HAAGA-HELIA
ammattikorkeakoulu

Designing architecture for invoice based automated reporting

Jaakko Hirn

Bachelor's Thesis
Degree Programme in Information and Communications technology
2015

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

30.4.2015

Tekijä(t)
Jaakko Hirn

Koulutusohjelma
Tietojenkäsittely

Opinnäytetyön otsikko
Automatisoidun laskupohjaisen raportointipalvelun arkkitehtuurin suunnittelu

Sivu- ja liitesivumäärä
19 + 1

Designing architecture for invoice based automated reporting

Www-sovelluspalvelut ovat moderni tapa toteuttaa elektronisia palveluita. Monet suunnittelumallit, kuten palvelukeskeinen arkkitehtuuri (SOA) auttavat luotettavan järjestelmän suunnittelussa ja teknisissä valinnoissa.

Opinnäytetyö on tehty osana Yritys x:n Y hanketta. Hanke tähtää luomaan uusia taloushallinnon innovaatioita, jotka automatisoivat finanssitransaktioita.

Tämä opinnäytetyö pyrkii löytämään keskeiset periaatteet Www-sovelluspalvelu arkkitehtuurisuunnitteluun. Nämä periaatteet tunnistetaan ja avataan osaksi tarkempaa tarkastelua. Tarkastelulla yritetään löytää mitä lisäarvoa ne tuovat projektille. Uutta suunniteltua kausiveronpalautus ohjelmaa käytetään esimerkkitapauksena, jotta nähdään miten näitä käsitteitä voidaan ottaa käytäntöön mukaan ja mitkä ovat parhaat käytännöt www-sovelluspalvelun toteuttamiseksi.

Tutkielmassa päädytään siihen, että on olemassa useita projektia edistäviä suunnittelumalleja, jotka auttavat modernin palvelun tuottamisessa. Yleisimmin käytössä olevat mallit ovat laajasti hyväksyttyjä, eivätkä aiheuta suurta vastakkainasettelua.

Asiasanat
www-sovelluspalvelu, palvelukeskeinen arkkitehtuuri, rajapinta, Ohjelmisto suunnittelu

HAAGA-HELIA
University of Applied Sciences

Abstract

30.4.2015

Author(s)
Jaakko Hirn

Degree Programme in Information and Communications

Report/thesis title	**Number of pages and appendix pages**
Designing architecture for invoice based automated reporting	**19 + 1**

Web services are the modern style to produce electronic services. There are a set of Design patterns, such as Service oriented architecture (SOA) and large numbers of technological approaches, which will help in planning a viable service.

Thesis is made as part of company X PROJECT Y, which aims to introduce new financial management innovations. As a part of this project, it is intended to produce a new service that automates certain finance transactions.

This thesis intends to find core principles in web service architecture designing. The principles are identified and opened up for closer examination on what they bring in to the project. The new planned taxation service is used as a case study to see how these ideas can be implemented and what the best practices to achieve web service architecture are.

The study concludes that there are number of helpful designing patterns, which help to realize modern day service. The most utilized design patterns are usually accepted across the board and don't trigger a lot of controversy.

Keywords
Web service, Service oriented architecture, Interface, software design

Table of contents

1 Preface

This thesis gives an overview of the web service design. The thesis has been made as part of a business project, which intends to manufacture new automated financial service. Through architectural planning, the thesis researches for the best practices on web service implementation and discovers how utilize them in the best possible manner.

1.1 Background

The thesis is mandated by Company X. It is made as part PROJECT Y, which is a part of bigger undertaking called project Z. Project Z pursues to advance the automation business transactions nationwide. PROJECT Y establishes a network of financial and technical specialist and policymakers. The project goal is to innovate services for automated systems and consolidate financial management information streams. There are multiple partners across industries and academic establishments. (Aalto) (PROJECT Y)

Production of one of the innovations was started in collaboration with Haaga-Helia. A proof of concept program was developed as a coursework by Haaga-Helia students and stakeholder advisors. The program automates process of declaring Finnish periodic tax return form. As a continuation of that project this thesis was assigned.

1.2 Choosing the subject

I had relative free hands in choosing the subject within the project. There was a lot of action and ideas around the PROJECT Y, but very little had been decided or specified. It was challenging to choose the subject. I tried to find something that would help launch the production of the service and add value towards the project in some meaningful manner. Because of my degree programme I thought that it would be good if the thesis was technical in nature, although there was a lot of business side questions still left unanswered. Technical meeting with project leader revealed that Company X developer team has their own very specific tools, which they work with. The developers use their company's frameworks, which are standard in their production. That is why I decided to make the thesis as technological neutral as possible. I ended up choosing to work on Architectural design, because it offers a good overview of the project as well as some solid substance. (Company X technical meeting 15.2.2015)

1.3 Research subject and goals

Architectural design is especially relevant PROJECT Y, because the service is largely undefined and still very much in planning. As of right now the reduced idea is to have service that automates filling out tax return form and possibly incorporates additional ac-

counting features. The main question of this thesis is: how to design architecture for accounting web service? Some additional questions are: What are the benefits of architecture design? Which are the current core web service architecture design ideas and what are they about? What it takes to realize this service from a technical standpoint -should it even go to production?

