Metropolia University Of Applied Sciences
Institute of Technology
Degree Programme in Media Engineering

Joonas Pajunen

Reserved Call Queue With Service Oriented Architecture

Bachelor's Thesis. 23 November 2009

Instructor: Jörgen Eriksson, R&D Manager
Supervisor: Kari Aaltonen, Principal Lecturer
Language advisor: Anne Perälampi, Lecturer

Helsinki Metropolia University of Applied Sciences ABSTRACT

Author Title Number of Pages Date	Joonas Pajunen Reversed call queue with service oriented architecture 62 23 November 2009
Degree Programme	Media Engineering
Degree	Bachelor of Engineering
Instructor Supervisor	Jörgen Eriksson, R&D Manager Kari Aaltonen, Principal Lecturer

The goal of this thesis project was to provide certain functionalities to allow interaction with a telephone box system. This was done by creating a graphical user interface that is usable on a browser. The client of this project was the mobile laboratory of the R&D department in Metropolia. The essential idea for the project was that it is a "reversed call queue" in a help desk type of environment. Customers can call the help desk, and instead of waiting on line, the phone number is stored to the system and by using it, the employees can call back when they are able. The system gathers, generates and displays more data such as general statistics and case specific information.

The system was established with Asterisk open source private branch exchange software running on a Linux server. The user interface implements service orientated architecture and model view controller design. The service, and the client consuming it, were both done in PHP with its SOAP libraries. The service communicates with Asterisk directly and via reading and writing data in a MySQL database. The system supports both analog phones and VoIP software phones.

The plan was to develop a version able to demonstrate the system's capablities, not to make a final product. These initial requirements were met, even with additional features in the application. The thesis also covers how service orientation affects a software development process and infrastructure, especially the model view controller.

Keywords SOA, MVC, SOAP, Asterisk

Metropolia Ammattikorkeakoulu Insinöörityön tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Joonas Pajunen Käänteinen puhelinjono toteutettuna palvelusuuntautuneella arkkitehtuurilla 62 sivua 23.11.2009
Koulutusohjelma	Media Engineering
Tutkinto	Bachelor of Engineering
Ohjaaja Ohjaava opettaja	T&K-johtaja Jörgen Eriksson yliopettaja Kari Aaltonen

Insinöörityön päämääränä oli hallinnollisen toiminnallisuuden toteuttaminen puhelinjärjestelmään. Tämä saatiin aikaan luomalla selaimessa toimiva graafinen käyttöliittymä. Projektin tilaajana toimi Metropolia Ammattikorkeakoulun T&K-osaston mobiililaboratorio. Ydinajatus oli toteuttaa niin sanottu ”käänteinen puhelinjono” esimerkiksi help desk -tapaiseen ympäristöön. Asiakkaat voivat soittaa palvelunumeroon, jossa odotuksen sijaan heidän puhelinnumeronsa kirjautuvat järjestelmään. Sitä käyttämällä asiantuntijat voivat halutessaan soittaa asiakkaalle. Järjestelmä kerää, tuottaa ja näyttää myös muun muassa tilasto- ja tapauskohtaisia tietoja.

Puhelinjärjestelmä toteutettiin Linux-palvelimella toimivalla Asterisk-puhelinpalvelinohjelmistolla. Käyttöliittymä toteutettiin palvelusuuntautuneella arkkitehtuurilla ja ”model view controller” -mallinnuksella. Palvelu ja sitä käyttävä asiakasohjelma toteutettiin PHP:llä ja sen SOAP-kirjastoilla. Se kommunikoi suoraan Asteriskin kanssa ja lukemalla ja kirjoittamalla tietoa MySQL-tietokantaan. Järjestelmä tukee sekä perinteisiä analogisia että VoIP-puhelimia.

Tarkoitus oli luoda ominaisuuksia ja mahdollisuuksia demonstroiva versio järjestelmästä eikä välttämättä hiottua ja viimeisteltyä tuotetta. Nämä vaatimukset saavutettiin, ja myös ylimääräisiä toiminnallisuuksia asiakasohjelmistoon. Insinöörityö käsittelee myös sitä, kuinka palveluorientaatio vaikuttaa ohjelmistonkehitykseen, erityisesti, jos käytössä on samanaikasesti ”model view controller” -arkkitehtuuri. Palveluorientaation toteutus vaikutti ohjelmistonkehitykseen huomattavasti, muun muassa ohjelmoinnin ja komponenttien lopullisen määrän osalta. Model view controller -arkkitehtuurin käyttö projektissa toimi hyvin ja helpotti osaltaan palvelusuuntautuneisuuden jäsentelyä ja toteutusta.

Hakusanat	palvelusuuntautunut arkkitehtuuri, MVC, SOAP, Asterisk, puhelinjärjestelmä

Table of Contents

Terms and Acronyms

SOA, Service Oriented Architecture. A design and implementation method, where a software is created in a modularized way by generalizing the protocols and functionality.

BPM, Business Process Management. A management process for a company to incorporate its business needs to other areas in it, especially into the technological ones.

SOAP, Simple Object Access Protocol. A standardized messaging protocol for developing web services.

REST, Representational State Transfer. A newer and simpler messaging protocol for web services development.

WSDL, Web Services Definition Language. A language used in web service development to describe them and allow message generation automation for the protocols.

UDDI, Universal Description, Discovery and Integration. A registry for storing information about the different available web services.

MVC, Model View Controller. A design paradigm, where a software is divided into three separate components, each independent of others.

XML, eXtensible Markup Language. A text document used for hierarchially storing data.

AJAX, Asynchronous Javascript And XML. A technique for a website to allow more dynamic communication with a server and presentation in the browser.

VoIP, Voice over Internet Protocol. A term for transferring voice through Internet or other network using Internet Protocol.

1 Introduction

The goal of this thesis project was to provide certain functionalities for controlling and interacting with a telephone box system. The required functionalities were reached by creating a user interface by using server and client side scripting, so the resulting product is used with a web browser. The telephone box system was set up with an open-source software called Asterisk, and it runs on a Linux system.

The client in this project was the Mobile Lab in Metropolia University. The idea for the project was that an organization's client can call to a help desk, and if there are no lines available, the client can then leave his call records to them. Then, a queue is formed in their reverse queue system's database, and it can be viewed using a web browser. A secretary, or a professional can then choose which ever client they want to call back to. An administrator or a supervisor can view statistics and a real-time situation of the help desk. The web pages providing the queue information can be viewed with any computer or cell phone with a browser and Internet connection capabilities.

The project was implemented by two people. My part was the development of the human interaction functionalities, whereas the Asterisk telephone box communication system was developed by another student, Henri Mäkinen. The core functionalities were to provide real-time information flow between the user interface and Asterisk, which is achieved via database queries in regular intervals. This communication also implements service oriented architecture to allow generalized and more diverse communication.

The purpose of the project was to create a functioning demo, which displays the capabilities of this kind of system, not to make the final, polished product. The schedule for the project was roughly the first half of year 2009.

2 Principles of Used Technologies

The most essential technologies in this project are covered here in the order in which they appear in a reguar case flow. They are presented in a bottom-up order, starting with the essential system configuration with the Asterisk telephony & PBX (Private Branch Exchange) platform, then moving into SOA (Service Oriented Architecture) and MVC (Model View Controller) to cover the service and application logic.

2.1 Asterisk Telephony Platform

Asterisk is an open source, private branch exchange software. It can be set up on a hobbyist's laptop or on a system with multiple servers. This makes Asterisk scalable from small, few channel systems to a wide business sized branch exchange. It generally runs on top of a Linux operating system, adding no expenses to software investments. The system can be attached to real analog phone lines or software, VoIP (voice-over-IP) phones. In addition, Asterisk is completely programmable to suit any needs required by the system. [1, 2; 1, 12]

Complete customization and programmability in an application usually mean that they cannot be the simplest to implement or easiest to understand. Asterisk requires configuration, call flow design and management, and good understanding of low level software, hardware and Internet technologies. [1, 2] The software can be managed completely through plain configuration files or a GUI (Graphical User Interface) specifically developed by the same company that developed Asterisk, Digium Inc. The GUI is extensive and slightly complicated, but very useful [1, 246].

The success of VoIP software phones has been already proven by the millions of Skype users, even though the actual term is relatively unknown. Comparison between the traditional analog telephone and software phone is similar to that of film camera and digital camera or analog television and digital television. The difference being only the transferred content. Asterisk covers several protocols and codecs from GSM to MP3; it can convert the analog sound into multitudes of different formats and send them through the Internet using various different protocols [1, 185-186]. Figure 1 displays the versatility of an Asterisk PBX in terms of end devices and transfer mediums.

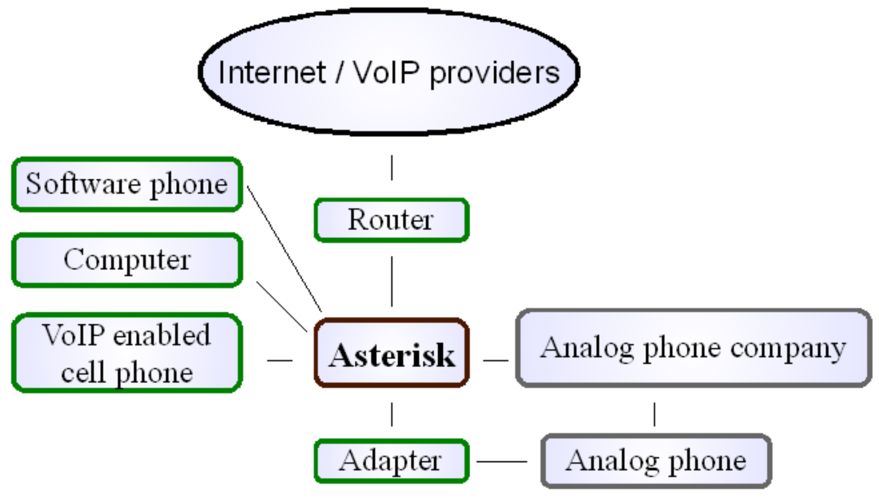


Figure 1: Drawing 1: Possibilities of Asterisk connections

As illustrated by figure 1, the Asterisk Server accepts analog and digital connections, of which the digital can be already converted. The possibilities for software phone connections are numerous and location independent, covering the Internet and therefore the whole world. The available end devices range from headset to public pay phones.

2.2 Service Oriented Architecture

Service Oriented Architecture, SOA, is based on the idea that instead of providing software with language specific functions, it provides a service with generally formatted messages. One service therefore is similar to a function: it completes one operation. SOA provides a set of services, each doing their own operation, for example handling the input from a form or delivering information from a database. The service allows these operations to be accessed without any knowledge of the underlying system and software architecture, simplifying the use of that service dramatically. [2, 38-39]

The services can communicate, but are independent from each other. It means the services are loosely coupled, requiring only the messaging protocol to transfer information. A simplified explanation of a service oriented system is that it consists of a client application interacting with a service application with standardized message protocol. Using the service is called “consuming” it. The most popular ways to achieve this chain of events is with Web services and SOAP(Simple Object Architecture) messages. The following figure presents the idea of separating a client from a service, as well as a third party providing information about the service. [3, 6, 43]

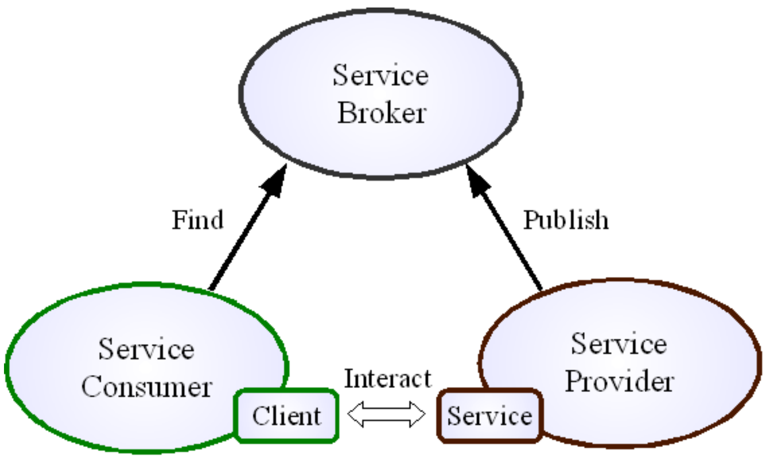


Figure 2: Service Oriented Architecture[4]

As seen in figure 2, the client and service are not physically connected. This interaction can indeed happen between any two entities between any distance. Although, it is not uncommon for an enterprise to provide services inside their own infrastructure to abstract their software logic.

These services are generally accessed through the Internet, but many applications using them run in the intranets and extranets. Technically, services can interact on the same local network, same computer even, or through Internet. The services appearing on the Internet are normally of public nature and general information, whereas companies use them internally to achieve business-to-business goals or better management of their software infrastructure.

