Technical University of Denmark

Emil Lysgaard Hansen, s082714
Søren Fuhr, s082724

A Framework for Hosting Context-Aware Web Services

An Analysis of Available Solution Models

Bachelor's Thesis, June 2011

IMM-B.Sc.-2011-18

DTU Informatics
Department of Informatics and Mathematical Modelling

Emil Lysgaard Hansen, s082714
Søren Fuhr, s082724

A Framework for Hosting Context-Aware Web Services

An Analysis of Available Solution Models

Bachelor's Thesis, June 2011

IMM-B.Sc.-2011-18

A Framework for Hosting Context-Aware Web Services, An Analysis of Available Solution Models

This report was prepared by
Emil Lysgaard Hansen, s082714
Søren Fuhr, s082724

Supervisors
Jakob Eg Larsen
Sune Lehmann Jørgensen

Release date:	June 27th, 2011
Category:	1 (public)
Edition:	First
Comments:	This report is part of the requirements to achieve the Bachelor of Science in Engineering (B.Sc.Eng.) at the Technical University of Denmark. This report represents 15 ECTS points.

Department of Informatics and Mathematical Modelling
Mobile Informatics Lab (Milab)
Technical University of Denmark
building 321
DK-2800 Kgs. Lyngby
Denmark

www.milab.imm.dtu.dk/
Tel: +45 45 25 33 51
Fax: +45 45 88 26 73
E-mail: milab@imm.dtu.dk

Abstract

The industry of communication technology has shown great advancement in the last decade and has resulted in devices capable of performing complex computational tasks along with collecting context specific data about the users. This development, coupled with the growth of social networks, give way for a novel interest in advanced analysis of social groups.

The purpose of this thesis is to study the requirements needed to construct a system for the Mobile Interactions Lab's Context-Awareness research programme, a part of the Technical University of Denmark, that enables easy and flexible development of scalable context aware mobile applications and services for a campus-wide deployment.

We carried out a feasibility analysis of available technical solution models, based on a set of high level requirements, in order devise a proper system design. This design consists of a three-tier Web service system architecture hosted on Amazon Elastic Compute Cloud. Additionally, a proof-of-concept prototype was created to test the utilization of selected component functionality in regards to the initial requirements.

The results of this test indicated a clear potential of the suggested system design for rapid development of diverse Web services. However, it was found that the research outcome may not be directly applicable for a full scale deployment. Therefore, further research on the product specification is necessary.

Resumé

Der har inden for det seneste årti været stor fremgang inden for kommunikationsteknologien, hvilket har resulteret i apparater, som kan udføre komplekse beregningsmæssige opgaver, samt indsamle af kontekst bestemte data om brugerne. Denne udvikling, kombineret med væksten af sociale netværk, giver fornyet interesse i avanceret analyse af sociale grupper.

Formålet med denne afhandling er at undersøge de krav, der er nødvendige for at kunne konstruere et system for Mobile Interactions Labs Context-Awareness forskningsprogram, en del af Danmarks Tekniske Universitet (DTU), som muliggører nem og fleksibel udvikling af skalérbare kontekst-bestemte mobil applikationer og tjenester for udrulning på hele DTUs campusområde.

Vi har gennemført en forundersøgelse af tilgængelige tekniske løsningsmodeller, baseret på et sæt af høj-niveau krav med henblik på, at udarbejde et tilfredsstillende løsningsdesign. Dette design består af en tredelt Web service systemarkitektur, udbudt på Amazons Elastic Compute Cloud. Endvidere blev en proof-of-concept prototype skabt for, at undersøge funktionaliteten af de foreslået komponenter i forhold til de oprindeligt opstillede krav.

Resultaterne af denne test angiver, at der findes et klart potentiale af det foreslåede system design, til hurtig udvikling af forskelligartede Web tjenester. Dog fandt vi ud af, at dette resultat ikke er direkte anvendeligt for en fuldstændig implementation. Derfor er yderligere studier af produkt specifikationen nødvendig.