What this thesis excludes are: specific technologies and libraries used to build a web service. What are seen as outdated design models and frameworks or those, which are intended for completely different kind of services of that which PROJECT Y project intends to produce.

1.4 Research method

This a is functional thesis, which means that it tries to produce academic result and yield some value to real life project through those results. The study uses PROJECT Y as a case study to figure out the best solutions. The research method used is qualitative and with the research knowledge chapter followed by empirical data. My approach is to research large array of technologies, frameworks, designing models and pick the most used and relevant ones for closer inspection. Many of these didn't make it to the final thesis, because of time limitations, but I tried finding ones that matter and possibly might add some kind of value towards the project or raise unanswered questions. I had a couple of extensive books to guide my way, but I sought to find the latest relevant academic papers from web sources to back up my research, since web model trends are developing at such rapid phase. The idea was to read into the subject before writing anything and try to construct cohesive overview about the subject in question. There are also presentation events for key stakeholders and potential future users which I will partake. The idea is to present current proof of concept program for them and to collect general feedback, which might reveal about the direction this project should take to bring out viable service.

2 Web service Architecture

Standard web technologies such as HTML are designed to transmit static information. This does not simply suit the requirements of modern businesses. Web-services are the key to providing environment which is able to automate different processes in a larger scale and enable interactive exchange of information between the different stakeholders. (Alonso, Casati, Kuno & Vijay Machiraju 2003, 99)

A crude description of the web-service model could be that it is a program that is able to access different programs over the internet. For much more exact description one could quote World Wide Web Consortium (W3C), in which they state:

"A software application identified by URI, whose interfaces and bindings are capable of being defined, described, and discovered as Extensible Markup Language (XML) artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols". (Austin, Barbir, Ferris, Garg 2002, C.1.1)

In the book web Services ATM is used as an example of a simple web service. In the example ATM is the interface to customer, where the customer is able to access the information which is needed without danger of exposing information which might be exploited by the user. ATM as an example defines well the structure and the benefits of a standard web service although the cases are usually harder to conceptualize. (Alonso ec 2003, 97)

2.1 Service oriented architecture (SOA)

The benefit of approaching the project from SOA point of the view is to get the big picture of the services together, as well as to include both business and technical sides and see how they should come together. With SOA you are able examine the end product as a service with all of its internal and external components. Web services are the key in understanding how SOA based service is realized. The goal of the new piece of software is to add something new, not invent the wheel all over again. We can utilize networks for linking parts of the existing software to something completely new using web components. In this way businesses are in control of their respective software and its data even though that software may be just a piece of longer computer controlled production chain. When the need for the changes appear, for example a key supplier changes to another firm, it is possible to simply make the changes to the software in question and leave the rest of the supply chain's services untouched. (Papazoglou 2012, 14)

What is the value that the SOA adds towards the project then? SOA is a complete designing pattern, which answers to many questions which might not even been

asked. It offers framework to work with and baseline for architectural planning. The goals which can be pursued through SOA are best described by SOA manifesto which states:

> "Business value over technical strategy
> Strategic goals over project-specific benefits
> Intrinsic interoperability over custom integration
> Shared services over specific-purpose implementations
> Flexibility over optimization
> Evolutionary refinement over pursuit of initial perfection"
> (SOA Manifesto Working Group)

If these values are shared by the project then the SOA might be the correct tool for it.

