.

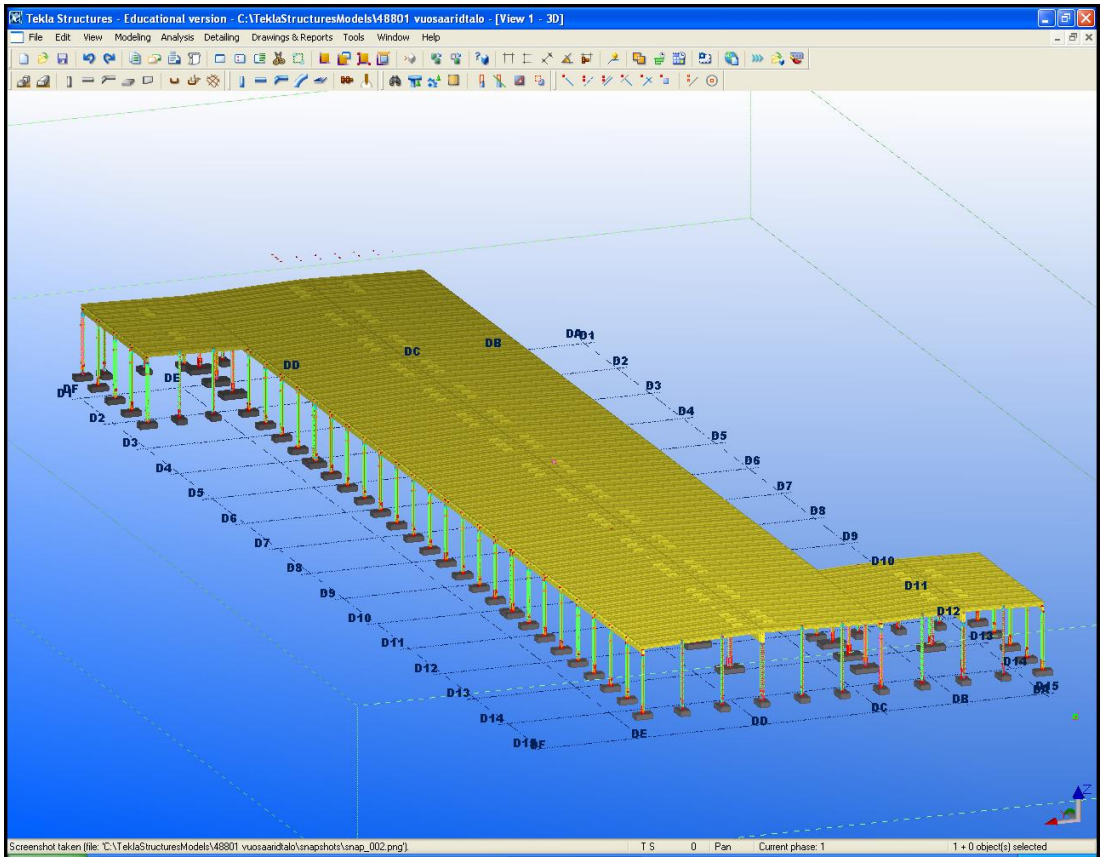

Figure 7: A Tekla Structures model and user interface

Mobile devices today are impractical for viewing full scale models. The displays are small, processing power and memory limited and the medium usable on the construction yards too light. If the loading times are too long the quality of the service will suffer and usability hinder. Specialized software is needed to view and manipulate the most common (DWG, DXF, DGN) modelling formats.

The model queried could be lightened by using a wireframe model, a partial model or a 3D model of a single element (see figure 8). This could enable the viewing of it on a resource restricted device. The small display might only be enough for single elements. The wireless medium might be able to transfer these in a reasonable time when queried. Using partial model the size of the file would be smaller but the number of queries would increase leaving it in between of screenshot and full model solutions.

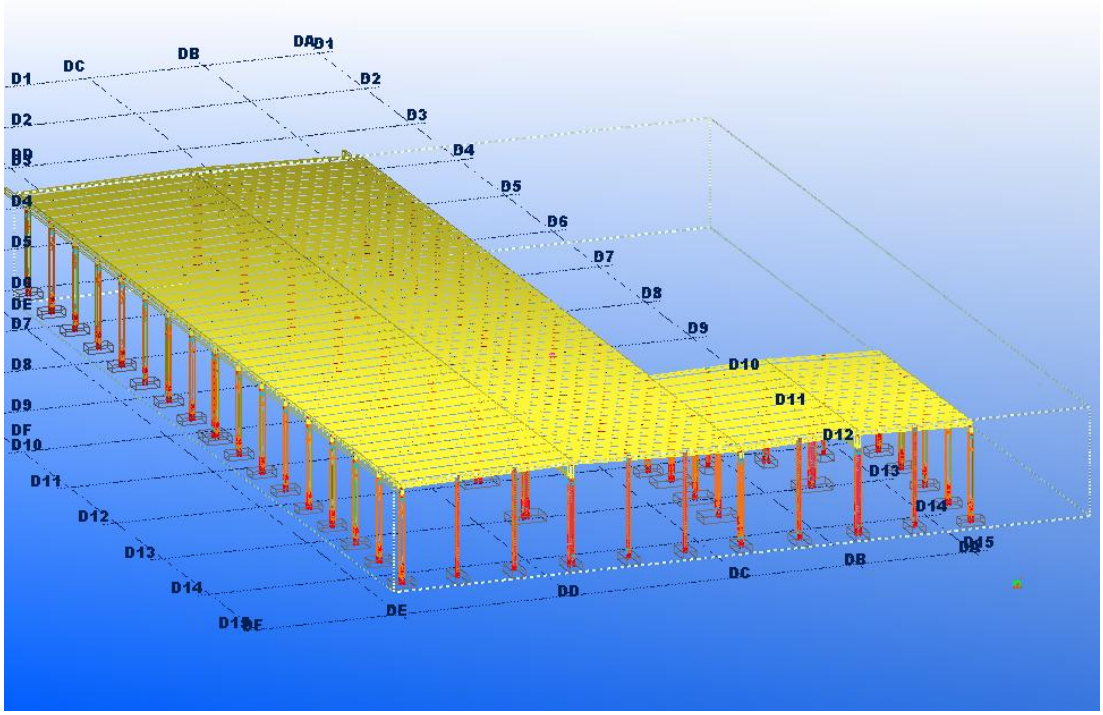

Figure 8: Partial wireframe model

The figure 9 shows the presumed positioning of the different output methods concerning the file size and number of queries related to bandwidth usage. The placement is not definite on the scale as a partial model (vector graphics) can consist of only few objects and it can be a lot smaller relative to file size than a screenshot (raster graphics) in a non compressed form. This does on the other hand lead to the need of additional component which compresses the screenshots to the required size and type when needed.