The loose coupling of services highlights one of the key issues and benefits of SOA. Developers should be able to "mash up" different services quickly and easily to add data or other features to their applications. Modularity of applications will allow reuse and decrease the need to redesign or recode programs. The idea of building a system and using pieces created by others makes the process faster and cheaper. Depending on the project and developmers, it could be generally assumed that there is, in most cases, someone who has already done some part of what is needed. There should be no need to do it all again, as a well constructed service allows reusability, similar to object oriented programming. A generically constructed service can be reimplemented to fit to another process completely, or implement a different processing logic underneath [2, 164].

The first SOA implementations were introduced in the mid 90's, in the forms of CORBA (Common Object Request Broker Architecture) and DCOM (Distributed Common Object Model). These are language and communication platfrom specific services. They are efficient and well working ways to create and maintain services, but do not provide any support for interoperability with each other. They are, in many cases, much harder to expand, maintain and discover. The newer and recently more popular way to implement SOA is called Web services. It has basically the same functionalities as CORBA and DCOM, but it is a W3C (WWW Consortium) supported standard. [5]

The use of services created by others raises an issue of security. The service should be trusted and cover all the general security requirements. SOA does not determine the security policies in any way, but it is up to the service creator to implement authentication, encryption and others. This technical viewpoint of the security will be implemented at the message protocol level, such as SOAP. [3, 21]

2.3 Web Services

There is no official definition for Web services, but it can be described as a system providing services in a SOAP, REST (Representational State Transfer) or RPC (Remote Procedure Call) fashion. These messaging protocols and one of the general transfer protocols as delivery platform form the basics of a web service. The general and most widely used approach is a service with SOAP messages using HTTP (Hyper Text Transfer Protocol) [3, 52]. A web service in general is comprised of three elements. The first of them is the service contract defining the web service data transfer protocols. The second element is the actual programming logic as in any other software. This can be developed specifically for the web service, or reuse code from a legacy system by wrapping it to the web services own logic. The final element is processing the messages to a proper format usable by the previous elements. [2, 48]

Web service is an application that provides an API (Application Programming Interface) to which another application can connect using SOAP or another suitable messaging protocol. The application consuming the web service needs to know only what kind of request to give to it, and as a response, it will receive the data associated with that request. These messages are usually in XML (eXtensible Markup Language) format, better explained in hte next chapter. For a program to understand the these messages, a web services is defined by a WSDL (Web Services Definition Language) file. This definition is especially important in machine-to-machine communication, and emphasizes the fact that web services should be "self descriptive" and "discoverable". [3, 7]

WSDL is basically a text file with XML syntax. It defines how requests and responses are handled, what datatypes are used by the messages, and if there are any special datatypes that need to be defined. WSDL instructs an application on how to use SOAP or other transport means to communicate with the web service. Modern development tools and programming languages can use these files to generate the required messages, so compared to a service creation, the consuming of them can be relatively simple. Figure 3 shows how a service is described in the file [6, 158]

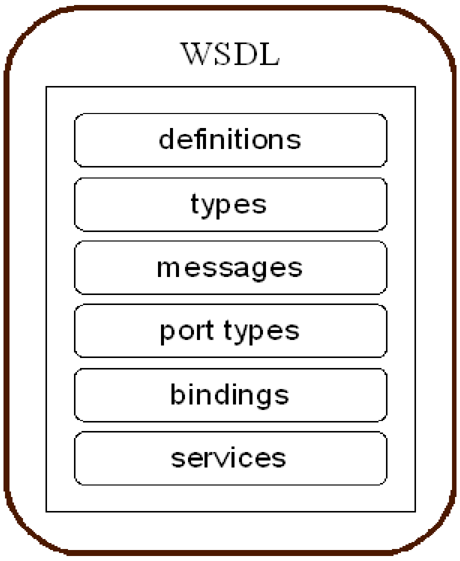


Figure 3: A WSDL file specification [3, 104]

As demonstrated in figure 3, the file contains enough information to both create and read the generated messages. The definitions is the root element of a WSDL file, containing the name of the web service, as well as other references used throughout the document. Types element is used to describe the different unit types already discussed in the previous page. A message is self-explanatory, consisting of either one way request or response. Port type element pairs the messages into operations, and can contain many of them. Finally, the service element provides the address for using the associated service.

UDDI stands for Universal Description, Discovery and Integration. It was created to allow developers share and, as the name suggests, discover the available web services. In practice, this results in an Internet accessible directory listing the web services available to the public. This can of course be implemented as to be usable by different persons with different access rights. The registry consists of a batch of XML formatted files.

UDDI, as well as the other technologies covered so far have been introduced to the SOA schema in figure 4. [6, 154]

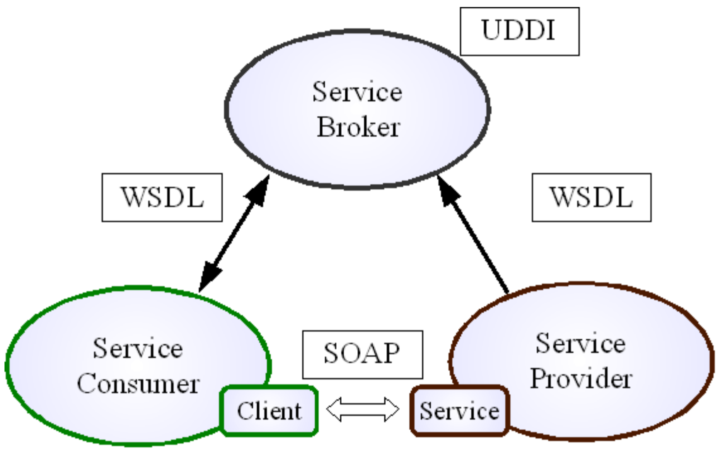


Figure 4: SOA and the popular technologies

The figure shows how each of the technologies has their own place in the architecture. All of the traffic in the figure are technologies based on XML structure, which makes the whole system very scalable and generalized. XML is covered better in the next chapter, as it is closest related to the SOAP technology.

The level of security for a web service can vary decisively. Web Services security can be inspected in three different groups, those of which are the Perimeter, Mid-tier and Back-office security layers. The Perimeter layer consists of the most commonly known issues in web development. This layer protects the service from external attacks by implementing features such as cryptography, authentication and authorization. The Mid-tier and Back-office securities are more advanced layers, and usually needed with corporate wide, large systems that need to defend against inside aggression and data security. These layers must consider the whole system infrastructure, and handle features like security association with servers and clients, and accountability by monitoring client behaviour. The project in this thesis requires only the Perimeter level security, as it is of a rather small scale compared to enterprise sized services. [7, 53-55]

2.4 Simple Object Access Protocol

Simple Object Access Protocol is a standardized method for service oriented data transfer. The data transferred with these messages are in XML format, so the data is therefore readable by any XML parser in any programming language. Most languages have their own SOAP libraries, so that the messages do not need to be manually parsed to read the containing message. This means that a SOAP message can be generated by one application and read by another, regardless of what language they were constructed on [3, p 43].

One of the features that make SOAP so functional, is that it most commonly uses HTTP, which is also used for the standard Internet traffic for HTML pages. Since the messages are basically text files, transferring them with the general transfer protocols does not differ from browsing the Internet. Therefore, these messages are as easy to transfer as regular Internet pages. The messaging uses the same port as Internet traffic, which means no extra configuration needs to be made on the firewalls. Although very convenient, this can also create some security concerns. Since the port for this traffic is open as a default, there is a possibility of mischievous data transfer. Although the messages are generally sent through HTTP, they can also be sent via FTP (File Transfer Protocol) or even e-mails. The similarities between soap messages and internet pages can be seen in figure 5. [6, 109]

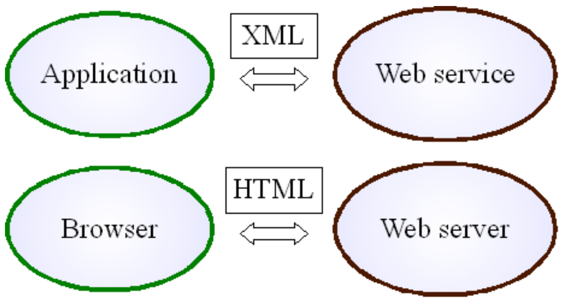


Figure 5: A web service compared to traditional web page in the Internet

This simple figure demonstrates how much these two proceduresare alike. The most notable change is the difference between HTML and XML messages formats, out of which the latter provides a standardized frame for the SOAP messages. For this to be useful, the messages and more specifically their contents must be formatted according to standardized rules. W3C has defined a set of datatypes, which includes String, Integer, Double, etc., the simplest of types. However, by combining these types and describing the resulting, new type in the WSDL file, a user can create more complex messages. The new type is a "complex type", and can, for example, be an Array containing several of the basic types. [3, 48; 8]

One slight drawback of using an XML syntax, is that it contains more excess data than some programming language specific messaging. Another kind of message made of only the necessary information, without any descriptive tags and attributes, contains only binary data. This kind of message is smaller and therefore faster to transfer. Because XML consists of a hierachial tag tree of text, a SOAP message consumes several bytes more than the somewhat more traditional binary messages.

On the other hand, XML is much easier to read by humans and the parsing speeds are increasing. This is because of parser improvements and especially since processor speeds increase exponentially. Internet speeds are also always increasing, and many of the web services are made of SOAP messaging between rather large company servers. Therefore, the speeds of the web service traffic is not necessarily even dependant on the client connection speed, but the connection between the web service provider and the underlying application logic that consumes the service. This dependancy is much easier to control since there are a lot less speed affecting variables to consider.

The security with SOAP messaging and web services is many times very important, especially when dealing with E-commerce applications. It has not been directly defined in any of the protocols, but a few options exist. A general solution would be to use the standard SSL (Secure Sockets Layer) or TLS (Transport Layer Security) to achieve encryption, authentication and integrity, but these do not fully cover some of the XML specific security issues [9, p 42]. The W3C, responsible for several Internet related standards, has created an XML Security Framework that contains technologies such as encryption and signing to cover these problems. There are also some issues with parsing a message by intermediares, which may cause alteration in the file. Canonicalization of the file to a proper format will allow the W3C Security to function properly, even after going through different intermediares like routers.[6, 197, 206]

2.5 Service Orientation in Software Development

There are several things to consider when developing service orientated software. The development is likely to take longer than normally, and when implementing SOA to an existing software it definitely requires additional time. The probable increase in ROI (Return Of Investment) should be a motivating factor for a company when facing these obstacles. This is expected to happen when the company's software infrastructure is modularized and abstracted, as they allows better reaction times in implementing changes and communication. [2, 55]

The steps in a service development process are the same as in any software creation, with the added challenge of needing to understand some of the business processes. This challenge needs to be dealt with in the beginning and designing phases of the development cycle. However, the cycle can basically follow any model designed for traditional software engineering, the steps can only have slight variations. The management of the services differ significantly from a traditional software's upkeep. Compared to a traditional library or component oriented applications, a developer must handle location independent, real-time management of the services. [10, 69-70]

Whether the selected development model follows the waterfall or an iterative process, the model generally doesn't cover the business side so crucial to service orientation. This part of service creation is called BPM (Business Process Management), and it is also an iterative process, better accomplished in cooperation with relevant business persons. Adding BPM to the process widens the scope from software development to an actual business process, concurrently shaping the organizational interactions and responsibilities. [11, 174]

Software testing with SOA applications can first be cumbersome when creating the initial components. Implementing SOA basically requires more code and more functions, unit types and other essential elements to consider. However, reimplementing an already tested service architecture on another system can be easier than regular software system. In this case, the dataflow models and architecture is already tested and has hopefully been succesfully proven. [11, 176]

Creating web services, to be precise, is to provide other users access to information the developer is providing. The users consuming the web service should have easy, coherent and systematic access to the data. The data itseld should be properly formatted and instructions to the service be available. Traditionally, the developer could decide how the data is presented and in what context. When adopting SOA, they might also have to consider how their data can be used or misused, if it's implemented somehow erroneusly, be that on purpose or not. The developers, in cooperation with business management, should consider which of the needed data and functions should be exposed and implemented in the web service. Making every possible feature accessible by the service would be pointless, as quite likely, some of them would never be used. [11, 187-168]

An issue to consider when creating a web service is if to build a client that consumes it. Depending on the project, it might not even be necessary, but in case it is created by the same developers as the service, it should be considered how well the client uses the available information consumable from the service. To attract more users to the service, the client application should be attractive and properly working. In a business-to-business case, this might not be so essential, since the usage of a web service might depend on company policies and contracts, instead of general attractiveness of your first client demonstrating the service.