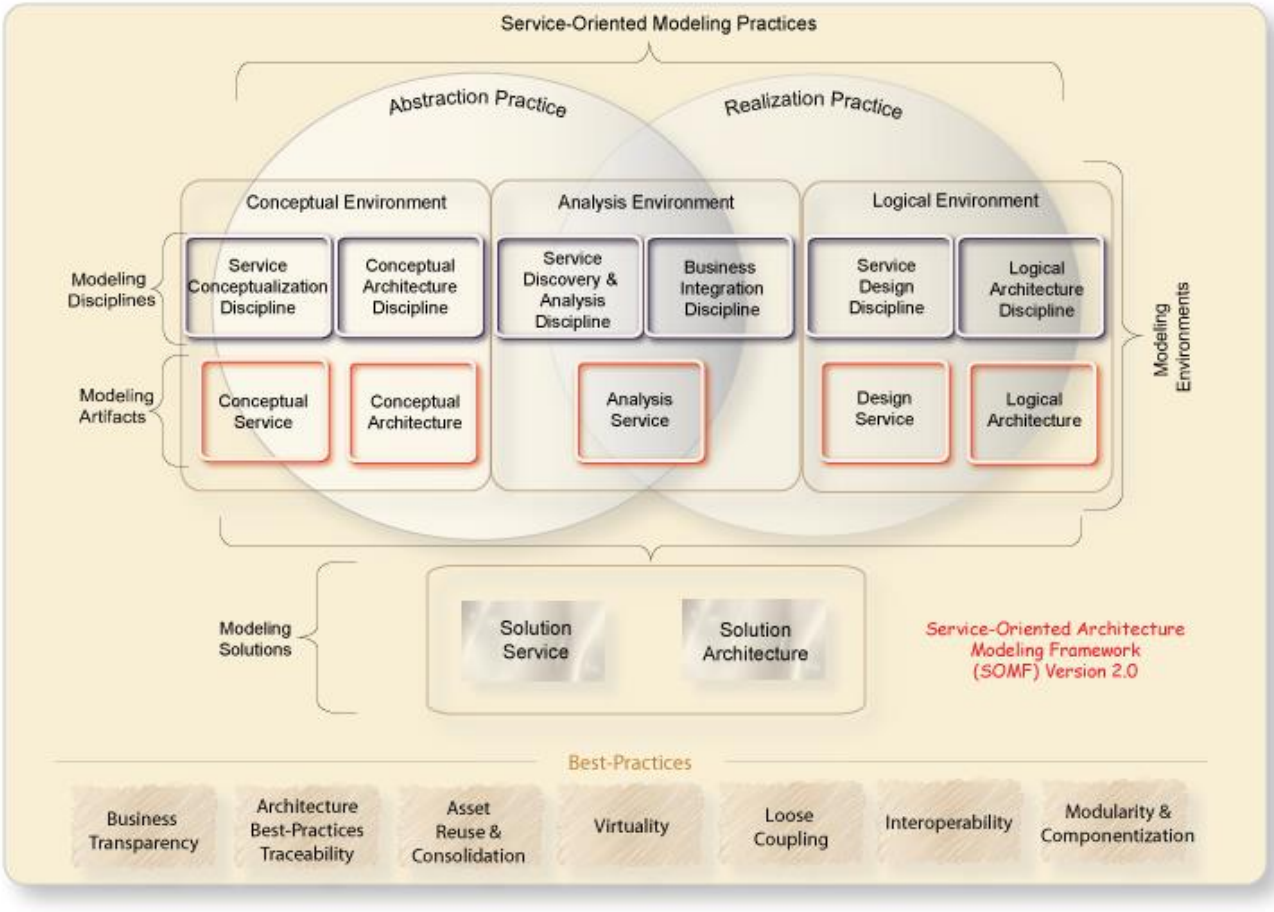

Figure 1 presentation of framework elements included in SOA thinking. (Angela Martin)

2.2 Cloud computing

The purpose of cloud computing is to lower demands towards the customer in terms of required investment and to keep more control over the service in producers hands. By removing complexity of running a service in user's perspective you can free up need for physical devices, the time it takes to set up and manage systems and even lower the technical know-how. Cloud computing also caters better towards pay by use model, like

subscriptions offering expensive tools to wider consumer and business base. (Papazoglou 2012, 2012, 750-754; Manes 2009)

There are three cloud computing models; Infrastructure as a service (IaaS), in which the whole computing power, memory, network, operating system is provided by the service producers. This is basically the whole computer and all the software it contains, leaving the end user only in need of internet connection terminal and knowledge of how to use the given system. The next level of cloud computing is Platform provider as a Service (PaaS). This means that the system is provided to user, which might be for example used as a resource by other systems or a user who doesn't want to deal with complexity of setting up the system in question, like coding environment. User might be granted access to configure the environment, but only as much as environment is capable. Final tier is to only provide Software as a service (SaaS), in which user gains access to software over the internet. This can be achieved by client software or through an internet page over a browser that grants access to the user interface. (Papazoglou 757, 2013)

2.3 How does example technologies run a web service

There are a great number of technologies included which form the web service platform. Some of which are so common that they have become a standard. The client side is usually accessed by the browser, so there you have at least HTML and possibly some functional front end languages such as JavaScript. Information is handled and processed by the backend which is run at a service provider's server. In order for a server to service multiple clients it needs a servlet to create instances for each customer. Once the instances have their respective sessions, server's program is able retrieve the information from database and move it to the client through java applets embedded in html. (Alonso es, 99-100, 102, 2003)