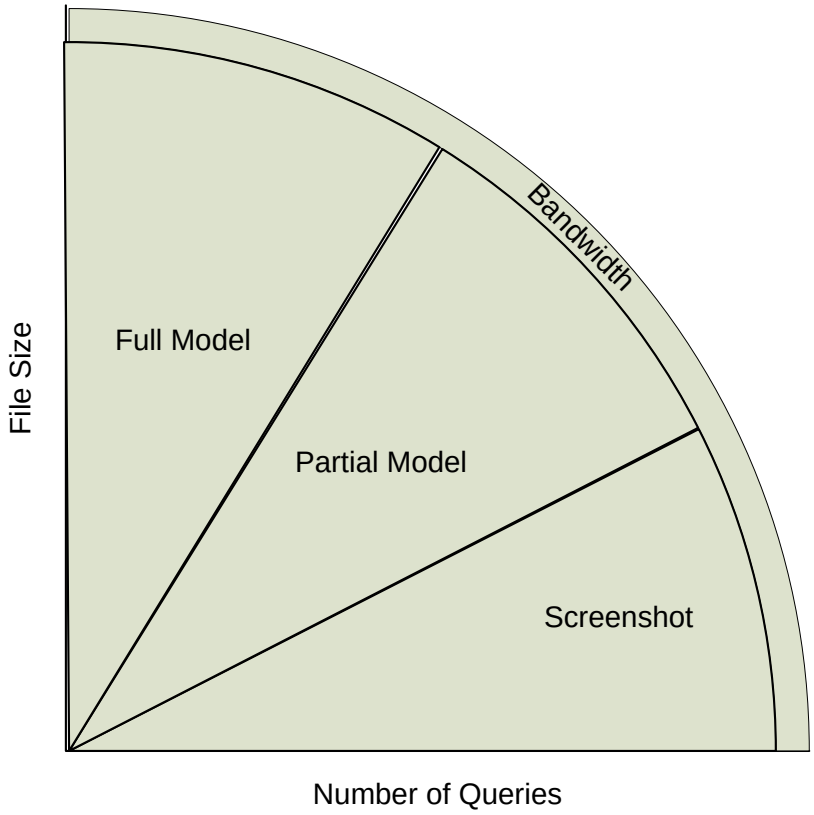

Figure 9: Bandwidth usage by output type

These observations lead to one of the prime contradictions: is it sensible trying to manage this kind of service with a mobile phone or PDA (Personal Digital Assistant) or should a laptop be used as a client? In what situations would the mobile phone be usable? This will be discussed in more detail in chapter 5.

5 Implementation

The chapter starts by describing the requirements and operation of the service and continues with a discussion about the problems and observations made during the course of implementation. A view of the demo system can be found in chapter 6.

The composition of the solution, which can be seen in figure 10, consists of three components: 1) *a Web Service,* which handles incoming requests and data transfer, 2) *an interface,* which links Tekla Structures API and the service and 3) *macros*, which export the desired type of visual information from the Tekla Structures model.

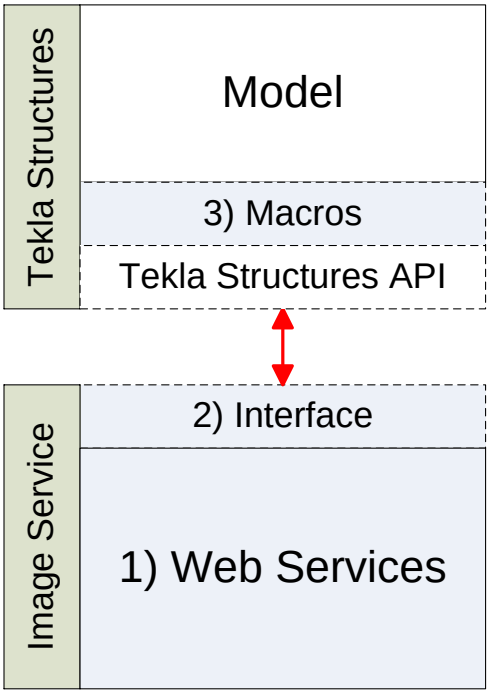

Figure 10: Composition of the service

Web Service is the foundation of the service providing necessary features to serve incoming service requests: interface described in WSDL, functionality for SOAP based XML communication and a house for our program logic to reside. Visual Studio has a feature to automatically generate the Web Services interface based on the exposed methods. One can create them manually, but it is often better to let the development studio do it, because the specifications can get quite complex and still

need to be precise in order to function. The service is a passive service waiting for incoming service requests; it does not actively connect to outside clients. Mobile Service acts as a client for the service. No other connections are needed.

The service uses methods of Tekla Structures API to access the functionality of the design program to operate the model. The interface is .NET compatible. With Visual Studio the interface can be accessed by including a reference to *Tekla.Structures.Model.dll*, residing in:

C:\TeklaStructures\VERSION_NUMBER\Int\bin\plugins.

Once a reference to the interface has been added it can be used like any other class in the project by adding '*using Tekla.Structures.Model;'* amongst the class definitions of a personal class. The documentation about class definitions can be found in the folder Tekla Structures is installed in:

C:\TeklaStructures\VERSION_NUMBER\nt\help\enu\TeklaStructures..chm*

The macros present us with the functionality of exporting the desired image files. Implementing the same functionality with the API would be a lot more complicated. Tekla Structures supports recording of macros from the user interface and running them with a command from the API. The macros used need to be copied to their respective folders to enable Tekla Structures to find them. The base folder for macros is:

C:\TeklaStructures\VERSION_NUMBER\environments\common\macros

5.1 Requirements

The service has to be installed on the same computer as the modelling software due to Tekla Structures application programming interface which only offers interaction on the same computer [REI07]. Because Tekla Structures API is based on .NET the service was implemented using Microsoft's technologies. Visual Studio was used as the development studio and C# as the coding language.