Preface

This thesis was prepared at Department of Informatics and Mathematical Modelling, Technical University of Denmark in partial fulfillment of the requirements for acquiring the B.Sc. degree in IT and Communication Technology.

The work on this thesis was done from March 1st, 2011 to June 27th, 2011. The workload corresponds to 15 ECTS points. The thesis supervisors are Jakob Eg Larsen and Sune Lehmann Jørgensen, Department of Informatics and Mathematical Modelling, Technical University of Denmark.

We would to thank Jakob Eg Larsen and Sune Lehmann Jørgensen for giving us the oppurtinity to work on this project and for their supervision. In particular, we would thank Toke Jansen Hansen for sharing his extensive knowledge and experierence on this topic, and for helping us with technical support during the project work.

Furthermore, we would thank Wanlika Kaewkamchand for her support on optimizing the graphic illustrations presented in the thesis, and Hanne Lysgaard Hansen for helping us with the editoral process. Lastly, we would like to thank our family and friends for showing us great support and interests, during the entire project period.

Kongens Lyngby, June 2011

Emil Lysgaard Hansen and Søren Fuhr

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The industry of communication technology has shown great advancement in the last decade. The development of faster network infrastructures, with higher capacities and wider dispersal of use, coupled with the tremendous growth in functionality and distribution of advanced mobile devices has opened a world of opportunities and changed the way people communicate with one another.

The technical development has resulted in devices capable of performing complex computational tasks along with collecting context specific data about the users such as GPS location, Bluetooth connections and nearby network access points. This progress has made the mobile device the power central of many people, merging together almost every aspect of their life from personal relations management, applications, entertainment, and in the near future, personal financing through *Near Field Communication (NFC)*[1], all together on one single mobile unit.

We think it is interesting to look into how empirical data from mobile devices can be used to study social behavior among individuals and groups; specifically, the ability to couple the enormous quantity of contextual data generated in the network by the devices with the personal information and social identity of the users provided by social network services such as Facebook[2], Twitter[3] and LinkedIn[4].

The core functionalities of these services are used to establish, maintain, and interact with personal and professional networks. Furthermore, a growing trend among these social services is to provide context-aware applications for both web- and mobile clients, consisting of distinctive user input, technical data, and sensor readings from affiliated devices that are stored and analysed on large online server clusters, or *Clouds*, such as *Amazon Elastic Compute Cloud (EC2)*[5].

[1] http://en.wikipedia.org/wiki/Near_field_communication
[2] http://www.facebook.com
[3] http://www.twitter.com
[4] http://www.linkedin.com
[5] http://aws.amazon.com/ec2

Recognizing this relatively new paradigm created by context-aware mobile systems and advancement in social networking, also known as Mobile Social Networks, we are interested in investigating the possibilities of utilizing evaluations of behavioral patterns and social relations of users in order to provide even more intelligent social service products.

In particular, we want to construct a system for the Milab DTUs Context-Awareness research programme, that acts as a foundation for advanced social group analysis; from the task of identifying already established social groups to being able to facilitate dynamic creation of "ad-hoc" groups, based on the available contextual information provided by existing technologies and services.

However, in order to create and deploy such system, we must first acquire a thorough understanding of the problem at hand and look at what is needed to realize such objective.

1.2 Related Work

In the last decade, interest in the research field of context-awareness has increased alongside the growth in availability of mobile devices in the general public. A number of people have been working on the creation of frameworks and platforms for context-aware applications. The articles presented in the following serve as a basis and inspiration for the analytical work presented in this thesis.

Larsen and Jensen (25) illustrate the possibilities of rapid prototype development of mobile applications, based on their framework "Mobile Context Toolbox" for S60 mobile phones. The framework utilizes device sensors, such as GPS, accelerometers, and proximity sensors along application data to derive user context. Two prototype applications were developed for initial experimentation with real life usage of the platform.