Model-view-controller design (MCV) is usually used to represent the user, software and database interactions. Although it's not originally web service architecture model, it is currently widely used in planning them. The model describes hypothetical program's functions as follows. Controller receives the inputs and steers the programs actions. Model manages and modifies the data and finally view is designed to show the response to the user. View data, can be wrapped in different contexts, like multiple forms of charts. This is the intended logic, which the program follows within the user interface to the database. (Reenskaug, Coplien 2009) (C2.com 2013)

When you are designing more complex web service there is also the data traffic that travels between the different services. Easy data transportation between the systems over internet is layered method of constructing web components. There are multiple ways to implement data transportation between applications, which are different, but

not necessary competing methods and technologies. It is for example possible to build method to transform XML-document across to another system purely using SOAP or REST, but it is also possible to combine these two to take advantage of their best aspects. It is also possible only to use Hypertext Transfer Protocol (HTTP) for data traffic, but this sets up certain constraints as for using exact data format, which both software are able to understand. It is dependent on the functional and quality requirements on which guidelines and technologies should include in to a project. (Champion, Ferris, Newcomer, Orchard 2002) (Alonso es p.93-96, 118, 2003)

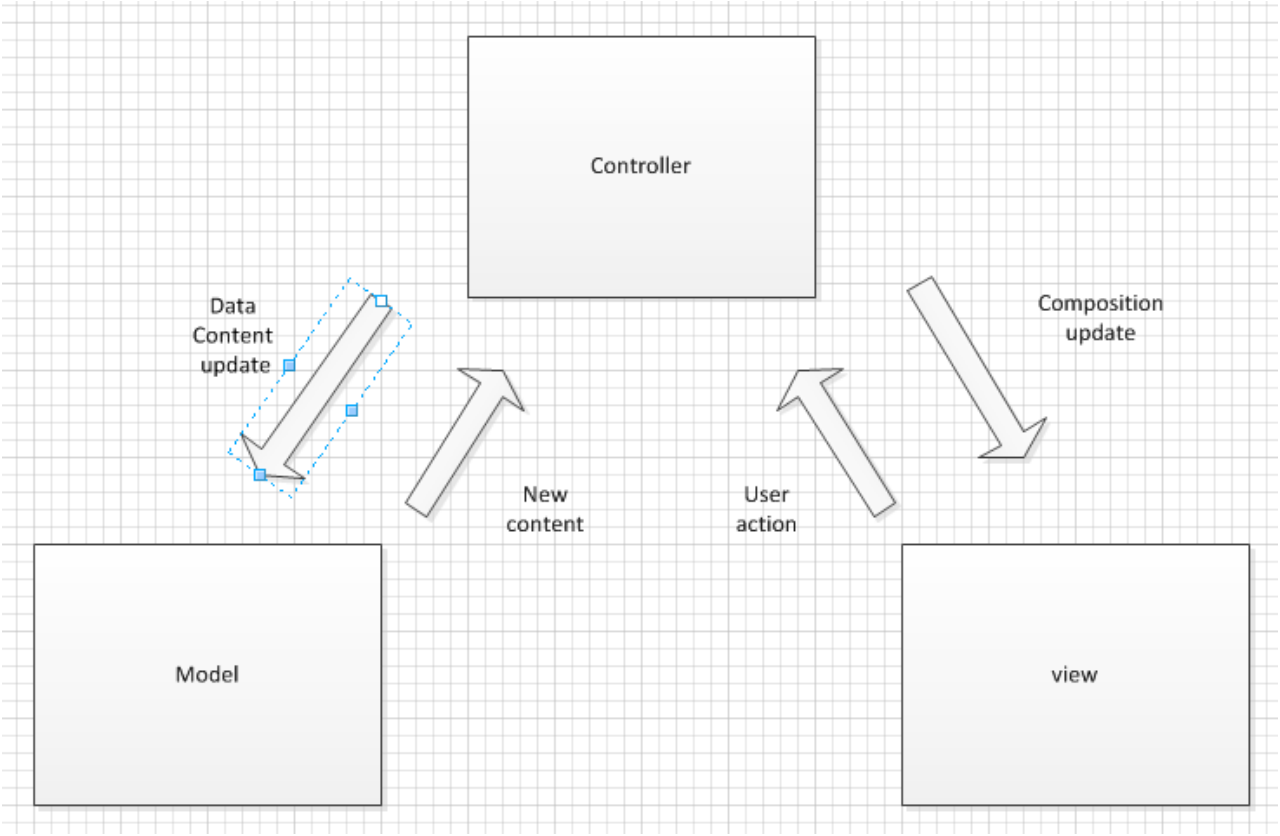

Figure 2 representation of MCV. (Code project 2008)

2.4 Web Services Description Language (WSDL)

WSDL describes information about the application interfaces and services which it contains. The idea is to provide machine readable description to developers who are interested in the software in question. Developers should acquire essential information about the possible services interfaces and how to connect to those interfaces. It is even possible to generate code required to access service in question from WSDL.