Visual Studio produces Web Services for .NET framework which is only available for Windows at the time of writing (apart from incomplete open source projects) so

the service requires Microsoft's IIS-server (Internet Information Services). This has been part of Microsoft's operating systems from Windows 2000 forward. The firewall of the server needs to be configured to allow access to the service.

On the thesis case the IDs on the RFID-tags on the construction elements are different from the IDs generated by Tekla Structures. Thus we need to have a database that connects these two ids and swaps them for the service. This Mobile Service and the database it connects to have been implemented on a previous 'Mobilding'-project [MOB07] at Lappeenranta University of Technology's Department of Information Technology. The Mobile Service acts as a relay in communication with a mobile client and presents us the functionality of switching the RFID-ID to the Tekla Structures –ID.

5.2 Operation

As seen on the system architecture on figure 11 the service has an open WSDL described interface (see appendix 1) through which it communicates with XML based SOAP messages (see appendix 2). It communicates with the Mobile Service that relays the requests from a mobile device on a construction yard.

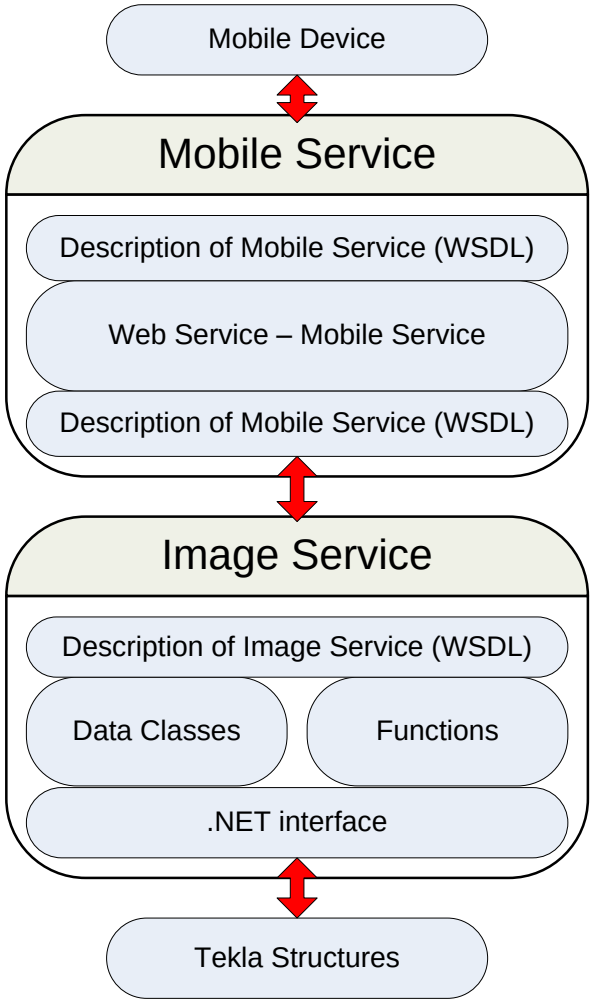

Figure 11: System Architecture

The process of the service can be seen on figure 12: On request the service will create a connection to Tekla Structures using the API, open the corresponding model, locate the element, run the required macros that create the requested imagery file, encode the data with base64 and return the resulting string to the requester which will decode and save it as the requested file.

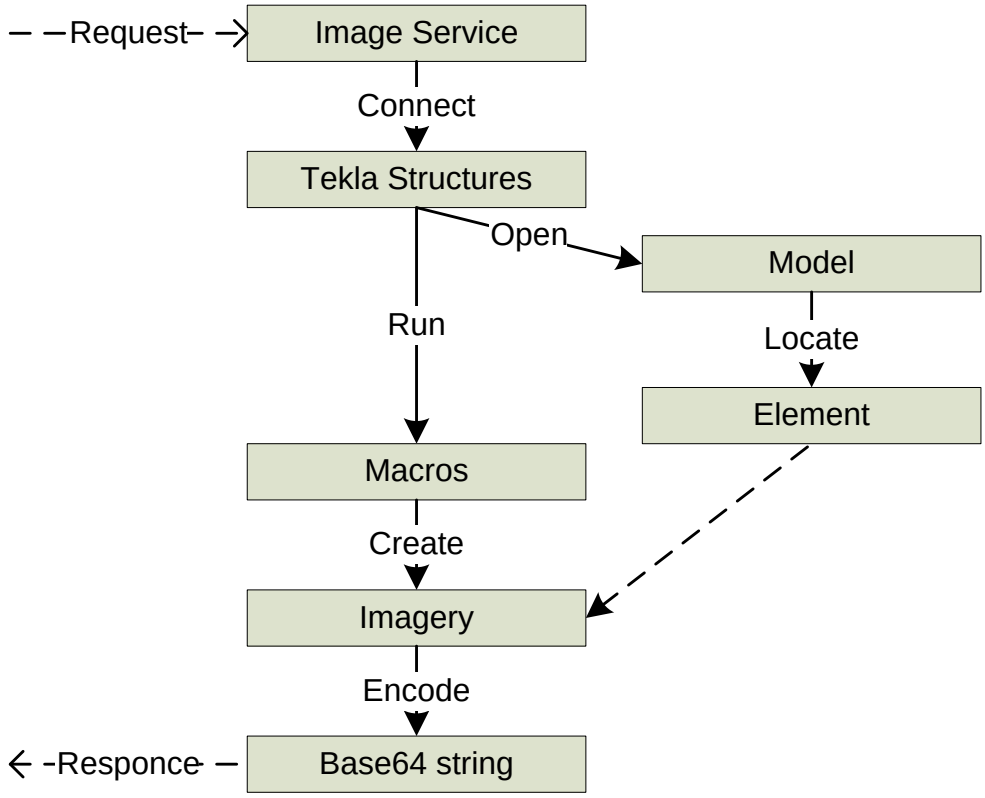

Figure 12: Process chart of the service

5.2.1 Communication

The service requires three attributes as an input when called (see figure 13):

1) *string project_name*, 2) *int ID*, 3) *int type*.

The *project_name* defines the name of the folder where the project we are working on resides enabling us to open the right project. The element *ID* is used to locate and select the element in the model. The *type* defines what kind of output is requested.