2.6 Model View Controller

The Model View Controller structure is based on dividing the software source code into three different kinds of parts. The division allows the software elements to have loose coupling, similar to the situation described in the SOA before. The most important feature of MVC structure is that the three components are independent of each other. They do not work alone, as all three are required to form a usable and complete application. Modifying or completely replacing one component does not require changes in others. In software development, this provides good opportunities for a development group to divide tasks, as they can be completed almost independently without requiring constant synchronization between the developers. Another feature of MVC is that repairing bugs in an application usually requires attention to one of the components instead of fixing a recurring problem in several parts of an application. [12, 546]

The model component in MVC stands for the application's objects, that is, the data that the application uses. This data is often retrieved from a database and stored in the objects. The view component presents the data from the model. All of the presentational attributes and the user interface is defined in the view. The controller acts as an interpreter between the model and the view. Generally, the controller reacts to user input and based on the user's actions, communicates with the model accordingly. A model can also directly affect the view, without requiring any actions from the controller. The interactions of the components can be seen in figure 6. [12, 5-7]

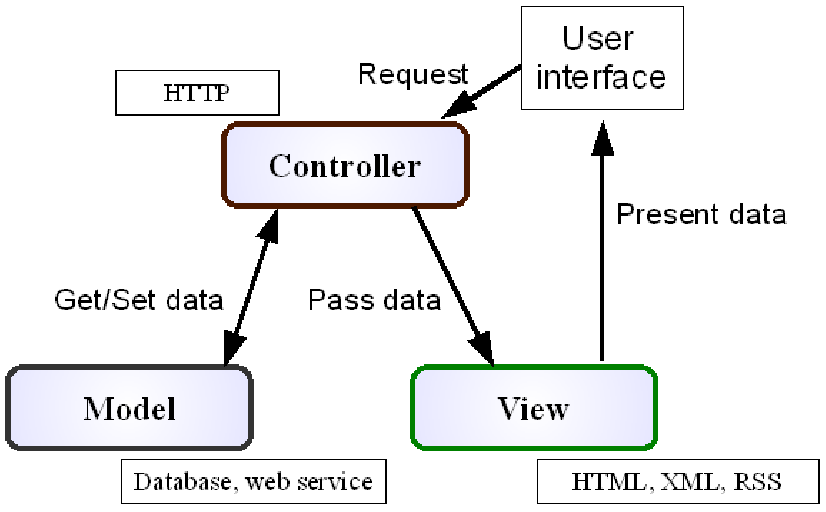


Figure 6: Model View Controller

The figure also provides examples of what technologies can be associated with each components, such as the database with model and html with the view. As the controller needs to deal with both of the two other components, it is generally the most complex one and the technologies involved hardest to define.

Some additional benefits of this system are that since the components are loosely coupled, any of them can be replaced in real-time. This can provide different layouts, input interpretation and therefore different user interface behaviour. The application data can be presented differently or different model could be used to retrieve different data. An application can actually display many of the views simultaneously, giving it a somewhat modular properties. Using MVC allows the developers to concentrate on one part of an application at a time and aims to give them a better way for maintaining the software [13, 471].

3 Project Outline

3.1 Project Timeline, Client and Workgroup

On my behalf, the project began in late January of 2009. The Mobile Laboratory R&D department in Metropolia was contacted by a company, Q4U, with intentions of producing a "reversed call queue" system. Me and another student, Henri Mäkinen, were chosen for the project. We began working on it, but after a few months, the original client withdrew from the project.

Since the project was well on its way, the Mobile Laboratory manager Jörgen Eriksson decided to follow through with it. After a small break we resumed working with the project and set the timeline for a working version of the application to be the end of July of 2009. The client became therefore the Mobile Laboratory R&D department in Metropolia University, and the project instructor was Jörgen Eriksson

The project was carried out by myself and an IT student, Henri Mäkinen. My part in the project was to create a client run in a web browser to communicate with the Asterisk telephone box environment. The Asterisk telephone box was set up and configured by mr. Mäkinen. Our common goals were the call logic and database planning, since those strongly affected both of our working areas.

3.2 Requirement Specifications and Test Planning

Since the original client was switched to being the Mobile Laboratory R&D department, the initial requirements for the application were rendered less important and were otherwise inconclusive. The the requirements were re-defined within the development group and the instuctor. Following are these requirements for the project, written in the beginning stages of the development. All the definitions and requirements have been written in the beginning of the project, with little explanations and descriptions added afterwards.

We have discussed about the use cases, roles and different views for different users. We have charted the most important features for this type of a system, and decided to focus the most on making the system as easy to use as possible and additionally extending the system with the more advanced profile management features. The use of this application is to be in a help desk type of environment of a small or medium sized business.

Use Cases

The following use cases describe the the call logic in Asterisk as well as the case flow in the Internet browser application. Figure 7 presents use cases for a customer calling the service and staying in the reversed call queue or waiting on the phone.

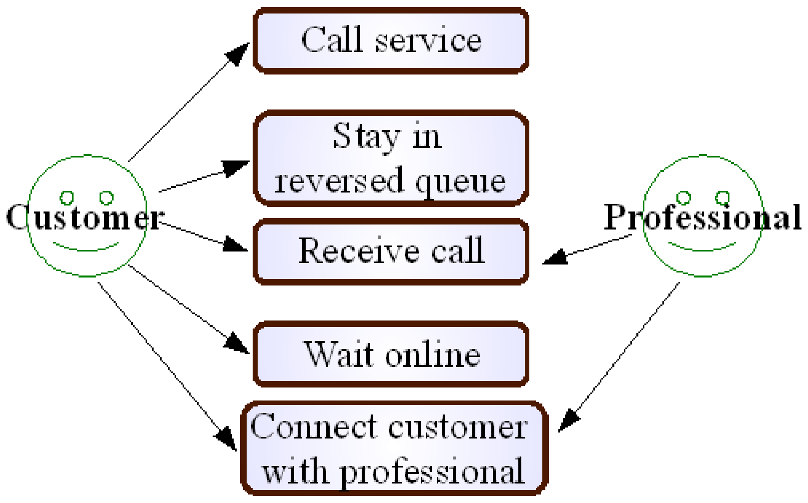


Figure 7: Use case diagram for essential customer functions

These are the most important cases, as they contain the integral idea of the whole project. The actual flow of events is presented later on in the chapter dealing with the Asterisk configuration, but this figure better describes the actual features.

The use cases for the users of the website application are similar, as the secretary and professional cases differ a little, and the administrator has additional options. The center of figure 8 is the application and the cases are from the website application user's point of view.

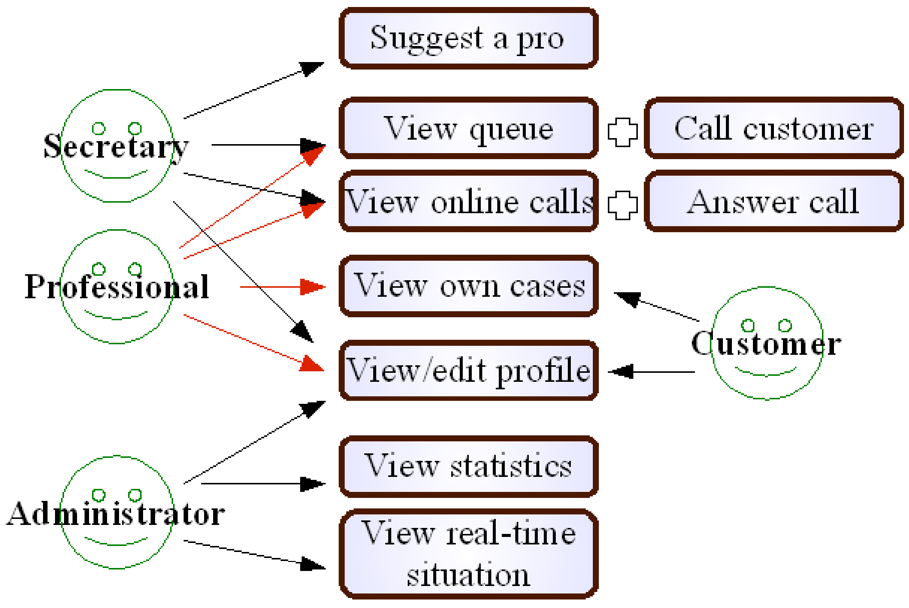


Figure 8: Use case diagram for the web site

This figure presents the main features needed from the client application to fulfill the requirements of a properly working resreved queue system. The possibility to view the queue, online calls and statistics are the most important ones, as they have been from the very beginning. Remaining features have been realized in the design or implementation phases.

User Interface

The most important feature of this application is for user to be able to view, manage, interact and response to the call cases created by the customer and the asterisk call queue system. The user in this case is a “secretary” or a “professional”. These are the most important roles, since they do all the work required by a help desk.

The application will display both the customers waiting for a return call (using the reversed call queue) and the ones waiting on the line (still on the phone). The secretary can direct these cases to different professionals, and the professionals will have their own page in which they see these suggestions. The professionals can also see the same page as the secretary, in case the secretary becomes unresponsive. All of these views should be displayed in real-time and the application should refresh itself automatically.

A call case will display user number, call time, and if available: name, number of calls, time of return call and number of return calls. The secretary should see which professionals are available, as the professionals will be able to set their status. The status will be automatically altered if the professional makes a call.

Professionals can call back to the customers and make comments in each of the call cases, using a simple form. The cases can additionally include a voice message recorded by the customer, which can be listened from the user interface. If approved by the customer, the professional can also reply with email or an SMS.

The secondary or extended features are for the “customer” and “administrator” roles. The administrator should be able to view all call details and histories for all of the professionals and their respective cases. This view is mainly interpreting the accumulated data in the database and to draw graphs based on them. A simple mobile phone interface will be created to allow the administrator and professionals to quickly and briefly view information such as numbers of customers in queues and possibly view case details.

All the users will be able to update their profile, either with a regular browser or a mobile phone browser. Since a customer can add and edit data in the database, certain security issues must be handled. The application must validate all the user, or at least customer, input and have proper authentication methods. The profile will define the user's personal information, options for the reply methods.

Technical Issues

The application is created on the basis of implementing service oriented architecture, to allow interoperability and different, more generic options in the development. The SOA will be implemented with web service technology, more specifically SOAP (Simple Object Access Protocol) messaging. This means there will be two different hosts, both of which can be on the same server though.

The first of which is the host with the asterisk application and the queue system. It handles all the database connections and heavier operations. The second host shall communicate with the first one via SOAP messages, display the received information and send small requests to the first host.

The application will also investigate how a model view controller approach can be implemented with this kind of system. The first host will contain the model and the control, whereas the second host has control and view parts. More specifically, the control parts in both of the hosts generate and interpret the SOAP messages.

The realtime communication with the database requires automatic refreshing, and it will be implmented with AJAX (Asynchronous Javascript And XML). The playback of the recorded messages form the users will be done with a free, open source flash component.

Test Planning

To see how the requirements are met, there must be proper planning for the testing scenarios of the end result. The testing is done with the help of a few persons, since this system is designed to be used by multiple persons concurrently. One of them can be an ignorant, "customer" type of user, who does not know much about the system. Two other users should be people properly instructed on how the secretary and professional views are operated. A few default case flows will then be implemented to see if the application meets the above mentioned requirements.

The criteria for the test to meet can be dervied from the previous parts in this chapter. It should be noted, that some features, especially the ones defined in the user interface section, can eventually be discarded or modified. The overall user experience is more important than blindly implementing every single detail defined beforehand.

3.3 Importance of Different Features

Since this type of an application has the potential for becoming huge and packed with features, it is needed to specify the most imoprtant ones of them. By specifying the core features and first concentrating on those, it will be easier to continue the development by adding more advanced and extra elements to the application. The goal for the project in the scope of this thesis was to create a functioning demo to present the possibilities achievable with the currently available open source technology.

The most important feature of the application is naturally displaying the reversed call queue and the ability to interact with it. This is the one feature the application would be useless without. This includes the online queue, where the customer is not waiting to be called back, but is waiting to someone to answer the call.

The remaining features have been built around this idea to help both the professionals and customers use the application. It can provide the users with better management of customers or professionals, and the ability to mine data from the call records. One important aspect of the system is the controlling of the cases generated for the calls. Development of the case and customer control mechanisms steers the application towards a customer management system, instead of being just a mere "point and click" list of events.

Instead of just having a number of concurrent users browsing a list of arbitrary phone numbers, there is a profiling system for each user. This is of course optional for the customers, but efficient nevertheless. The user profiles help the system in collecting data, and allow this data to be used in calculating statistics and viewing the real-time situation by an administration. The user profiles are stored in the database and the authentication is done via a simple login screen.

Most of the extra functions help the professional, but a few features can ease the customer's interaction with the system. The customers can set contact method settings or provide the company information about themselves. These features are secondary though, for they are only optional for the customer. The usability of the call service for the customer is the main area of interest, and it cannot be expected for the customer to go through an Internet registration, but if they feel it to be of any use, they can choose to do so.

3.4 User Role Specific Features

There are 4 different user roles that each have different views and permissions. The two most similar of those are the "secreatary" and "professional" roles, which both have access to the call queues and user info. The difference being, that a secretary can suggest some professional to a case, and a professional has a view of the cases suggested to them. The professional users have a specific database entry, which informs both the secretary and admin if the professional is currently at work. This allows the secreatary to redirect calls to certain professionals and the admin to see who are at work on any given time.

The user role with most rights, is the "administrator". An administrator can view the call queues, the professional specific queues, customer history, case history, and a set of statistics. The statistics include a variety of calculated averages per time, average call time, answer time and others. These values are calculated everytime the page is opened and are therefore dynamic. These values are read and interpreted from the case histories, so no static list of statistics is ever written in the database.

The customer can register themselves to the site, linking a phone number with a name, address and an e-mail address. This allows them to provide more information to the system. They can also choose additional contact methods from their profile page.