WSDL 2.0 is written in XML and it consists of five different parts, which are:

Service exposes services that are accessible by web protocols.

Endpoint provides the services URL used to connect the service.

Interface states the detailed operations and parameters used by these operations.

Operation describes the way in which SOAP message is encoded.
Types convey the type of data contained within the service.
(Alonso es 2003, 165-174; Chinnici, Haas, Lewis, Moreau, Orchard, Weerawarana 2007)
WSDL 1.1 differs little from 2.0. The main differences between the two are that 1.1 contains **messages** as a sixth part, which expresses information that is contained in other five elements in 2.0. 2.0-version is approved by W3C and is officially endorsed, while 1.1 is more widely used and supported by wider array of tools. Representational state transfer (REST) is supported in 2.0 so it doesn't relay completely on SOAP, however it is currently recommended that you create both versions of the file if you elect to use 2.0-version. (Manes 2008)

2.5 Simple object based protocol (SOAP)

SOAP likes to view itself as the name tells us a protocol. The idea behind it is to create an envelope, in which you include the message in the XML format. SOAP is easy to extend, meaning that it has libraries which are included to the project depending on which functions are needed. There are many tools to support SOAP, for example to generate interfaces. SOAP is independent, so it doesn't set any requirements to other technologies used and it can be used with any programming language. Performance wise SOAP might set some roadblocks if there is huge amount of network traffic. It is usually heavier than its alternatives. (Kohlhoff & Steele; Moore 2001; Papazoglou 2013, 125-145)

The SOAP message holds three parts: Envelope itself, which is mandatory and represents the message itself. It guides message to right URI. Header is optional, but it extends the use for decentralized actions and can be used for example authentication. In the final part the body converts the data into the XML form and provides the error handling. It also triggers the call to the application programming interface (API). (Don Box ec)

SOA's principles also dictate that the program should be easy to access by external resources and SOAP can be send over any underlying protocol, such as HTTP. Although it takes some guidelines on how to connect one system to other using SOAP if you don't have any previous experience with it. That's why SOAP might not be the easiest to pick up, if you don't have any knowledge to start with, but it's certainly easy to operate with, once you are familiar with basic concepts and the syntax used in SOAP libraries. (Company X 2008; Moore 2001; Papazoglou 2013, 125-145)

2.6 Representational state transfer (REST)

REST is compiled best practices for creating architectural model. It uses basic web-service commands like PUT, GET and POST, but also offers framework in how to use

these elements in system to system communication. Just like for SOAP, REST has multiple libraries for different languages. REST is not as much of a technology itself as it is a guideline for using existing components.

REST's basic idea is that it places constraints in how these components are to be used. These constraints are: Application programming interface (API) should not limit what protocol is used for data transform, nor should it make any changes to protocol unless protocol is incomplete to perform its task. API should use its effort in defining resources' media types. The resource names should be left to be defined by the user. REST API should be possible to use by someone who does not have beforehand understanding of its underlying function. The only information that client is assumed to have is uniform resource identifier (URI) and understanding of the service's media types. (Roy T. Fielding)

REST is highly modifiable, thus you are able to build extremely specific systems. High modifiability also means that it scales and offers possibility for a great performance if configured properly. It is as independent as is currently possible, only requiring client to know its entry point information. (Pautasso, Zimmermann, Leymann 2008; Pedro Verdeck 2013)

A simple REST program was made to simulate fetching data. It was made using Jersey library, because after testing multiple ones, it proved to be easiest to handle. The goal was to see how difficult it would be to use REST library. Program, is included in appendix 2.

3 Architecture design implementation

PROJECT Y project aims to create more information about their innovation to find out whether the intended service would be viable to execute. This chapter will produce technical information, more specifically architectural solution. First the current work and plans will be established. After that the research base (second chapter) findings are used to design potential architecture for the service. Specifications for the service are still somewhat under a progress, so in some cases options are provided instead heading to a single core solution.

3.1 Service description

The service seeks to automate creation of periodic tax return form and send it to taxation administration. The service gathers its data from the electronic bills which are retrieved from external system(s). The user the ability to remove, modify and add bills within the program. The original bills retrieved from external system will not be modified, there for the copied bills are the ones being processed by the program. The user is responsible of all the changes that are made to the data and the service doesn't take responsibility for factual data content.