On success the service returns a base64 encoded string that has to be decoded at the client end to retrieve the information in a viewable form. The client needs to be aware what type of file it is requesting. In the case of a failure a text string with the information about the occurred error is returned.

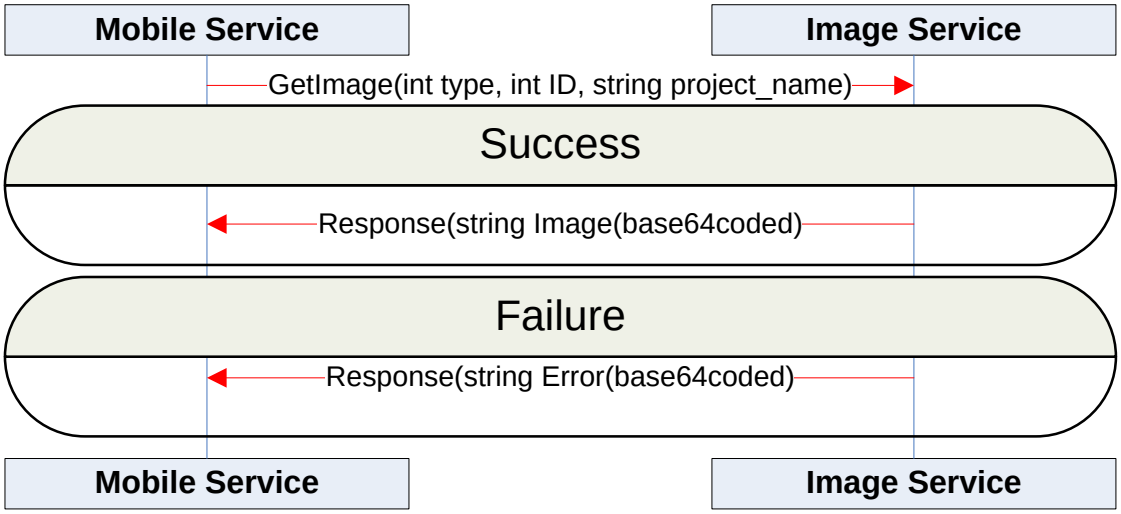

Figure 13: Message sequence chart of the service

5.2.2 Program flow

This part will go through the program flow of service step by step. Every step was coded in try-catch sequences to enhance error tolerability. Whenever an error is occurred the current state and probable reason is returned to the requester and execution of the request quit. All classes, methods, attributes and libraries they belong to are described in more detail in appendix 3. Tekla Structures will be referred as TS in this chapter for the sake of repetition.

When called the service first tries to connect to TS:

```
Model myModel = new Model();
myModel.GetConnectionStatus();
```

This creates a handle for the model and then checks if the handle has connection to TS. A failure results most likely because TS is not running on the same computer with the service.

If a connection is established through the API the service opens the requested project:

ModelInfo myModelInfo = myModel.GetInfo();
myModelInfo.ModelName.Length
myModelInfo.ModelName

If a project is already open on TS (*ModelName.Length!=0*) the service checks if it is the wanted project comparing *myModelInfo.ModelName* to the *project_name* variable of the request.

Otherwise the requested project will be opened:

myModel.Open(path);

'C:/TeklaStructuresModels/PROJECT_NAME' is the default folder for TS projects.

The requested element can then be located from the open model:

ArrayList myObjectList = new ArrayList();
Identifier myID = new Identifier(ID);
myObjectList.Add(myModel.SelectModelObject(myID));

This part is a bit complicated. TS handles selected objects in arraylists so we need to create one. Then we need to create an identifier which TS understands. Last we order TS to add the element-object identified by *ID* and returned by *SelectModelObject()* to *myObjectList*. To make things a little bit more interesting in order to use the *SelectModelObject()* and *Identifier()* classes we need to add a new reference to *Tekla.Structures.dll* in addition to the original *Tekla.Structures.Model.dll*.

We still need to highlight the object in the user interface in order to use macros:

Tekla.Structures.Model.UI.ModelObjectSelector mySelector = new
Tekla.Structures.Model.UI.ModelObjectSelector();
mySelector.Select(myObjectList);

This creates a *ModelObjectSelector* object which is then finally able to select the element listed in *myObjectList*.

The element with the requested ID should now be visually highlighted in TS user interface and ready for use:

```
myModel.RunMacro("MyMacro.cs");
myModel.IsMacroRunning();
```

The code above runs a macro in TS designated in *MyMacro.cs* file. By polling with '*myModel.IsMacroRunning();*' we make sure to wait until the completion of the macro in the TS side is done before continuing the execution at the service side. A switch-case –structure is used to switch between different macros exporting different types of images. The macro is selected considering the request (*int type*) and element properties (see chapter 5.3). Macros will be covered in more detail in the next chapter 5.2.3.

The requested file has now been exported from TS to *C:\TeklaImageViewer*. The file content will then be read and encoded to base64:

```
base64string = Convert.ToBase64String(binaryData, 0, binaryData.Length);
```

Convert.ToBase64String() is a standard function included in *System* namespace of .NET. The output *base64string* will then be returned to the requester completing the service.

5.2.3 Macros

The macros export the image or model from Tekla Structures for the service. Different types of images, drawings and models can all be made available in the service by creating a macro performing the necessary steps to produce it. Tekla Structures presents a vast number of possibilities, from which two options were implemented. The figures figures 14 and 15 are images of the same metal element: 14) single-element drawing (cast-unit drawing) in DWG format, 15) screenshot of

single element with joining elements, in PNG format. Discussion about other type of exports can be found in chapter 7.