4 Service Development

The service orientation in the application development had a considerable impact on the development process and time, as well as increased the total challenge considerably. Developing the application with SOA resulted in constructing the application in two different phases. These phases would be to first create a server side application to provide the service, and then implement a client application that consumes the service. The distinction between the two applications is covered more precisely in chapter 5. Ideally, the development of the two would overlap only slightly, but there was a need in this project to provide testing capablities to the Asterisk system as soon as possible. This scenario resulted in having to concentrate on both sides concurrently. The order in which the topics are covered is similar to that of in chapter 2. The first topics are of the lowest layers of system configuration and the last ones cover the user interface, leaving the service and application layers in between.

4.1 Asterisk Configuration

Asterisk was configured by Henri Mäkinen, but I should cover some of the basics in my thesis to clarify and explain the system as a whole. The Asterisk runs on a Linux platform and is controlled with a series of call redirection patterns. Even though this was not configured by me in any aspect, I participated on designing the call logic, which dictates the configuration goals. The call logic is presented in figure 9.

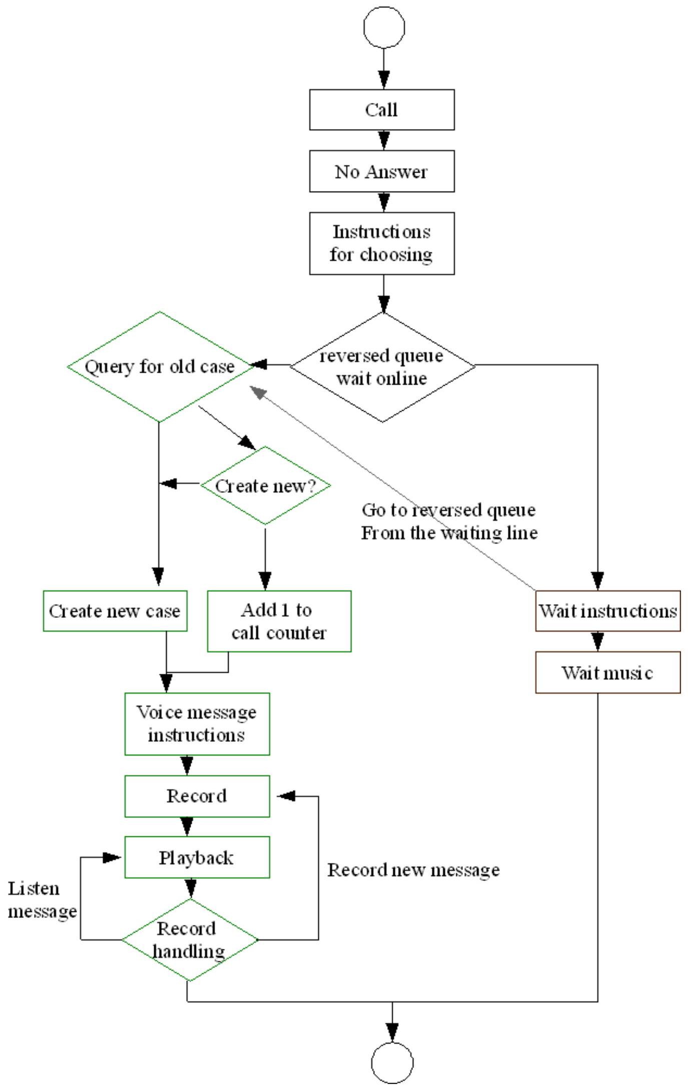


Figure 9: The complete call flow diagram

The figure displays how the customer's call is redirected to different paths depending on their actions with their handset. The system plays audio files instructing the customer to press a button on their phone to choose, for example, the call back queue or waiting online.

The main core of Asterisk call logic is defined in so called dialplan files, which contain code that instruct the phone switch to handle the calls. These files are the customizable essence of any Asterisk system, what makes each different system unique. They are constructed in a procedural fashion, which also makes inspecting them easier. An exaple of the code in Figure 10 adds a caller to the online waiting line, stores the event's information in the database and plays music for the caller to listen while waiting.

```
[online wait service]

exten => s,1,Answer() ;welcome to the wait service
exten => s,2,Set(aika=${STRFTIME(${EPOCH},,%Y/%m/%d %H:%M:%S})

exten => s,3,MYSQL(Connect onlinedb localhost acd acdpass acd) ;login to the database
exten => s,4,mysql(query online1 ${onlinedb} insert `callers` set ... ;mysql clause continued

exten => s,5,Background(omat/haluatjonoon&omat/paina&omat/1)
exten => s,6,waitExten(1,m(default))  ;wait for the exten, while playin the "on hold"
exten => s,7,goto(6) ;if playback ends, go to start

exten => 1,1,answer() ; the user presses "1" if needs to go to reserved queue
exten => 1,n,mysql(query poisto ${onlinedb} delete from `callers` ... ;mysql clause continued
```

Figure 10: An Asterisk dialplan for waiting an answer to the call

The figure covers the 4 basic pieces of a dialplan, which are:

- context – divides the dialplan into isolated categories, contains the following
- extension – a procedural set of instructions the dialplan follows
- priority – one (of possibly many) step in an extension, defines an application
- application – performs an action: similar to a function with parameters

The example dialplan is in the context of the online wait service, and the first line of code sets the extension named "s". The first priority is set to be the built in Answer application of the Asterisk system. With the use of these elements, Asterisk is configured to fit the needs of the call logic. [1, 119-124]

Asterisk can be externally controlled with various methods, such as opening an url with PHP GET data defining an action, writing a "callfile" to a folder surveyed by the system or, in this project, using the Asterisk Manager Interface. The last solution is implemented by opening a socket connection to the built-in HTTP server in Asterisk and issuing the desired commands directly. All of these options are protected by requiring either login credentials or disk write permission. [1, 228]

4.2 Database Structure

The database was designed in unison within the development team, as it was the one element that tied our work together and we both had an impact on. Because the main core of the application works by polling the database for calls and user data, the structure is very important. Most of the tables in the database must contain just right amount of information, the most essential data that is available. Some tables should, at the same time, store as much information as possible, to allow proper call detail records and the generation of statistics. The database structure presented in figure 11 should therefore be optimized for these purposes.

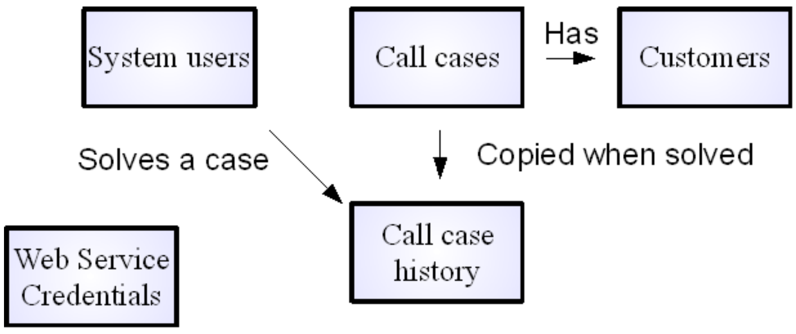


Figure 11: Database tables

This figure siplays only the tables and gives and idea of their relationships, the complete structure can be seen in appendix 1. The Asterisk telephony system is programmed to write some of the the call records in to the callers table shown above. This is the information needed to establish a call back to the number it was originated from. The Asterisk collected call information is:

- Number – from which the customer made the call
- Time – the time when the call was made
- Unique id – generated by Asterisk, based on time and random number
- Online – defines if the customer is in the reversed queue or waiting online
- Channel – is used by Asterisk to correctly connect the calls

This information is read in certain intervals by the PHP application. The application creates objects from the call information and they are used to display the data to the user. The main information in this table is generally created by Asterisk and read by the application, as each row represents one call and a case. Other information is written by the application, for example if the user decides to create a case manually.. These rows can be distinguished by the fact that the channel field has no value.

The remaining tables on figure 11 are written, modified and read by the application. The user table contain information about everyone using the system but customers. Customers have their own table to keep them apart from the users and because they have slightly different needs in terms of data storage. The customer table contains data about their specific settings and a descriptive field generated by the users. What users and customers have in common, is the credential storage. Each are authorized to the site by retrieving and matching the excisting values in the database. The company authentication uses this same structure in the companies table. The remaining table, callhistory, contains rows copied from the calls table, with two additional columns. This data is copied when a case is solved by a professional, with the solve date and solver id added in the end of the row.

Though quite simple, the database provides the needed functions to store the needed data still separate the calls from user profile functionalities. As shown in the figure, the database tables contain relationships, but could be split into two different ones in the case in which either the call data or user interactions would require too much bandwith or processing power. In this case, the se two different sets of data could be processed separately.

Classes

The program utilizes object oriented programming in dealing with the call cases and user management. The classes are almost a direct representation of the equivalent database tables, and therefore highly related to the databse structure and logic. The classes represent the call cases, users, customers and a connection to the MySQL database. Since the user and customer entities are very much alike, the customer class extends the user class to avoid reprogramming and bloating of the code. As shown by figure 12, the application does not contain many classes, and these classes are exclusively used in the server application.

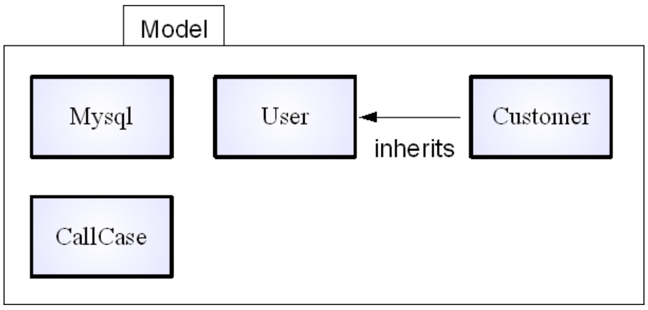


Figure 12: Classes

These classes provide the required methods for manipulating the database and providing the service with the needed data. The idea of object oriented programming is not so exhaustively implemented in the appication as a whole, since it is separated from service orientation. These two approaches to software development easily co-exist. In fact, service oriented architecture has been developed on, or at least heavily influenced by the principles of obect orientation. The main difference between the two design principles is generally their scope. As object orientation is usually focused on single application level, the service orientation has a larger, system wide coverage. [2, 446-448] Figure 13 demonstrates the orientation scopes in a project similar to this.

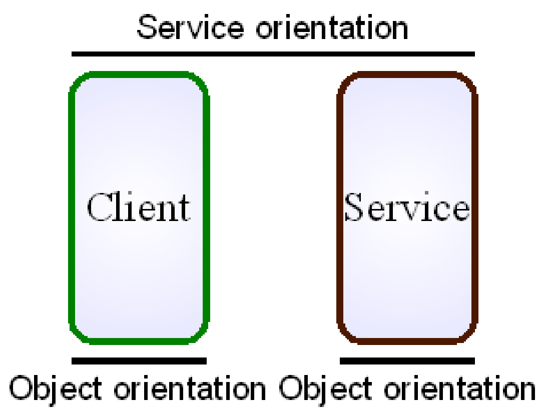


Figure 13: Comparison between Service and Object orientation [2, 447]

As seen in the figure, the two different application utilize their own OOP models, as it is generally a bad idea to exchange objects via a web service. Object orientation covers the server and client applications independently, as they can many times be constructed by different developers. As a whole, these two loosely coupled and independent applications together implement service orientation. In the case of the reserved call queue application, the client side does not utilize OOP, but very well could, if it had been beneficial in the development.

4.3 Web Services

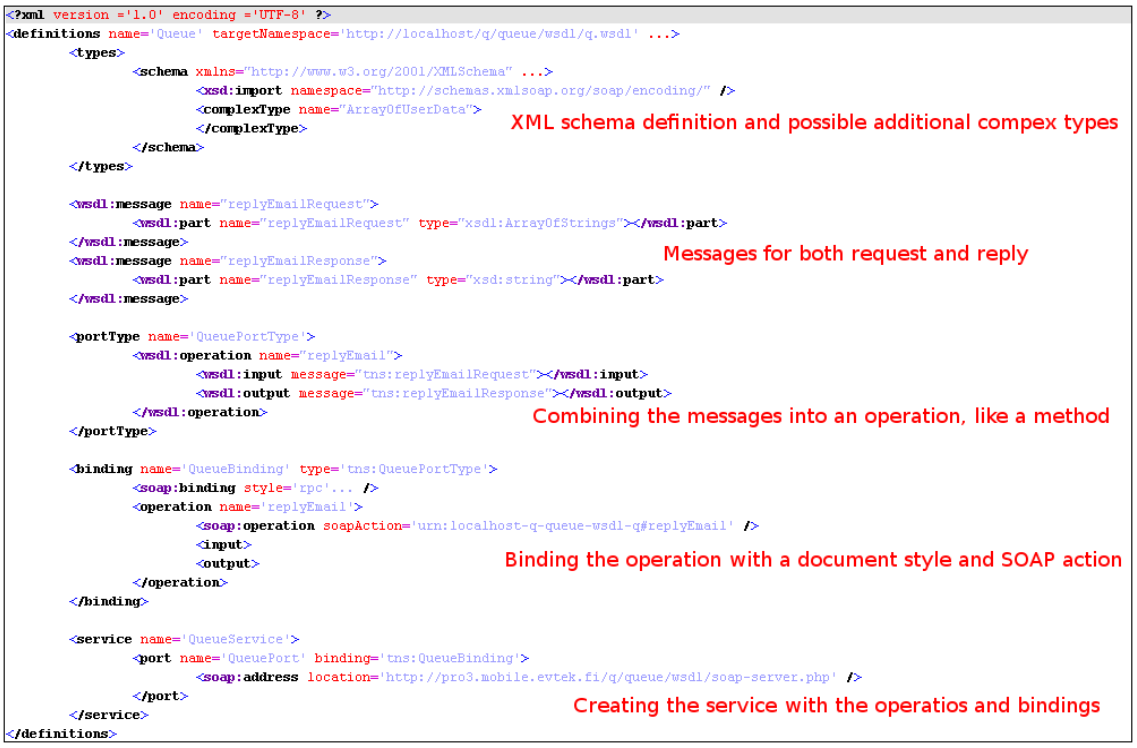
The service oriented architecture and web services with this application were originally thought to make the application technologically more capable and generally accessible. Being more general means that it could be accessed also by other means than the specific PHP programmed application. The creation of web services introduced both problems and later benefits to the system as a whole. They also divided the application into two distinct parts. So far, I've presented the system as Asterisk and my application, but the application itself is made of two parts.