Some additional accounting features are also planned to be included in the service. The electronic bills have a possibility to offer vast amounts of data for wide array of functions, but these additional accounting features are yet to be defined. Functions that are defined are as functional requirements chart in appendix 4. (COMPANY X 2014, ARA use case diagrams; project meeting 18.2.2015)

Since the automatic value-added tax (VAT) return form is already a well-defined service concept shall be referred as automatic VAT reporting (AVR). See appendix 3 for a chart illustrating AVR architecture. (COMPANY X 2014, ARA use case diagrams; project meeting 18.2.2015)

3.2 Proof of concept as a guide

Designing architecture from the scratch is extremely difficult and even unproductive. It is essential to have the difficult questions made earlier in stages rather than later. For this proof of concept (POC) is excellent tool. It is usually beneficial step to create first low budget version in order the see all the technical requirements of the service. (Odysseas Pentakalos 2008)

For PROJECT Y some groundwork has been made. One of the produce is a demo version of the AVR program. Demo program was made by a small team of students

as a part of the school course; the goal was to prove that such program is at all possible. Company X also had representatives to work as product owners and specialist educating about the subject for the students. The focus was more directed to the business logic than technical requirements and shortcuts were made in order to get the product out in time. However these technical shortcuts were noted and documented to help the designing of the end product. One of the shortcuts was excluding EU-trade and VAT-relief from business logic. To complete business logic a chart was compiled for EU-trade and VAT-relief in appendix 3. (Verohallinto 2015)

3.3 Service requirements

The service must be met certain goals, that has been set during the project. First of all it is designed as a white label service. What this means that the service seeks to gain multiple big clients and for each of the clients it is meant to be integrated to their existing system. In order to achieve that for every client there will be a customizable user interface (UI), bearing the client's brand and user experience. The end user of the service is assumed to be small business owner. So it is unlikely, that the software should go through huge amount of data. As for the services data, currently it might be incomplete to run the software's business logic. Architecturally you can only tackle this problem by recognizing the incomplete data and exclude it from the actual functions. (Project meeting 18.2.2015)

There also exists a market outside domestic sphere, which might be tapped in the future. This would require multiple sister programs, that have different business logic to cater the differences in taxation laws. (Business meeting 27.2.2015)

Additionally functions to business logic will be a potential user notification system. A user can be informed through a text message of a pending tax return form or about the fact that certain monetary thresholds crossed. This might require somewhat larger processing power in order to conduct the passive calculation checks. It is possible to hardcode these checks to certain dates and during the timeframes, when the actual usage is otherwise low. (Project meeting 18.2.2015)

3.4 Feature design

The proof concept software has been presented to multiple key people, like small business owners and those who contribute to national financial and political projects. Feedback from these people has been positive, but there have always been wishes for more. The most common feedback has been that there would be more extensive service, than what the POC offers. It takes great deal of effort to introduce new practices for the people and it helps if they feel that the benefit of adopting new system is significant. The planned additional accounting features, which are combined with current AVR system, might help

reaching that. However it would require greater investment to launch all these services at the same time. Web services enable easy way to build larger services piece by piece and the accounting services could be developed as a somewhat separate service. This would enable to place its status to secondary place and first go ahead with smaller steps and launch the AVR service first. (advisory board 20.1.2015; Suomen yrittäjien littto 2.3.2015; Kunnantaito OY 24.3.2015)

AVR and the additional accounting services use mainly the same data. That is why a joint database option should be considered as well. Even if AVR and the additional accounting services would be mostly their own entities, it would be possible that work under a single database solution. This would lead to less interfaces and possibly reduce the work on the databases. Based on this model, class diagram and relation diagram illustrate possible database design, which can be found on appendix 5 and 6. There are two extra tables created in relation diagram to account for multiple users for one company and people who might have multiple companies.

The programs interfaces are on an important position when there are high customizable requirements. ARV programming interfaces should be concentrated between the business logic and backend controller. This way it is entirely possible write a new logic in case service will be provided on abroad as well. Every country has its own tax laws and there for requires its own business logic. Whether the user interface should be independent from back end code depends on the length that the white labeling is necessary to take. If the clients are prioritizing preserving the user experience, like navigation throughout their whole service the user interfaces should also be on a high priority. This would make it possible to retain the navigation to match every clients existing service. The easier solution is to make a single neutral user interface and rely on modifying style sheets by utilizing style sheet language tools such as Less.

All the services designed functions and the data moving between these functions are composed to data flow chart in appendix 8. Squares indicate interface, bubbles functions and open box data storage.