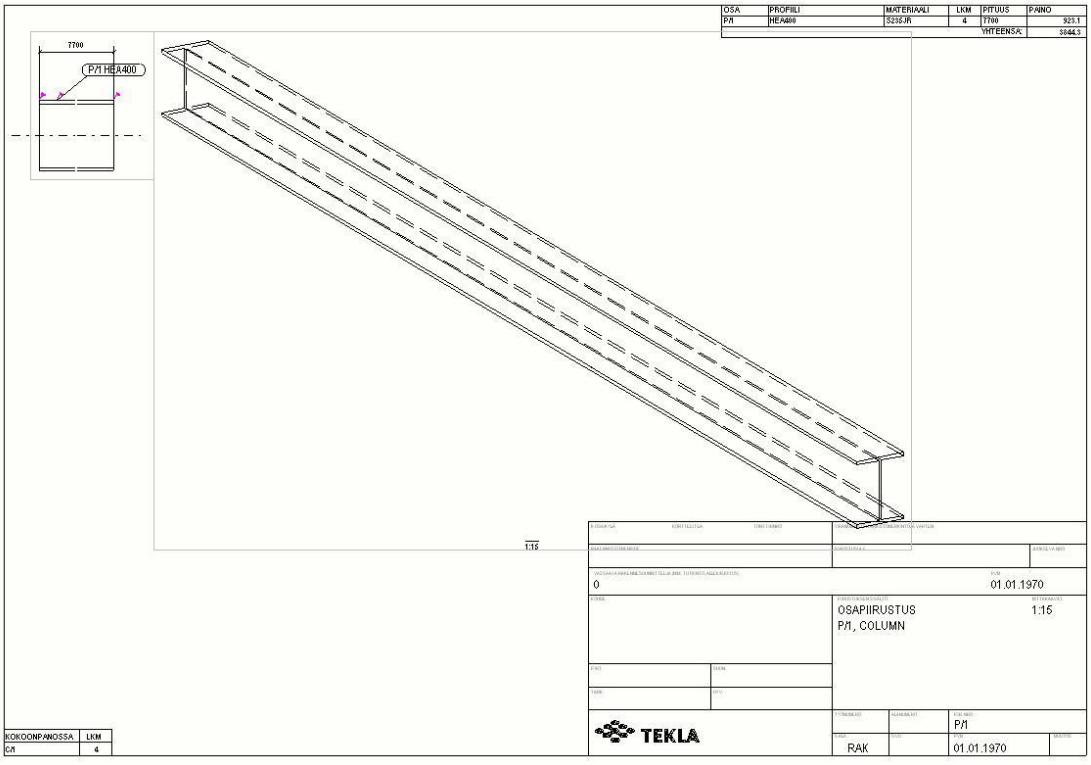

Figure 14: Single Part Drawing

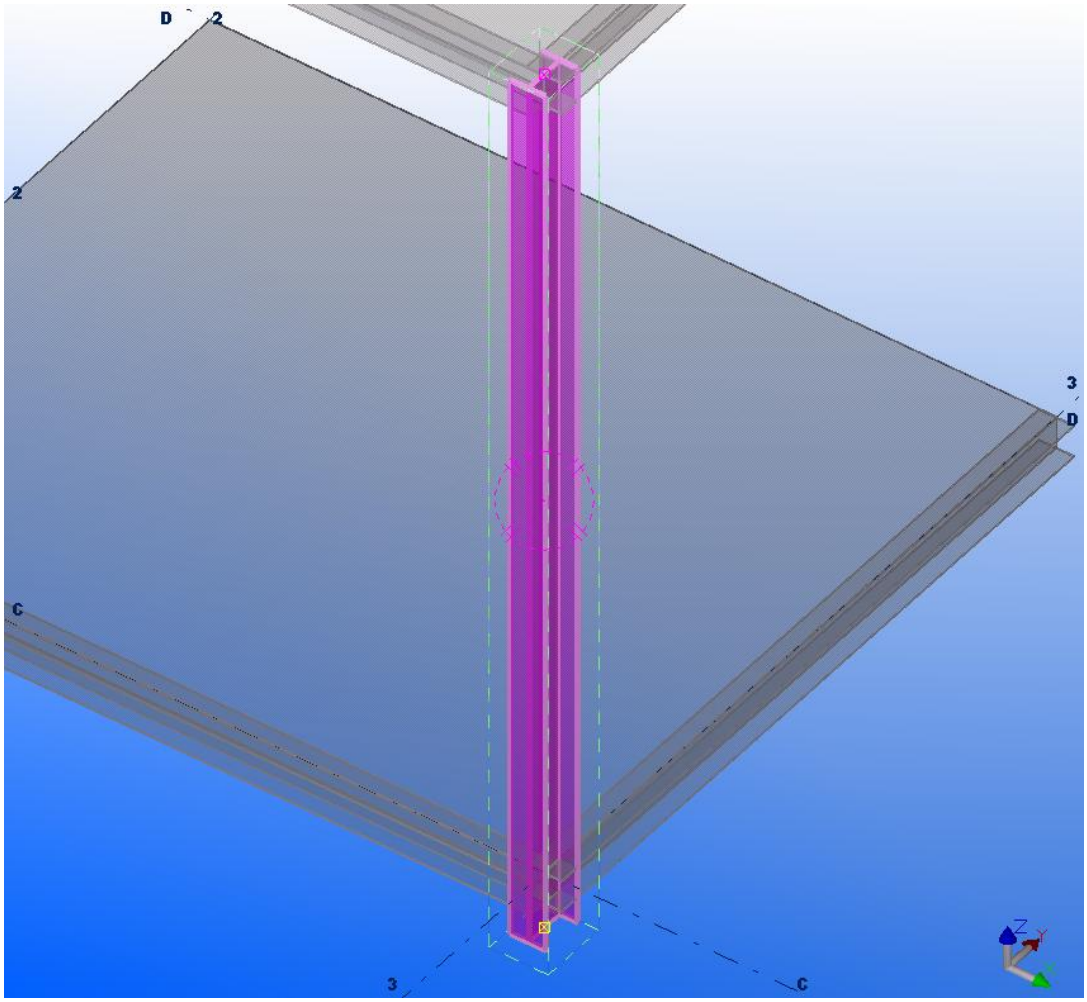

Figure 15: Screenshot of single element with joining structures

Tekla Structures has a built in script builder to record macros in the user interface. It can be accessed from the top menu of the UI (User Interface): *Tools* > *Macros*. Typing in a new name for a macro and pressing *Record* saves all subsequent actions in the user interface to a script file which can be run later in order to reproduce the operation. This helps the creation of macros as one does not have to get to know the Tekla Structures script language, commands and engine in depth.

After experimenting with the capabilities of Tekla Structures and keeping in mind the restrictions of a mobile phone, a conclusion was made to export a screenshot of the requested element and joining parts and an element drawing in DWG format with a three dimension view of the requested element and some additional information like material, etc. The drawings do not show the surroundings or interaction with other elements. The main benefit pro using them is the compact size and clearness of

details of the single element. The screenshot on the other hand provides information about the immediate surrounding elements and how the queried element situates amongst them.