The application is divided to a portion which provides the web service, and a portion that consumes it. These could be considered as being two different applications, as at least the consuming part could be replaced with anything without affecting the providing one. After all, that is is the core idea of SOA and especially loose coupling. In fact, any application can connect to the system, provided that it holds proper authentication credentials. The two different parts shall now on be referred to as client and server application, and are more specifically described in the next chapter.

SOAP & WSDL

PHP has a built-in library that allows the developer to use SOAP by accessing the several classes and methods, the most important ones of those being the client and server classes. It would be difficult and time consuming to manually construct the messages, but instead, they are automatically constructed by a set of rules defined in the WSDL file. SOAP messages are practically always computer generated, and for in this case, the available functions are used to assign variables to the pre-defined functions the web service provides.

Web Service Definition Language is used in the XML formatted file providing all the needed information for an application to consume the web service. The file provides information about the web services. Figure 14, displays a very simplified version of the wsdl file used in the application. The actual file would bee too large to print on this page.


XML schema definition and possible additional compex types
Messages for both request and reply
Combining the messages into an operation, like a method
Binding the operation with a document style and SOAP action
Creating the service with the operatios and bindings

```xml
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions name='Queue' targetNamespace='http://localhost/q/queue/wsdl/q.wsdl' ...>
        <types>
                <schema xmlns="http://www.w3.org/2001/XMLSchema" ...>
                        <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
                        <complexType name="ArrayOfUserData">
                        </complexType>
                </schema>
        </types>

        <wsdl:message name="replyEmailRequest">
                <wsdl:part name="replyEmailRequest" type="xsdl:ArrayOfStrings"></wsdl:part>
        </wsdl:message>
        <wsdl:message name="replyEmailResponse">
                <wsdl:part name="replyEmailResponse" type="xsd:string"></wsdl:part>
        </wsdl:message>

        <portType name='QueuePortType'>
                <wsdl:operation name="replyEmail">
                        <wsdl:input message="tns:replyEmailRequest"></wsdl:input>
                        <wsdl:output message="tns:replyEmailResponse"></wsdl:output>
                </wsdl:operation>
        </portType>

        <binding name='QueueBinding' type='tns:QueuePortType'>
                <soap:binding style='rpc'... />
                <operation name='replyEmail'>
                        <soap:operation soapAction='urn:localhost-q-queue-wsdl-q#replyEmail' />
                        <input>
                        <output>
                </operation>
        </binding>

        <service name='QueueService'>
                <port name='QueuePort' binding='tns:QueueBinding'>
                        <soap:address location='http://pro3.mobile.evtek.fi/q/queue/wsdl/soap-server.php' />
                </port>
        </service>
</definitions>
```

Figure 14: A highly simplified WSDL file

The figure presents the most important features of a WSDL, but contains only one method in it and is lacking much of the actual code. This file would give an error when executed, but was simplified to increase the readability.

The server side application has it's functions written in PHP, just like any other program. Instead of directly calling the functions, they are added to the SOAP server and are described in the WSDL file. These functions return values, that are formatted to match the types also defined in the file. The client application reads the file with the SOAP client library, and the functions defined in the server are mapped to SOAP client instance.

4.4 Model View Controller with Service Oriented Architecture

The decision to use MVC in the projects was made on the basis that it is relatively large and the approach keeps the structure of the application coherent. Implementing MVC to the application added yet another extra layer of complexity to it. However, since the software is indeed divided into two parts, the MVC is also halved. This is actually a rather good thing, and is best explained by figure 15.

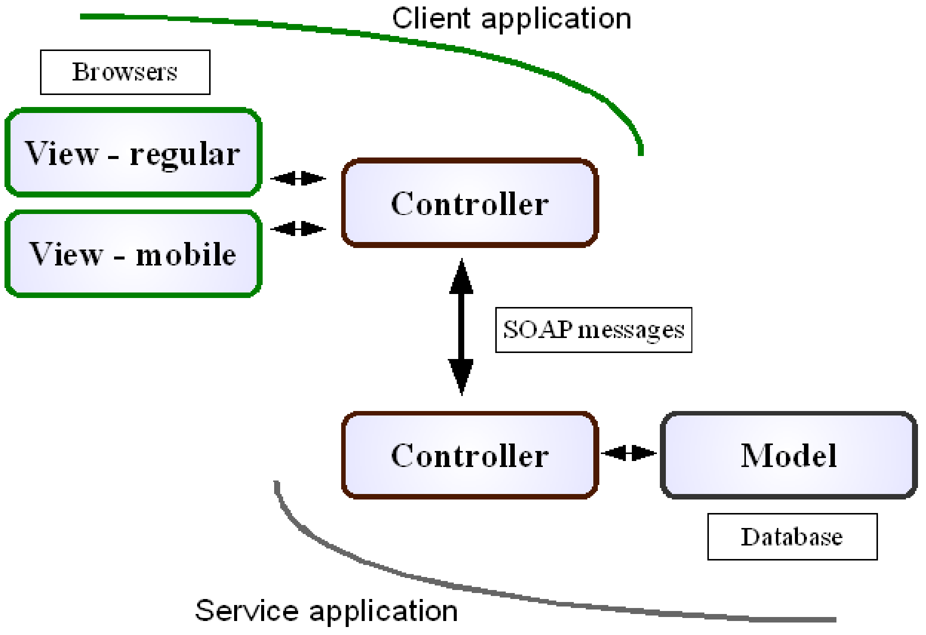


Figure 15: Division of the MVC into two parts due to service orientation

The applications divide the MVC components so that since the server deals with the data and calculations, it implements the Model. The models are represented by the classes presented earlier. Whereas the server deals with data processing, the client displays it. View is naturally left for the client to handle. The more deviating part here is that both the server and client have Control components. These communicate with each other by using SOAP messages.

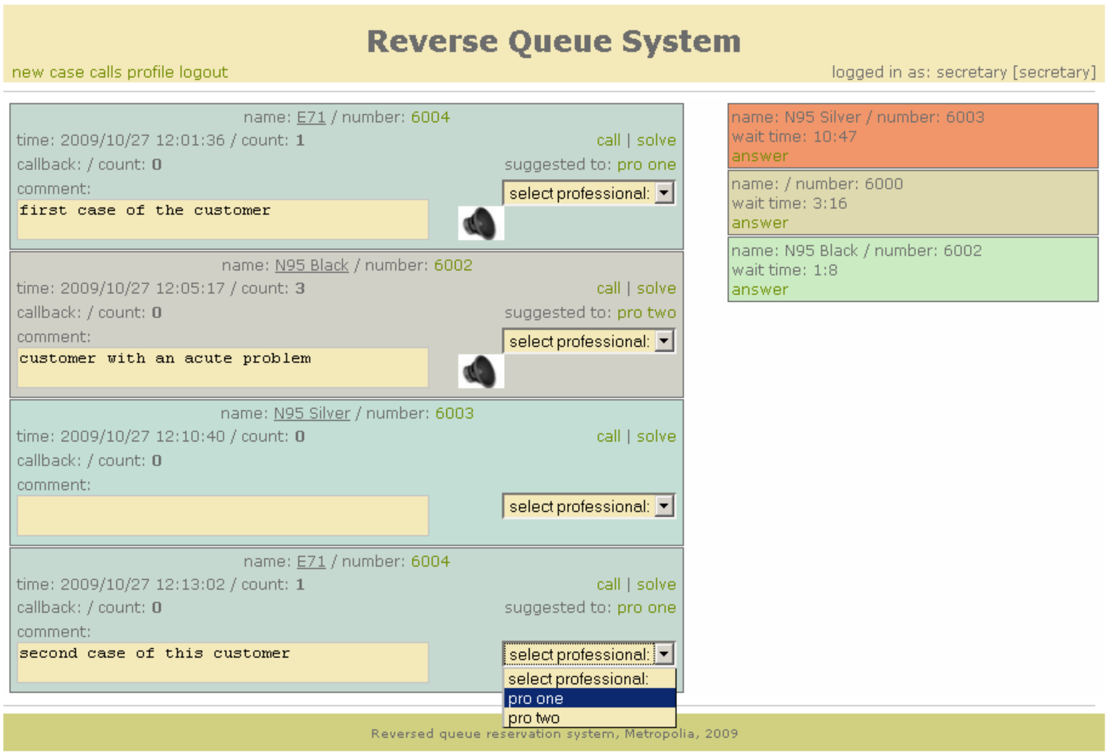
A clear example of the usefulness of the MVC logic is that the application contains two distinct views for both the regular web browser, as well as for the mobile browser users. These views are designed and constructed separately, and the controller of the client application chooses which view to present to the user. With this approach in mind, it becomes evident that there can be as many different user interface designs as needed, without having an effect on the controller and underlying code.

5 Client Application

5.1 User Interface

The user interface is an essential part of any application, even though it might not require as much coding and thinking as the underlying architecture. This is the case in this project, but in other situations it might even require more effort to develop. This project had a clear goal of concentrating less on the user interface development.

The resulting user interface contains different page availability and slightly different presentation of data, depending on the user mode associated with the users credentials. Figure 16 presents the small variations between the secretary and professional views.



Reverse Queue System

new case calls profile logout

logged in as: secretary [secretary]

name: E71 / number: 6004
time: 2009/10/27 12:01:36 / count: 1 call | solve
callback: / count: 0 suggested to: pro one
comment:
first case of the customer
select professional:

name: N95 Black / number: 6002
time: 2009/10/27 12:05:17 / count: 3 call | solve
callback: / count: 0 suggested to: pro two
comment:
customer with an acute problem
select professional:

name: N95 Silver / number: 6003
time: 2009/10/27 12:10:40 / count: 0 call | solve
callback: / count: 0
comment:
select professional:

name: E71 / number: 6004
time: 2009/10/27 12:13:02 / count: 1 call | solve
callback: / count: 0 suggested to: pro one
comment:
second case of this customer
select professional:
select professional:
pro one
pro two

name: N95 Silver / number: 6003
wait time: 10:47
answer

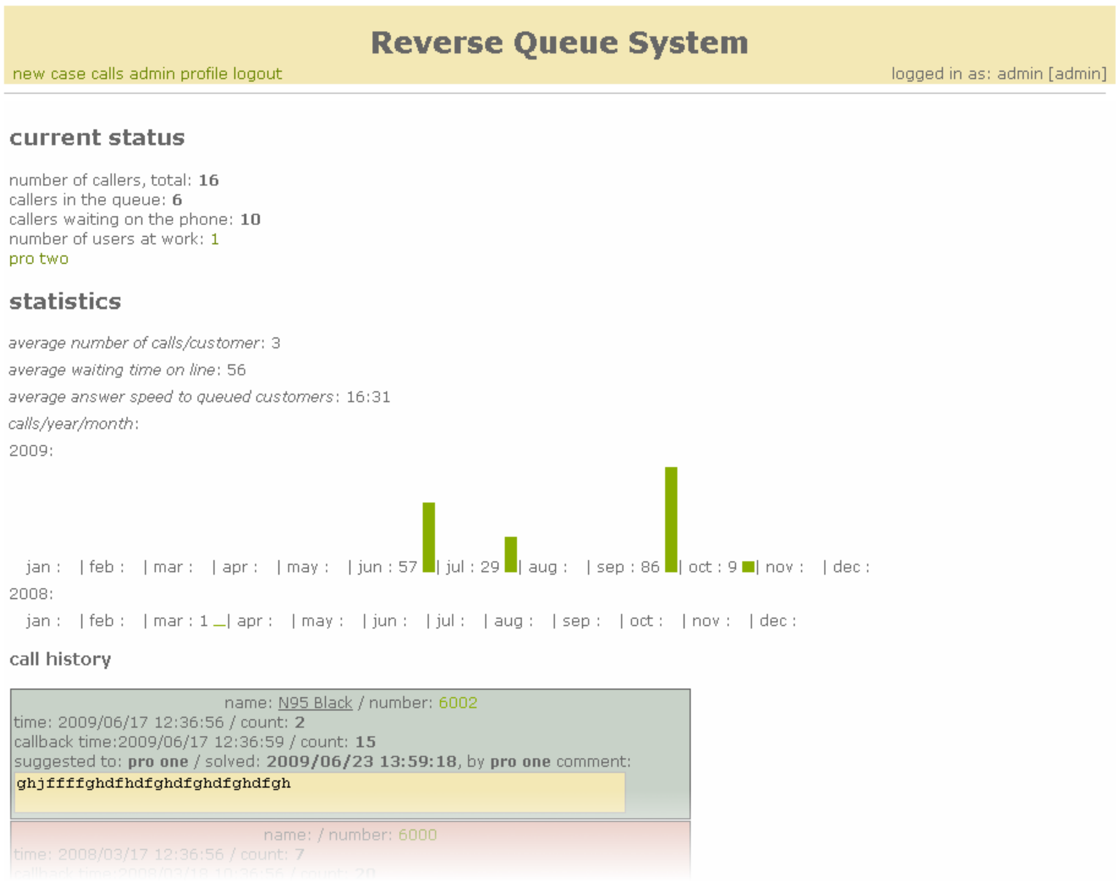
name: / number: 6000
wait time: 3:16
answer

name: N95 Black / number: 6002
wait time: 1:8
answer

Reversed queue reservation system, Metropolia, 2009

Figure 16\: User interface in the secretary mode

The differences between these two are only slight, the main ones being that the secretary can choose which professional to suggest for a case, and a professional has a possibility to view the cases suggested to them. The biggest difference with the different user modes is the administrators page shown in Figure 17.