3.5 Inter service APIs

The benefit of the web services can utilize by fact that the server's program is able extent its capabilities by accessing other programs interfaces. The transported information is packaged in XML and send over the network using for example SOAP to convoy the message to the target program. Well-structured interfaces allow recursive approach to a service, which is the main benefit that service-oriented architecture is trying attain. It might be worth to implement highly customizable architectural style such as REST. This allows eas-

ily extending existing features, such as combined accounting services. (Paul Prescod 2002)

Ilmoitin is the finish tax administration's web interface that has ability to check the tax return form format, present your earlier tax fillings and receive filled tax return form. In order to gain access to interfaces KATSO identification must be acquired, except for the checking interface, which doesn't require any identification. Ilmoitin has been has been made using SOAP and it has extensive guidelines on how to connect a web service to Ilmoitin using SOAP. Use of SOAP should be somewhat limited if there are intentions to scale up the amount of data to huge quantities in the future, because of its network performance issues. (Kohlhoff & Steele) (Ilmoitin)

Send function was created for POC to test Ilmoitin's check interface and experiment on how difficult it is to create a SOAP message. Ilmoitin web-site has detailed specs about the interfaces and guide on how to access them using SOAP. Detailed code for this function can be found on appendix 7. (Ilmoitin)

3.6 Infrastructure

There are multiple ways to offer a service to clients. It is possible to simple handout lease the software for clients use. However it might be more beneficial to follow the SaaS model. In SaaS model the server is provided by the same company that owns the software. It leaves Company X more in control of the service and enables better access to metadata. This metadata could reveal some useful statistics about the use of the program such as usage. This would help in future service improvement as well as with marketing to potential clients. The metadata also enables optional billing methods such as pay per use. Client would be charged every time its customer would use the server. SaaS would also offer the client more complete package, not having to focus on server side choices. (Papazoglou 2013, p.757)

Alternative methods could also be implemented with other clients if they wish to have the additional control, for example to centralize their services under a single domain. (Business meeting 27.2.2015)

4 Results

The concepts researched indicate that all of them have their respective uses in modern service design. They offer a great way to produce a conventional service. One of the research goals was to find the best practices in service design, but methods in this thesis aren't in really competition of each other and can coexist and even complement each oth-

er. Of course there are clear alternatives to concepts, which have been studied in this thesis. However there should be more in depth and specific study to be conducted in order to find the best methods across all the possible alternatives.

Benefits for implementing the concepts in this thesis are: a clear blueprint for development. This enables a well-structured project, which is easy to document and produce with a bigger more complex team. It is possible to create a scalable service, which can be realized in phases. That means that it is to restart the development and add additional functions after the product has been launched. Utilizing the concepts also leads functional externals APIs and description of the service to stakeholders, who might be interest to gain access service's features. Service is also depends less on specific stakeholders and is easy to configure to use different systems.

PROJECT Y has complex service in hands not only organizationally, but also technically. On the bright side project is very suitable to be implement in stages, as long as all the architectural considerations are kept in mind while producing software. There is opportunity to tailor the service for a single customer. Then measure if the project is still potential enough. Possible additional researches for Company X to conduct next would be extensive market research, which could clear questions on the business side of the service and ultimately tell whether it should be launch. As a compiled result a simple diagram reflects overview architecture of the intended service in appendix 9.

5 Conclusion

By this researched I aimed to shed light of different architectural solutions for a software based service. What I discovered is that the studied field is more like eco system, rather than different competing ideas. There are very few cases when a single methodology or technology triumphs objectively over another one. SOA for instance is almost like a mother of concepts in architectural design, which benefits from connecting multiple systems. Its principals may be utilized in old school CORBA (Common Object Request Broker Architecture) based development as well as with modern day cloud services.

You can usually manage without deep understanding of architectural design, but it most likely helps to understand them as then you will be able plan the bigger picture as well. Fitting analogy could be a programmer, who is well rehearsed in higher level coding language. One can make functional programs, but if there is lack of understanding of the underlying processes which take place closer to hardware level. It might leave programmer in situations, where is unable to fix a bug, or even conceptualize what it is that he should be exactly doing. Like this understanding of the underlying processes also understanding architecture design might be useful, it help the programmer to have consistent picture of the software and enables long term right approach, rather than one that simply seems the best, given the current situation that the project faces.

Understanding web services and the design patterns relating to it is absolutely necessity for someone who is planning getting into software development. It might take some patience to internalize all the information about the tools and design patterns and how they are connected. But approaching the subject from a top down perspective starting from SOA offers great learning experience, which will leave you with insight and ideas.