The drawing script performs the following tasks: 1) Enable 3D view for single part drawings, 2) Generate single part drawing, 3) Export the drawing, 4) Delete the created drawing and 5) Disable 3D view for single part drawings. The script cleans all traces of its use i.e. removes created drawings, closes windows and reverts changes made to options. The macro saves the file to *C:\TeklaImageViewer* folder for the service to find. The screenshot script creates a close-up view of the element: *View > 'Fit work area' > 'To selected parts in selected view'* and then exports a screenshot to the *C:\TeklaImageViewer* folder and reverts the view.

5.3 Problems, observations and solutions

There were several problems during the implementation ranging from small performance defects to full scale bugs preventing the operation of the system. Most of them could have been avoided with better experience and knowledge of Tekla Structures.

There are no features for *handling dialogue windows* in the current build of service. An auto save of the model is created automatically by Tekla Structures if the model is open and operated upon long enough. When opening the model next time a dialogue window pops up: '*load the auto save or the original save?'* This halts the execution, because the dialogue needs confirmation before the loading can actually happen. A solution for this would be disabling the auto save or coding in functionality for detecting dialogue windows and reading their information. The latter might be complicated to code and as the option for auto saving was not found no solution was implemented.

There is an overall *lack of failure messages* in Tekla Structures. A lot of actions do not give any feedback to user even if they are not carried out successfully. The creation of a single-part or cast-unit drawing does not succeed if there is another

drawing of the same element available. It does not matter if the result would be different from the available one. Without any way of knowing if the task is carried out or not during the execution of a macro it is very hard to pinpoint the cause of the problem. To solve the problem we would need to discard macros altogether and only use classes and methods provided by the API, but this would complicate and increase the amount of coding considerably.

Tekla Structures has *separate modules* for handling models and drawings. Exporting the created drawing from the list of drawings worked when using the UI manually, but running the same commands from a macro did not perform the same task. I found out that there are actually different folders for macros for both of these modules and it seems that switching between the modules during the recording of a macro breaks the script being created. A workaround was found by recording the macro in three parts and then manually putting them together in an editor: 1) Modelling: create drawing and open drawing module, 2) export the drawing and close the drawing module, 3) finish the actions in the modelling side. This increased the processing time of the service because the drawing needs to be opened for viewing before the exporting can be done by the macro. Again the lack of error messages complicated locating the problem. An error should occur when user tries to record macro ranging over these two modules or the modules should only use one and same folder and engine to run their macros.

Single part drawings can not be created of cast unit type objects. Instead a cast unit drawing needs to be created of these elements. This effectively means that we need to detect the material of the located element:

```
ModelObject test = (ModelObject)myObjectList[0];
test.GetReportProperty("MATERIAL_TYPE", ref objectMaterial);
```

We create a new *ModelObject* named *test* which becomes a copy of the selected element. Then we save *MATERIAL_TYPE* named attribute of *test* to *objectMaterial*. If the *MATERIAL_TYPE* is *CONCRETE* we know the element is cast made and thus requires cast unit drawing instead of single part drawing. Another macro was created

to use the *'create cast unit drawing'* feature of UI. Even though the problem is easily solved Tekla Structures lacks the automation of switching between correct drawing types and even lacks error message notifying that the creation of the drawing failed.

A complete model made with Tekla Structures should include all drawings needed to construct the represented building. There were *no drawings* for the model used in development. A small model was created for faster loading times when testing the service during development. As there were no drawings the macro was made to create the required drawing, export it and delete it so that the excess drawings would not flood Tekla Structures. So the macro presumes that the created drawing is the one we want to export, meaning that if there are any other drawings a wrong one could end up being exported and/or deleted. On hindsight the right drawing should just be searched and exported – no creation should be necessary. This leads to second primary observation about the whole thesis – the absence of specification and definition regarding the possibilities, definitions and wanted outcome.

6 Demo system

The demo system was installed on a personal computer with Intel Pentium 4 2.40GHz processor and 1GB of RAM (Random Access Memory) running a Microsoft Windows XP Pro, Service Pack 2 and Internet Information Services (IIS).

The required Mobile Service, relaying the data between the service and client mobile phone, as well as the mobile phones software was provided by the Mobilding project [MOB07]. The mobile phone, the software used in it and the Mobile Service relaying the commands are all viewed as black boxes.

The phone sends the request to Mobile Service, which swaps the ID read from the element to a Tekla Structures ID and relays the request to the service. The service produces the necessary imagery and returns it to the Mobile Service, which relays it to the mobile phone, where the data can be decoded (base64) and the file restored for viewing.

6.1 Unable to connect to Tekla Structures API

In Visual Studio Web Service applications can be tested with a virtual development server which provides local environment for debugging and testing. The service was fully functional with the development server. After publishing the service to IIS and after the necessary configurations were done the service could be accessed, but the Tekla Structures API could not be reached anymore.

By default the .NET Web Service applications are executed by *aspnet_wp.exe*, a worker process which is run by the ASPNET user account. This user account is created during the installation of .NET on IIS and is used for controlling access rights of web applications. On the other hand Tekla Structures was run with an administrator user account. Thus the calls made by the ASPNET user account could not reach administrator user run Tekla Structures API. This is perfectly rational because allowing a user to access an API of another users process could be

hazardous regarding information security. To fix the issue the worker process' user account had to be changed. There were two options: 1) Impersonating another user account and 2) Changing the user account of the worker process.

IIS and .NET Web Services offer impersonation in order to start processes as other users. But with a .NET Web Service a process always inherits the user credentials of the parent process; even if the invoking call is impersonated, the process still starts under ASPNET user account. Thus impersonation did not solve the problem.

Scott Allen [ALL04] describes a method of using Win32 API calls to create and manipulate process threads and their user credentials. The processes started with this method initiate in a non-interactive state remaining invisible but still consuming memory. In our case Tekla Structures requires desktop so this approach does not help us either.