Reverse Queue System

new case calls admin profile logout

logged in as: admin [admin]

current status

number of callers, total: **16**
callers in the queue: **6**
callers waiting on the phone: **10**
number of users at work: 1
pro two

statistics

average number of calls/customer: 3
average waiting time on line: 56
average answer speed to queued customers: 16:31
calls/year/month:
2009:
jan : | feb : | mar : | apr : | may : | jun : 57 | jul : 29 | aug : | sep : 86 | oct : 9 | nov : | dec :
2008:
jan : | feb : | mar : 1 | apr : | may : | jun : | jul : | aug : | sep : | oct : | nov : | dec :

call history

name: N95 Black / number: 6002
time: 2009/06/17 12:36:56 / count: **2**
callback time:2009/06/17 12:36:59 / count: **15**
suggested to: **pro one** / solved: **2009/06/23 13:59:18**, by **pro one** comment:
ghjffffghdfhdfghdfghdfghdfgh

name: / number: 6000
time: 2008/03/17 12:36:56 / count: 7
callback time:2008/03/18 10:36:56 / count: 20

Figure 17: Administrator view

The administrator can view the current status of the system, which basically summarizes the reversed call queue situation. This data is directly retrieved from the same database table that the real time call pages use, as well as the status of each professional user. The statistics information is retrieved from a call history table and then calculated with simple formulas to display some of the information essential to an administrative user.

All of these pages and navigation to them are dynamically loaded by the controller part of the application, based on the user mode associated to each user's profile. The user mode is set when the user logs in to the system, and is from then on stored on the client server as a session variable. The user needs not to worry about the user mode or the available pages, the correct options are provided automatically and preserved until logging out of the system.

5.2 The Displayed Call Data

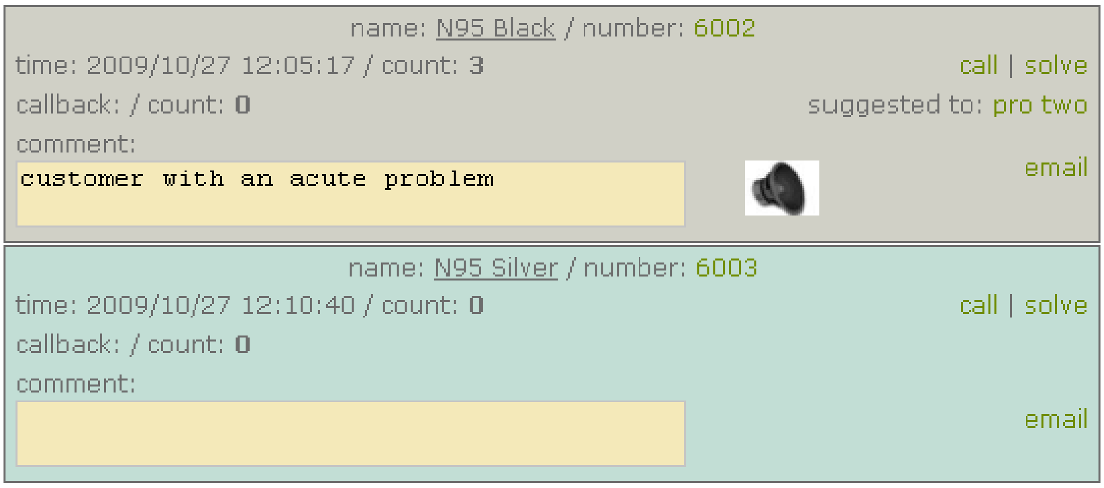
To make using the system a user friendly experience, the data to display on the call queue list needs to be carefully chosen. Since there can be several cases on the screen at the same time, the application should not list every available piece of information and meta-data to the user,s since too much characters per a "case box" would confuse the them and therefore slow down the whole system. The following data was thought of being most essential:

- Name & number
- Call time & callback time
- Number of retries
- Optional voice message

These values are automatically generated by the Asterisk sytem or fetched from the database by the server application. Comments and suggestions are user generated via the client application.

Depending on the user's role, the boxes hav a slightly different layout, but in each case display user mode relevant data. For instance, the secretary can see a list of online professionals to suggest the case, but the professionals can see only to whom a case is suggested. The professionals can naturally see a list of cases suggested themselselves.

Figure 18 displays two common call case boxes viewed with the professional user mode.



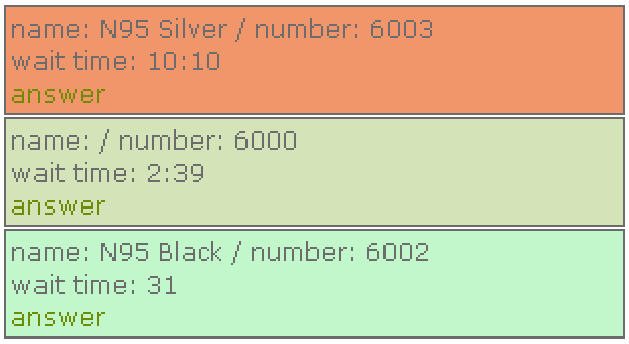
name: N95 Black / number: 6002
time: 2009/10/27 12:05:17 / count: 3
call | solve
callback: / count: 0
suggested to: pro two
comment:
customer with an acute problem
email

name: N95 Silver / number: 6003
time: 2009/10/27 12:10:40 / count: 0
call | solve
callback: / count: 0
comment:
email

Figure 18: Call case visualization in the user interface

The main focus on these boxes are on the wait time of the caller, followed by the user id or phone number. As the customer's call amount increases, the box gradually changes it's background color towards red. The red color naturally implies something is wrong and should get the attention of the user. Changing the color of a box reflects actually the state of the whole call queue system, not any individual case. A completely red screen implies the system has a number of customers that have passed the optimum waiting time. When considering an individual box, the identification of the caller is key. This can be more or less important, depending on the systems purpose and the area of use. If the user is identified, he can be redirected to a proper professional and his possible previous cases can be examined before returning the call.

The idea with a box of a person waiting in line is similar, but the presented (and available) information is smaller. The color of these boxes change based on the wait time, since it is more important for a customer waiting on the phone. The effect of this can be seen in figure 19.


name: N95 Silver / number: 6003
wait time: 10:10
answer

name: / number: 6000
wait time: 2:39
answer

name: N95 Black / number: 6002
wait time: 31
answer

Figure 19: Boxes for a customer waiting on line

As implied by the color changes in above figures, wait times and call retries are important, as they can affect the customer's patience, state of mind, availability or even the customer's seriousness. Commenting and listening to the customer's audio messages are almost self explanatory, since they can both provide insight to the customer's case or problem. Depending on the customer or commenter, they can also be useless. Because of this, they are not considered as important as the factual data.

Real-Time Responsiveness

The user interface needed to present the call data in real-time while not requiring the user to constantly hit the browser's reload button. The most obvious solution to this problem was to use AJAX tehniques. In this case, as the requirements for this technology asre reasonably low, only the Javascript portion of the abbreviation was needed for implementation. There is no need to deal with or parse any XML messages at the Javascript code. Despite of this fact, the use of the term AJAX is still considered valid, as the ActiveXObject in Internet Explorer or XMLHttpRequest in other browsers are used by Javascript to establish the asynchronous connection between the server and the client. [14, 14-15]

The automatic reloading of a page can be achieved rather simply. A small amount

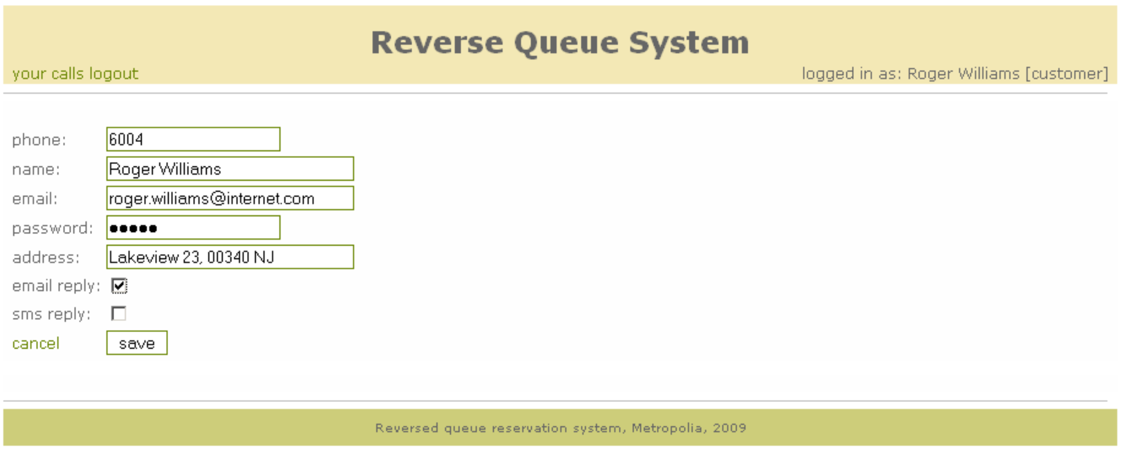
Javascript can make the page refresh itself at regular intervals, with a standard JS setInterval function. Refreshing the whole page is not optimal though, since it produces the flicker effect and resends PHP's GET and POST data. To circumvent this problem, the Javascript needs acces to the DOM (Document Object Model) of the page. There, it can modify the current page, and in this case, alter a certain div element of the Html structure. Only the essential part of the page is therefore reloaded, leaving the actual page and unaltered.

5.3 Client Device Differentiation

To allow different views on different devices, the application must know what type of device is accessing it. Two reasonable options were available to implement this challenge. First was to identify the screen size of the accessing device with Javascript and then send this data to the server. The server would then provide a page suitable for that device. The other option, which was used in this case, was to read the header information sent by the device browser, which indicates the browser type and version it is using.

When a browser requests a web page from a server, it sends information about itself in the request header. This information contains the browsers type and version in a "user-agent" field, such as "Firefox 3.5.3" or "MSIE 6.0". The actual string sent by any browser is larger and the values here are simplified to give the idea how the browser detection works. By interpreting this information, the server then responds to the browser's request by sending it the correct page.

In this case, there are two different views. The first one is the regular view intented to be viewed with a desktop browser and a standard display with at least 1024x768 resolution. Figure 20 shows the profile page for a customer in a regular desktop browser



Reverse Queue System

your calls logout

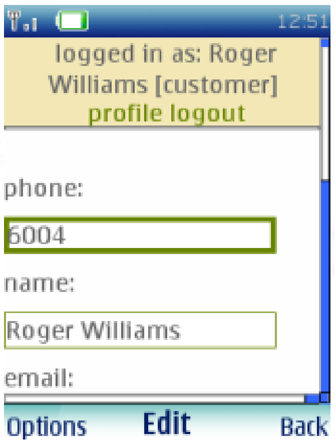
logged in as: Roger Williams [customer]

phone:	6004
name:	Roger Williams
email:	roger.williams@internet.com
password:	●●●●●
address:	Lakeview 23, 00340 NJ
email reply:	☑
sms reply:	☐
cancel	save

Reversed queue reservation system, Metropolia, 2009

Figure 20: User interface for desktop browsers

This figure can be compared to the mobile view formatted for the common 240x320 resolution mobile phone screen. The mobile site was added mainly for the possibilites for the customer to change their information and the administrator to have a real-time grasp of the situation, regardless of the location. The whole site can actually be viewed with a cell phone, but the essential views can be seen in figures 21 and 22.