Information resources for architectural design and interface implementations might be somewhat scarce and very rarely written in an easy and understandable manner. Technologies age, but the ideas and concepts usually don't. The books used in this thesis offered a consistent and cohesive view of concepts. They helped to understand how different parts are associated together and what the next subject you should read about is. As for the web sources used for this thesis they are usually written by someone who is respected professional working on these tools. This does not remove the chance that if an article for example has been written by a single person and that person writing about extensive subject, that he might have some misconceptions. Most of the sources are however written by multiple people who are working with highly regarded establishment as W3C. In some cases it is unclear who has written the source and in these cases reliably

has been sought from establishment which has published it. There is not huge amount of controversy about the subjects, except for some individual cases like in how easy it is to adopt use of REST. It was often contrasted to SOAP and the opinions varied. My own experience working with both during functional side of this thesis is that REST as a concept is seems easy, but the frameworks' libraries demand a understanding of the project environment setup. I did not experience same difficulties with rigid, but straight forward SOAP frameworks.

I would suggest further research be done about the REST and SOAP frameworks. There are multiple competing libraries and it would shed new light to have study of their properties. As for now choosing a library for your project that utilizes REST feels like lottery game, where you hope to get a easy to configure working tool.

The process of making this thesis was somewhat challenging. However it has also been educational. I have always been interested how web services actually work on a bigger scale and creation of this thesis has absolutely answered that. The most considerable benefit for me and hopefully for the reader was to gain new perspective how services CAN be implemented and through that introduce new ideas about potential services.

Sources

Aalto yliopisto http://information.aalto.fi/en/research/rte/ 29.4.2014

PROJECT Y http://www.taloushallinnonrunkoverkko.fi/hankkeesta/ 29.4.2014

Daniel Austin, Abbie Barbir, Christopher Ferris, Sharad Garg 2002 http://www.w3.org/TR/2002/WD-wsa-reqs-20021011 visited 6.4.2015

SOA Manifesto Working Group http://www.soa-manifesto.org visited 8.4.2015

Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer 2000 http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383497 Visited 2.4.2015

Figure 1 Created by Angela Martin based on Michael Bell's book Service-Oriented Modeling: Analysis, Design, and Architecture, Wiley. Figure has been contributed to public domain.

Brian Moore 2001 http://www.techrepublic.com/article/an-introduction-to-the-simple-object-access-protocol-soap/ visited 1.3.2015

Michael Champion, Chris Ferris, Eric Newcomer, David Orchard 2002 http://www.w3.org/TR/2002/WD-ws-arch-20021114/#basicext Visited 1.4.2015

Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, Sanjiva Weerawarana 2007 http://www.w3.org/TR/wsdl20-adjuncts/#soap-operation-decl-description Visited 15.4

Code project 2008 http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design Visited 29.4.2015

Verohallinto 2015 periodic tax return form's precise information http://www.vero.fi/fi-FI/Syventavat_veroohjeet/Lomakkeet/Yritys_ja_yhteisoasiakkaiden_lomakkeet/Kausiveroilmoitus/Kausiveroilmoituksen_yksityiskohtainen_t(19441)

Anne Thomas Manes 2008 http://searchsoa.techtarget.com/answer/WSDL-11-vs-WSDL-20 Visited 15.3.2015

Odysseas Pentakalos 2008 https://msdn.microsoft.com/en-us/library/cc168618.aspx Visited 22.4.2015

Christopher Kohlhoff, Robert Steele http://www2003.org/cdrom/papers/alternate/P872/p872-kohlhoff.html Visited 30.4.2015

Company X 2008 Technology materials IntroductionToWebservices.pdf

Roy T. Fielding 2008 http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven Visited 30.4.2015

Pautasso, C.; Zimmermann, O.; Leymann, F. 17th International World Wide Web Conference (WWW2008) www2008-restws-pautasso-zimmermann-leymann.pdf visited 3.3.2015

Pedro Verneck 2013 http://stackoverflow.com/questions/19884295/soap-vs-rest-differences Visited 1.3

Company X ARA use case charts 2014.

Anne Thomas Manes 2009 http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html Visited 21.4.2015

C2 2014 http://c2.com/cgi/wiki?ModelViewControllerHistory Visited 26.3.2015

Ilmoitin https://www.ilmoitin.fi/kehittajat/Etusivu 1.5.2015

Paul Prescod 2002 http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html Visited 9.4.2015

Trygve Reenskaug, Jim Coplien 2009 http://www.artima.com/articles/dci_vision.html Visited 26.3.2015

Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. Y.2003. Web Services: Concepts, Architectures and applications. Berlin. Springer

Michael P. Papazoglou. Second edition. Y.2012. Essex. Pearson.

Company X (Technical meeting) 15.2.2015

Company X Business meeting 27.2.2015

Company X Project meeting 18.2.2015

PROJECT Y presentation to Suomen yrittäjien liitto 20.1.2015

PROJECT Y presentation to Project Z advisory board 2.3.2015

PROJECT Y presentation to Kunnantaito OY and Company X partners 24.3.2015

Appendices

Concealed