So we need to change the user account on which the aspnet_wp.exe worker process is ran. To achieve this the *machine.config* file at

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG

needs be modified to have *userName* and *password* fields. The system did not accept any of the tried user accounts and their passwords regardless of user rights, but *username=SYSTEM* which switches the worker process to be run under SYSTEM user account was succesfull. A desktop under SYSTEM user account [XPHACK] was then used to run Tekla Structures. Now both the Tekla Structures and the Web Service are run by the same SYSTEM user account and communication works.

6.2 Observations about the demo system

The hardware used was rather insufficient and had quite hard time handling Tekla Structures and the model in a moderate time. Better hardware suitable for modelling is highly recommended. Opening the small model with under 100 objects took approximately 15 seconds and a medium sized with under 20 000 objects well over one minute. The times show that opening the model pre-use of the service is better option than letting the service do the opening.

The processing times for the tasks (screenshot/drawing) on the two different models can be seen on chart 1. As the hardware used for testing is impractical for modelling these values are only directional showing that the processing time increases immensly with the size of the model. Processing time can be optimized in the future development by for example removing the creation of drawings and just exporting an existing one.

Model	**Small** (<1000 elements)	**Medium** (~20 000 elements)
Screenshot	14s	1min 37s
Drawing	41s	3min 38s

Chart 1: Processing times of the service during tests

The service needs further development before deployment. Additional features and testing are required to produce a version for other than demo purposes. The return value should be redefined from string to a class. The class should include a variable for determine if the request is successfully completed and a string containing the file extension in addition to the base 64 encoded data(see figure 16). This would provide the client more information about the result of the operation. Additional output types should be made available and security measures such as user identification implemented.

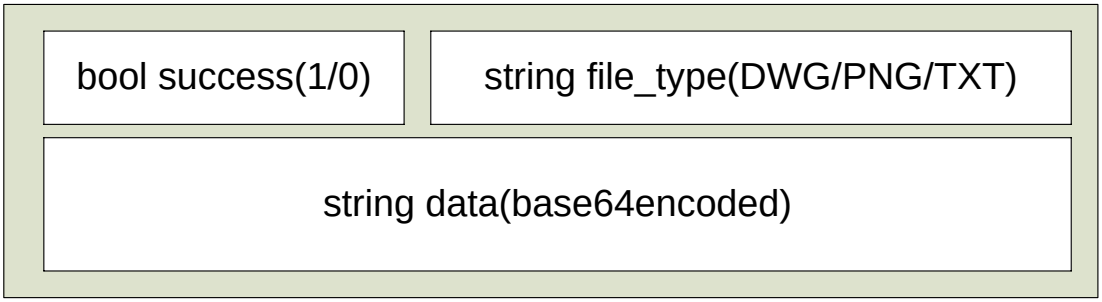

Figure 16: Example of a return class

7 Discussion

Using digital information over plain paper documents on the construction sites has clear benefits. It is more accessible, available and enduring when used with the right tools. A collection of services need to be developed in order to make the digital media available and the right tools need to be chosen to benefit from them. This thesis only provides one way of relaying data from design software to a mobile phone, which is by far from perfect.

Web Services trade performance for accessibility and availability. The XML structures, which describe attributes and functions in a coherent way, need to be parsed at both ends of the communication increasing overhead. When using a mobile device and wireless data communication, which are lacking compared to a stand-alone computer with a fixed internet connection, we have to question the use of Web Services. The Mobile Service could use standard transfer methods like TCP or FTP (File Transfer Protocol) instead. Other middleware protocols such as CORBA should also be considered. This would improve response time and reduce the amount of redundant processing.

The mobile phone seems to be useful in handling small amounts of information i.e. small drawings and pictures. Element drawings are usually provided for every element on the construction leading to a big binder full of drawings. Using the created service for fetching the required drawing on the go would be beneficial. The user would not have to browse through binders only to find the required one. On the other hand the small sizes of the devices nowadays are a trend less likely to change. With the small size comes a small display which is hardly viable for viewing any larger drawing or detailed picture. These observations lead to one of the prime contradictions: is it sensible trying to manage this kind of service with a mobile phone or PDA at all or should a laptop be used as an end client?

A laptop would increase the number of devices that need to be carried around, but would provide us with sufficient processing power for manipulating the model at the construction site. There are features in Tekla Structures that could be employed to get visuals of the 3D model to the laptop such as Web Viewer, which can be used to export the current model to be viewed with a web browser. The display on a laptop would be sufficient enough for viewing all the drawings of the structure. The drawings could either be stored at the laptop or a central server from where they would be queried when needed.

Further research is needed. The needs of the contractor, benefits for the building master and inspector and full potential of Tekla Structures need to be mapped out. Based on these assessments sufficient requirements for the project could be set. Without the knowledge of what is practical and useful on the construction sites the development of these services will only produce deficient results. A topic for master's thesis would be to carry out said surveys and to further develop the service in the direction discovered.

8 Conclusions

The aim of the thesis was to explore the preliminary possibilities of exporting digital imagery from a construction design software to a mobile phone. This was done by developing a Web Service which operates a BIM software's model through an application programming interface in order to produce imagery for the mobile phone. The API can be used to operate the construction model and to export various types of imagery. A Web Service can be utilized to transport the data, but other options should be explored to find more suitable solution for mobile phones.

The basis of the end client being a mobile phone has to be questioned. The small sized display, insufficient processing power and lack of software make mobile phone unoptimal choice for viewing large and detail rich visual information. Only small drawings or imagery can be viewed in a sufficient manner. To utilize the built service to its full effect, removing most if not all paper drawings from the construction site, a fully featured and environment resistant laptop is recommended to be used as a client.

Before further development, the true possibilities, needs and potential of the service need to be explored. Specifications and requirements detailed and defined and a clear outcome of the resulting features composed. The surveys providing this information should have been either available for this thesis or a main object for it. Further research is needed.