12:51

logged in as: Roger Williams [customer]

profile logout

phone:

6004

name:

Roger Williams

email:

Figure 21: User interface for mobile phones

The browser view is indeed simple and cannot handle much data at one time. However, the mobile accessability of a website is becoming increasingly valuable, as the number of mobile phones in the world easily surpass the number desktop computers and the mobile phones are equipped with better browsers, larger screens and innovative typing solutions [15, 42].

The creation of a mobile friendly site is of course yet another time consuming obstacle in the development process. Even though a mobile site contains less content because of the screen size restrictions, the huge amount of different devices and standards can make the design complicated. The design can have different orientations, such as concentrating on keeping the site so simple that as many as possible devices can access it, or concentrating on some niche area that can provide better and more complex feaures, but is restricted to specific devices. [15, 44 – 47]

5.4 Security

The security issues with this application are relatively important. In addition to the more traditional threats like SQL injections, the application needs to authorize all the web service messages as well as the individual users of the application. Since the SOAP messages travel using HTTP, standard SSL can be used to secure the messages in between the client application and the web service [9, p 42]. However, SSL has not been implemented since the application is currently being used inside an already secured Intranet and outside access is not allowed.

Every input that can be made the customer, or in other word, those forms that are submittable by any Internet user, are evaluated before inserting them to the database. These inputs must be evaluated, as they can possibly have harmful intent, as they can be those of someone trying to hamper the system. Most of this is done with PHP's existing MySQL escape function, ignoring characters that would seem to the database as malicious commands. The possible actions allowed to the professional users are not all validated with such an effort, because it is assumed that the employees themselves will not attempt to hack and destroy the system. These security features mean that only the Perimeter layer is implemented, thus providing no security against internal attacks. This approach was taken in consideration of the organization size proposed for using the application.

Authentication of Individual Users

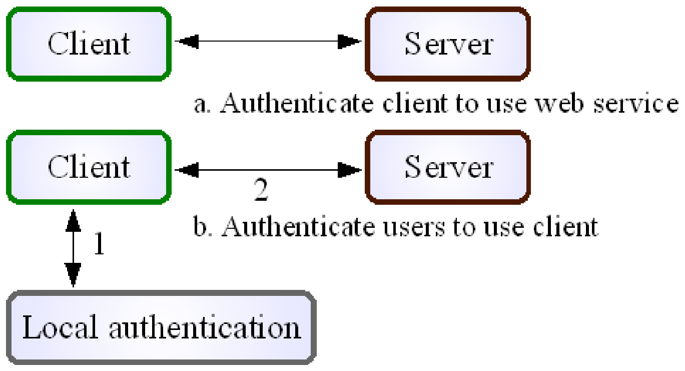
User and customer validation is handled by the web service consuming, client part of the application. The client program prompts for the credentials and based on those, decides whether the person login in the system is a customer or a professional user. In this case, these credentials are requested from the web service, but in theory, the client application can decide where to store the credentials. The approach implemented in this client application derives form the fact that both the client and the web service are hosted and managed by the same individuals, and the data for both of them are stored in the same database.

Since the user roles and management is handled completely by the client application, a company developing one can implement their own user logic. They can then decide which web service data and methods they like to assign to their users. From this it is evident that the web service actually provides a set of functions to communicate with the server side application. It does not care if the person using the system is a customer or something else. The client application must make sure the data transferred to and from the web service is correct. However, the web service does not let any client to connect to it; it has its own, separate means of authentication.

Client Application Authentication

To prevent misuse of and unauthorized access to a web service, it must require and process credentials from the consuming client application. The client consuming the web service is provided with these credentials and they are sent separate of the actual user authentication. It is important to keep these two processes separate, to allow the loose coupling of SOA, as mentioned earlier. Had these authentications been dependant of each other, the web service would be much harder, if not impossible, for other developers to consume.

Generally, and in this case, the credentials are stored in the SOAP message's header part, before the actual body of the message. The server receiving the SOAP message first checks the header part before reading or processing anything from the messages body. If the received data matches that of stored in the database, the application continues to process the body of the message. If the authentication fails, the server side application returns an error message to the client application, and doesn't allow access to any of the web service functions. These two authentication steps can be done in the opposite order, if the client developer chooses to implement the individual user's authentication by their own means. Figure 22 presents the authentication flow in two separate steps a and b, of which either one can be carried out first.


Client
Server
a. Authenticate client to use web service
Client
Server
2
b. Authenticate users to use client
1
Local authentication
Figure 22: Auhtentication flow

In the case of this project, the a step is performed first, after which the users are authenticated via the web service, skipping the local authentication step completely. An outside developer consuming this service would most likely use their own user database and therefore first authenticate the user and later the client to be used with the server.

6 Development Review

6.1 Requirements and Test Results

The end product was finalized in October of 2009, slightly late from the proposed schedule. On the other hand, the product contains more features than were initially planned. Also, since the client had no actual environment where the application would have been implenented in, the delay did not have any negative impact on the project. Many features were invented and the existing ones modified while constructing the system, but none of the initial features were discarded completely.

Because the product could not be tested in a real environment, no long run statistic gathering or even any experience queries from an end user. It would be bold to suggest the application is completely ready for a company to use, but it is definitely ready for presenting the reserved call queue idea or testing it in an actual environment. Some conclusions must be drawn from the initial requirement fullfillments and the somewhat modest test case results.

Met Requirements

The most important features, such as the presentation of the call queue, answering and handling the calls were successfully implemented. Many features were discovered and implemented during the development, after the planning process. The end product ended up having several features additional to the initial proposal for the presentational demo of this kind of a system.

The administrative pages contain statistics that seemed to be essential at the time of the application development and discussing the requirements. The system records more information that is currently displayed at the administrative view, so more usable information can be provided if necessary. The final collection of this data should be defined after the system was in use for a while and the actual users could provide their opinions on the required information. The statistical data can be provided in two different ways. Either by providing the precalculated data directly from the service or by giving the raw call records, from which the consuming client can calculate the statistics it needs. Both of these options would likely be welcomed to the application in the future.

Testing

The testing was done according to the testing steps described in Pfleeger & Atlee's Software Engineering: Theory and Practice book.. The requirements used in the test process are those generated in the beginning of the project and some of the tests are pure inspection of the application's performance in it's current state and context. [16, 371-371]

These steps have been evaluated and described in the following paragraphs. To begin, the application can be divided into separate parts to achieve the modular unit testing. In terms of the client application's different functions, such as calling, answering, authentication, statistics presentation. The server application can also be considered in terms of call case handling and user handling. None of the separate units have any essential problems that would make them fail. Integration testing makes sure the units work together and data flow is error free and fault tolerant. This is testing also passes, except that the fault handling in data transfer errors is not completely covered. In the event of a communication error, the application does not always correctly inform the user of the current situation. Otherwise, the units work together as expected.

Function testing covers the technical requirements. As described in the beginning of this chapter, they have almost all been implemented. Only features such as SMS remain unimplemented, but only on the server side. In this, the client side application passes. The client side application has some problems in the extra featuresdealing with the less importatn real-time aspects of the application. These problems are minor, and do not prevent the application from being validated and moving to the next tests. The performance test is troublesome for this project, as it covers rest of the software issues, the most important of them being the testing in the final environment. In addition to the obvious problems, the inability to test the program over a real network does not relieve the application from it's current problem of cell phone timeouts due to NAT (Network Address Translation) in the testing environment.

The steps so far have covered the developer's understanding of the application requirement's and results. The acceptance test covers the customer's expectations and evaluations. Those are done in this case with the use of the original use cases and some external opinions of the user interface. The use cases defined in chapter 3 are all met, meeting the acceptance test in some areas. User interface seems to be coherent, even though instructions on different screens and how they're related to each others are first slightly unclear. The information on the call case boxes is somewhat excessive, and some effort should be invested in clarifying them. Partly with this test and especially with the final installation test, where the application should be set up in the final use environment, the results are inconclusive because of the lack of a proper client.

To test the software source code on programming level, some unit testing was performed on the classes used in the server application. A test is done on an application by implementing it to one, as small as possible piece of code at a time. This piece of code is generally a method in object oriented programming. An example of the testing is shown in appendix 2, where two tests are performed. One tests the relatively simple constructor of the CallCase class, and the other tests it's function used for retrieveing case specifics from the database. The relations between the class and source code can be derived from the class diagram, and the results of the test can be seen in the output picture.

6.2 Ecountered Challenges

Some initially required features were discarded from the application, because they could not be implemented in the testing environment. Since the testing environment was a private network and there was no actual outgoing phone line, all testing was done with VoIP (Voice-over IP) software phones and limited amount of those. This created some problems for the call retransferring and especially, the SMS (Short Message Service) text message reply function was forgotten. The program itself has the possibility of defining a text message message and a receiver, but the message cannot be sent because of the above mentioned reasons.

The application has no maximum limit of cases in the queue or concurrent callers online. This feature has not been implmented since there is no knowledge about how the system would behave under heavy traffic on either the Asterisk system or the web service level. The fact that a professional should automatically be made unavailable when making a call has not been implement, as it would be difficult to define when that call is ended or if the call was succesful at all. This was forgotten mainly because the professional can also do it manually.

We encountered some problems in defining the call logic in such cases where the customer already has a pending case in the system. In this situation, it is difficult to allow the customers control their existing cases with just the use of a phone. Because a phone with it's mere dial buttons is too simple of a tool to handle complex actions, there is a possibility for the users to view and delete their cases via their personal profiles in the website.

The Internet browsers interact with a server by adding the variables of the given page to the request when contacting the server. With this in mind, there arises a problem when a user has performed an action and chooses to press the refresh button on the browser. This resends the action command to the server and the server performs or tries to perform the same action again. There are ways to cirumvent this problem, but those have not been implemented. The current solution is to instruct the user not to use the browsers refresh button, but to rely on the automation and navigation bar of the application.

One of the most suprising problem was the audio message playback recorded from the customer. The Asterisk system records an audio file and saves it in waveform format. The most obvious playback solution was instantly that of using a lightweight, open source playback program running in the Flash player. This problem was solved with HTML 5, an approach that only requires a browser update. Providing that the user is reluctant to change or upgrade the browser, there is still a chance to listen to the audio with an external player.

6.3 Usefulness of Selected Software Architecture

The approach of using SOA and MVC in this project proved to be both challenging and rewarding in the end. MVC affected the development and maintenance of the application from the beginning, whereas service orientation was more helpful in providing ideas in personal level and forcing to consider the software development from a much larger aspect.

Implementing MVC was more straight forward and the benefits soon became evident, as the structure of the application allowed development of different parts separately. I could work on web service messages one day, and on the user interfaces on another day.

It strips the developer from the need of concurrently thinking about data processing and presenting. As the number of features in the application grew, the need to distinct them became more obvious, and the MVC proved to be the right structure for the application. The only, almost insignificant drawback, of the MVC is that in the beginning, it requires more effort to create the structure and data handling for it.

The usefulness of SOA has not proved to be so prominent. Its development required a lot of work, and the most amount of research of the individual topics. To convert the call data into general messages, the PHP objects needed to be deconstructed into simple values to be transferred with the SOAP messages. This, in a sense, doubled the amount of code, since the data needed to be reconstructed again to be displayed in the user interface. On the other hand, the fact that SOA allows the construction of separate, programming language independent client applications, the further development of those would instead be easier.

The actual benefits of SOA started to emerge in the later stages of the development. Once the the server application was completed and the service running, testing and development on the client application became considerably easier than in a regular testing environment. Considering the Asterisk developer needs a client application to test the call logic, he can use a stable but unfinished version of the client, while the user interface developer can concurrently work on an unstable version of it. Both of their client versions can consume the service and receive the same data from it simultaneously. This provides an interesting and somewhat unusual approach to software development.

6.4 Thoughts on Development Approach

Retrospectively, I believe I should've considered more on using a framework in the development of the application. A proper framework would have provided the MVC structure, as well as functioning and tested modules for web services security. The benefits of a ready made structure would've provided faster development in the beginning stages of the application. The modules would have provided me with proven methods to implement some of the low level, yet important features. A framework, once mastered, would have speeded the development and probably incresed the reliability of the application.

Constructing the application personally from scratch does provide more control over it and better understanding of the core technologies involved. It generally also means that the source code does not contain any unnecessary libraries or leftover code from a framework, thus making the application prospectively faster.

The use cases and user interface were derived from the initial customer's demands and a few meetings with the thesis instructors later on. These provided enough information about the needed features and the data to be displayed. Some more concrete design and consultation on the user interface might have been appropriate, since the current one is based on a few people's opinions and was not tested so much with the usability in mind.