9 Bibliography

[ALL04] OdeToCode.com. Thursday, October 28, 2004 10:04 PM http://odetocode.com/Blogs/scott/archive/2004/10.aspx

[ALO04] Alonso, Gustavo. 2004. Web Services – Concepts, Architecture and Applications. ISBN 3-540-44008-9

[ASPNET] Microsofts ASP.NET forums. http://forums.asp.net/t/1377201.aspx 28.8.2009

[DEJ02] de Jong, Irmen (et.al.). Web Services/SOAP and CORBA, April 27, 2002, http://www.xs4all.nl/~irmen/comp/CORBA_vs_SOAP.html#2 . 20.8.2009

[DHE04] Dhesiaseelan Arulazi 2004. What's New in WSDL 2.0 http://www.xml.com/pub/a/ws/2004/05/19/wsdl2.html , 13.8.2009

[EAS07] Eastman Charles. 2007. What is BIM? , http://bim.arch.gatech.edu/?id=402, Georgia Tech - BIM Resources

[EBU] EBU middleware report Tech 3300. http://tech.ebu.ch/docs/tech/tech3300.pdf , 13.8.2009

[GPRS] Mobile Phones UK: GPRS (General Packet Radio Service), HSCSD & EDGE. http://www.mobile-phones-uk.org.uk/gprs.htm . 2001 - 2009, Landmark Internet Ltd. 20.8.2009

[XPHACK] Login 'SYSTEM' user-account Microsoft Windows XP. http://www.youtube.com/watch?v=cri-sCe2av0 , 3.9.2009

[HÄM08] Hämäläinen, Harri., et.al.. Mobiiliteknologioiden hyödyntäminen rakennusteollisuudessa. ISBN 978-952-214-639-7 (nid.), ISBN 978-952-214-640-3 (PDF), ISSN 0783-8069. Lappeenranta 2008.

[KAL06] Kallonen, T. Masters Thesis: RFID-tekniikan käyttö betonielementtien tunnistamiseen. Lappeenranta University of Technology, Lappeenranta 2006.

[KIM06] Kim, Sergey. Masters Thesis: Composing Web Services. Lappeenranta University of Technology, Lappeenranta 2006

[LEU08] Leung Sze-wing., Mak, Stephen., Lee, Bill L.P. Using a real-time integrated communication system to monitor the progress and quality of construction works. Automation in Construction 17 (2008) 749–757

[MCK98] McKinney, Kathleen,. Fischer, Martin. Generating, evaluating and visualizing construction schedules with CAD tools. Automation in Construction 7_1998.433–447

[MOB07] Hämäläinen, Harri. Happonen, Ari. Mobilding WIKI. 30.8.2007. Not available to public.

[MSDN] Microsoft Developer Network, http://msdn.microsoft.com/en-us/default.aspx, 14.7.2009

[NAV07] Navon, R. Research in automated measurement of project performance indicators. Automation in Construction 16 (2007) 176–188

[PHIL] Phillips, Addison., Web Services and Internationalization, http://www.inter-locale.com/whitepaper/multilingual/ml73-ws-20050524.xml , 16.7.2009

[REI07] Reisbacka, Teemu. Bachelor's Thesis: Using Web Service as an interface in data transfers. Lappeenranta University of Technology, Lappeenranta 2007

[ROJ99] Rojas, E.M., A.D. Songer, Web-centrics systems: a new paradigm for collaborative engineering, Journal of Management in Engineering 15 (1) (1999) 39–45.

[TAM99] Tam, C.M. Use of the Internet to enhance construction communication: Total Information Transfer System. International Journal of Project Management Vol. 17, No. 2, pp. 107±111, 1999

[TEKLA] Tekla Web Site. http://www.tekla.com/ 5.8.2009

[UDDI04] Introduction to UDDI: Important Features and Functional Concepts, http://uddi.org/pubs/uddi-tech-wp.pdf , 22.7.2009

[W3C00] Box, Don, et.al. W3C: Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ , W3C Note 08 May 2000

[W3C01] Christensen, Erik, et.al. W3C: Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl , W3C Note 15 March 2001

[W3C04] Booth, David, et.al. W3C: Web Services Architechture, http://www.w3.org/TR/ws-arch/, W3C Note 11 February 2004

[W3C07] Chinnici, Roberto., et.al. W3C: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, http://www.w3.org/TR/wsdl20/, W3C Recommendation 26 June 2007

[WAN08] Wang, Lung-Chuang. Enhancing construction quality inspection and management using RFID technology. Automation in Construction 17 (2008) 467–479.

[WEN08] Wenfa, HU. Integration of Radio-Frequency Identification and 4D CAD in Construction Management. Tsinghua Science and Technology. ISSN 1007-0214 25/67 pp151-157, Volume 13, Number S1, October 2008

Appendix 1: WSDL interface description

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www2.it.lut.fi/TeklaImageService"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://www2.it.lut.fi/TeklaImageService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Tekla Image
Service</wsdl:documentation>
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://www2.it.lut.fi/TeklaImageService">
      <s:element name="GetImage">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="imgType"
type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="elementID"
type="s:int" />
            <s:element minOccurs="0" maxOccurs="1"
name="projectName" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetImageResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetImageResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetImageSoapIn">
    <wsdl:part name="parameters" element="tns:GetImage" />
  </wsdl:message>
  <wsdl:message name="GetImageSoapOut">
    <wsdl:part name="parameters" element="tns:GetImageResponse" />
  </wsdl:message>
  <wsdl:portType name="ServiceSoap">
    <wsdl:operation name="GetImage">
      <wsdl:documentation
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns image/model
from Tekla Structures.</wsdl:documentation>
      <wsdl:input message="tns:GetImageSoapIn" />
      <wsdl:output message="tns:GetImageSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
```

```
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
/>
    <wsdl:operation name="GetImage">
      <soap:operation
soapAction="http://www2.it.lut.fi/TeklaImageService/GetImage"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
/>
    <wsdl:operation name="GetImage">
      <soap12:operation
soapAction="http://www2.it.lut.fi/TeklaImageService/GetImage"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service">
    <wsdl:documentation
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Tekla Image
Service</wsdl:documentation>
    <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address
location="http://localhost:1265/WebSite/Service.asmx" />
    </wsdl:port>
    <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
      <soap12:address
location="http://localhost:1265/WebSite/Service.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Appendix 2: SOAP 1.2 request and response

```
POST /WebSite/Service.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetImage xmlns="http://www2.it.lut.fi/TeklaImageService">
      <imgType>int</imgType>
      <elementID>int</elementID>
      <projectName>string</projectName>
    </GetImage>
  </soap12:Body>
</soap12:Envelope>
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetImageResponse
xmlns="http://www2.it.lut.fi/TeklaImageService">
      <GetImageResult>string</GetImageResult>
    </GetImageResponse>
  </soap12:Body>
</soap12:Envelope>
```