6.5 Future of the Used Technologies

The future of the used development approach is important to be inspected, since it adds to the total amount of time required by the process. The added value of service orientation could not be proved in this project, so it is important to at least investigate if the selected architecture has any problems looming in the near future. The future is not, of course, documented and this chapter is naturally mere speculation and personal thoughts derived from the project

There's an expectation of growing demand for VoIP solutions as especially small businesses try to save expenses. As VoIP works over an Internet connection, the businesses can incorporate the telephone costs into their already existing broadband expenses, with little effort and small investment. The biggest savings come from the long distance calls, and it therefore depends on the business, if the real profit surpasses the actual investment in required tools and updates. [17, 4-5] The telecommunications industry still relies basically on the century old technology and is keen on keeping their systems closed and proprietary. They are slow to adapt to an open software phone industry, giving Asterisk a chance to make a difference. [1, 321] In any case, the growing amount of traditional broadband and mobile broadband connections will eventually generate customer interest for and knowledge of VoIP technology.

There are a few proposed development paths for SOA. One of which is suggests that integration with Web 2.0 would provide provide social and creative ways to provide services because instead of competing, they can contemplate for each other [18]. This is a rather interesting approach, but faces a problem of reasonably integrating these two, completely different technologies and ideas. Some believe that SOA is dividing into more specialized areas, such as SaaS(Software as a Service), BPM and cloud computing, stating that SOA provides an infrastructure for providing software or hardware rental services instead of huge investments. [20]

In the case of the specific technologies used in this project, especially UDDI seems to be gradually forgotten, mostly due to its complexity. This approach might be replaced by RESTful governance, simply because it is simpler and easier to use. This same trend might be affecting to SOAP technology as well, as the industries are keen to adopt the simplest of the available technologies, if otherwise as capable. REST is simpler, but lacks some of the advanced capablities of SOAP. [19, 1, 6]

Regardless of the used technology or terms, I believe service orientation will continue to spread and grow, especially in terms of large scale systems, where abstraction and modularity is of high value. Public services might not grow so quickly as even consuming, let alone providing, services remain difficult for average developer to handle.

7 Conclusion

Considering the application and its features, project was both succesful and personally challenging enough, mostly due to the service orientation. I feel the development approach with MVC sturcture was the right one, even though I might have even gone a bit further by using a proper framework. The service orientated approach was initially time consuming and frustrating, but began to display its benefits in the end of the development process. The actual, post-development benefits remain theoretical.

I thus remain skeptical if service orientation is a beneficial approach for exactly this kind of application, but am sure it is at least value adding. I believe the software development approach and added complexity required by the service orientation made me consider the whole project on a larger scope. I see this as a valuable experience and have only positive feeling about it.

The evaluation of met requirements and test cases provides an idea of the end result, but does not compare to any real world practical experience. As the future of the application remains uncertain, I suspect the VoIP, PBX and SOA technologies and ideologies will grow and become more widely used in the future.

References

1 Van Meggelen J, Madsen L, Smith J. Asterisk : the future of telephony, 2nd ed. Sebastopol, CA: O'Reilly; 2007.

2 Erl T. SOA: Priciples of service design. Boston, MA: Prentice Hall; 2008.

3 Cerami E. Web services essentials: distributed applications with XML-RPC, SOAP, UDDI & WSDL. Sebastopol, CA: O'Reilly; 2002.

4 Sperberg-McQueen CM. Web services and the W3C [online]. Canberra, Australia: W3C Internet Consortium. 21 August 2003.

URL: http://www.w3c.org.au/presentations/2003-08-21-web-services-interop/msm-ws.html. Accessed 1 November 2009.

5 Barry DK. Prior service-oriented architecture specifications [online]. Atlanta, GA: Barry & Associates, Inc. 2000 – 2009.

URL: http://www.service-architecture.com/web-services/articles/prior_service-oriented_architecture_specifications.html. Accessed 1 November 2009.

6 Coyle FP. XML, web services, and the data revolution. Indianapolis, IN: Addison-Wesley Professional; 2002.

7 Hartman B, Flinn DJ, Beznosov K, Kawamoto S. Mastering web services security. Indianapolis, IN: Wiley Publishing; 2003.

8 Booth D. Liu CK. WSDL Primer [online]. MIT, Boston, MA: W3C Internet Consortium; June 2007.

URL: http://www.w3.org/TR/wsdl20-primer. Accessed 1 November 2009.

9 O'Neill M. Web services security. Berkeley, CA: McGraw-Hill/Osborne; 2003.

10 Blake MB. Decomposing composition: service-oriented sotware engineers. IEE Software magazine 2009;6(24):68-77. IEE Computer Society Publishing.

11 Hurwitz J, Bloor R, Baroudi C, Kaufman M. Service oriented architecture for dummies. Hoboken, NJ: Wiley Publishing; 2007.

12 Gamma E, Helm R, Johnson R, Vlissides J. Design Pattern: Elements of reusable object-oriented software. Indianapolis, IN: Addison-Wesley Professional; 1995.

13 Constantine LL, Lockwood LAD. Software for use: a practical guide to the models and methods of usage-centereddesign. Reading, MA: Addison Wesley Longman, Inc; 1999.

14 Darie C, Brinzarea B, Chereches-Tosa F, Bucica M. AJAX and PHP: Building responsive web applications. Birmingham, UK: Packt publishing; 2006.

15 Brian Fling: Convert your site to mobile. .NET magazine 2008;8(178):42-47. Future Publishing.

16 Pfleeger SL, Atlee JM. Software engineering: theory and practice. Upper Saddle River, NJ: Pearson Education; 2006.

17 Wilson C. Recession and VoIP: good and bad news. Telephony 2009;1:4-5. Penton Media Publishing

18 Dutta A. The future of SOA - A service-based delivery model with Web 2.0 capabilities [online]. Armonk, NY: IBM DeveloperWorks library, October 2006.

URL: http://www.ibm.com/developerworks/rational/library/oct06/dutta/. Accessed 1 November 2009.

19 Juneja G, Dournaee B, Natoli J, Birkel S: SOA and future trends [online]. New York, NY: TechWeb, CMP Media LLC; 2009.

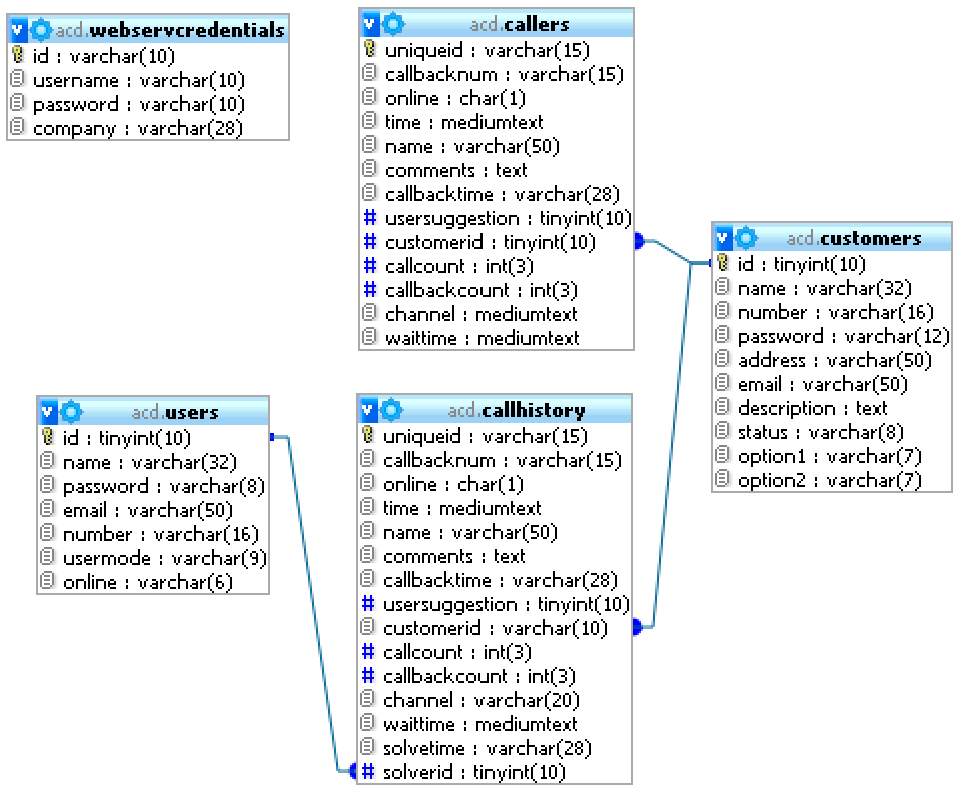
URL: http://www.ddj.com/architect/208402614. Accessed 1 November 2009.

20 Mortleman J. Service-oriented architecture in and beyond the credit crunch [online]. Cambridge, UK: Computer Weekly, Reed Business Information Ltd, March 2009.

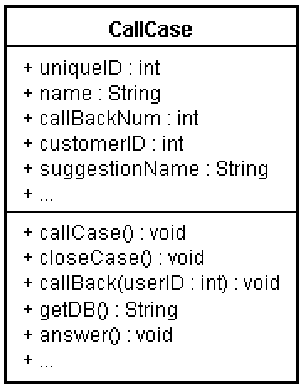
URL: http://www.computerweekly.com/Articles/2009/03/17/235299/service-oriented-architecture-in-and-beyond-the-credit.htm. Accessed 1 November 2009.

Appendices

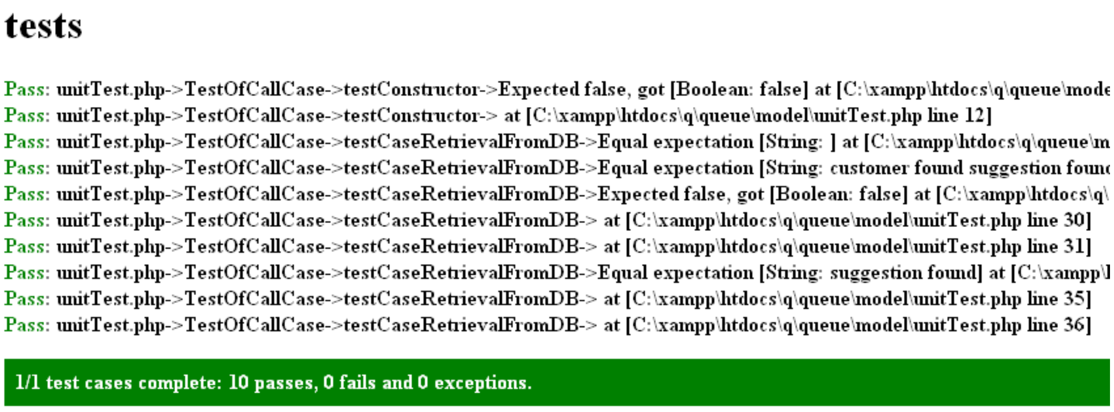
1 Complete Database Table Structure



2 Unit Testing for the CallCase Class



```
class TestOfCallCase extends UnitTestCase
{
    function testConstructor()
    {
        $cc = new CallCase('12345');
        $this->assertFalse($cc->uniqueID == null);                      // constructor does not fail
        $this->assertTrue($cc->uniqueID == '12345');                    // constructor set id
        //creation of case with any id is ok to allow dummy case object creation
    }

    function testCaseRetrievalFromDB()                   //id's used here are stored in the database
    {
        $ccImaginary = new CallCase('12345');            //this id does not exist, uniqueID not in db
        $ccRealCustomer = new CallCase('125309621439'); //actual callcase, caller is a known customer
        $ccRealUnknown = new CallCase('125309002415');  //actual callcase, caller is unknown

        //empty string expected as a return, since no data available
        $this->assertEqual($ccImaginary->getDB(), '');

        //from this case: name and suggestion is expected
        $this->assertEqual($ccRealCustomer->getDB(), 'customer found suggestion found');
        $this->assertFalse($ccRealCustomer->name == '');                    // name cannot be empty,
        $this->assertTrue($ccRealCustomer->name == 'N95 Silver');           // it is defined
        $this->assertTrue($ccRealCustomer->suggestionName == 'pro one');    // suggestion found

        //from this case: only suggestion is expected
        $this->assertEqual($ccRealUnknown->getDB(), ' suggestion found');   // return string
        $this->assertTrue($ccRealUnknown->name == '');                      // no name available
        $this->assertTrue($ccRealUnknown->suggestionName == 'pro one');     // suggestion found

    }
}
```



```
tests

Pass: unitTest.php->TestOfCallCase->testConstructor->Expected false, got [Boolean: false] at [C:\xampp\htdocs\q\queue\mode
Pass: unitTest.php->TestOfCallCase->testConstructor-> at [C:\xampp\htdocs\q\queue\model\unitTest.php line 12]
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB->Equal expectation [String: ] at [C:\xampp\htdocs\q\queue\m
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB->Equal expectation [String: customer found suggestion founc
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB->Expected false, got [Boolean: false] at [C:\xampp\htdocs\q\
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB-> at [C:\xampp\htdocs\q\queue\model\unitTest.php line 30]
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB-> at [C:\xampp\htdocs\q\queue\model\unitTest.php line 31]
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB->Equal expectation [String: suggestion found] at [C:\xampp\
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB-> at [C:\xampp\htdocs\q\queue\model\unitTest.php line 35]
Pass: unitTest.php->TestOfCallCase->testCaseRetrievalFromDB-> at [C:\xampp\htdocs\q\queue\model\unitTest.php line 36]

1/1 test cases complete: 10 passes, 0 fails and 0 exceptions.
```