Daniel and Matera (13) propose a new way of building context-aware web applications. They discuss current approach to context-awareness and describe MixUp, a component based framework model for easy-to-use integration, or "mash-ups", of web services in application development. The work of Daniel and Matera is an extension to a prior article by Ceri, Daniel et al. (30), in which conceptual modelling of new concept-aware frameworks based on WebModeling Language (webML)[6] is discussed. With origins in the studies of context-awareness, several people have worked on the establishment of new and integration of current social networks on mobile devices with context-aware functionality.

In *Context-Aware Middleware for Anytime, Anywhere Social Networks*, Bottazzi et al. (8) investigate the creation of ad-hoc Mobile Social Networks (MoSoNet)[7] and discuss the differences between Internet-based- and mobile-based social networking. A location-centric framework, *SAMOA*, is proposed and makes use of semantic modelling, allowing users to create proximity based roaming social networks. In addition, a concept prototype of a viral-marketing social network for a shopping centre, implementing *SAMOA*, is presented.

[6] http://www.webml.org
[7] http://en.wikipedia.org/wiki/Mobile_social_network

Similarly, Beach et al. (3) created WhozThat?, a system combining social network services with mobile phones to form a MoSoNet, that can notify users about the identity of nearby individuals and provide rich social content based on context awareness. In particular, a context-aware music player, that automatically picks out songs based on client preferences, for use in restaurants and bars, is shown. The system is based on an infrastructure that draws on wireless technologies e.g. Bluetooth and Wifi along with sharing social network IDs through a customized protocol. Moreover, security and privacy issues in MoSoNets are also discussed.

Eagle et al. (14) compares the reliabilty using of observational sensor data from mobile devices versus self-reported user data for social network analysis. According to experimental results, they demonstrate that it is possible to deduce 95% of friendships among test subjects using only observational data for statistical analysis.

Adding to this, in 2010 Beach, Gatrell et al. (4) published *Fusing Mobile, Sensor, and Social Data To Fully Enable Context-Aware Computing* in which they present *SocialFusion*, a framework that seeks to combine context data from mobile phones, network data, and social network data from users. They discuss the major challenges and possibilities of creating context-aware MoSoNets, including how to organize and store big sets of diverse data streams alongside extraction and analysis of this data. Several examples of use are described involving prediction of social groups and advanced data mining of context allowing for tailored user recommendations. In addition, security and privacy issues are considered and a new approach to k-anonymity algorithm design is presented.

The academic focus on context-awareness and MoSoNets also sparked interest in studying the use of Cloud Computing for scientific research and for commercial use in Web services and business IT infrastructures.

Armbrust el al. (2) from Berkely look into the development of cloud computing and give a detailed overview of the various cloud services available today, along with pros and cons of each solution. Furthermore, obstacles and opportunities of cloud computing are discussed and compared to traditional server solutions, in terms of both cost and performance with particular focus on Amazon EC2.

Kossmann et al. (23) published a paper on Cloud computing architures and did a detailed comparison on performance and cost of the three major Cloud providers; Amazon, Google, and Microsoft.

Additionally, a number of related studies on cost and performance benchmarking of Amazon EC2 for scientific applications have been conducted, including the work of Berriman et al. (6), Akioka and Muraoka (1), and Juve et al. (22), all presenting the potentials of utilizing cloud computing for scientific use.

1.3 Thesis Objective

Considering our motivation for partaking in the Milab DTU Context-Awareness research programme and the related work presented in Section 1.2, we have come up with a set of learning- and research objectives in respect to our long-term target of providing context-aware analysis software of social groups.

Taking into account the scope and time frame of this Bachelor's project, we will address the following issues in the thesis:

- Outline the required capabilities of a system to support context-aware applications and features.
- Describe currently available software- and network technologies for developing web compliant systems.
- Conduct a feasibility analysis of the given solutions in order to present a viable realization of the system requirements.
- Design a high level system model in adherence to the analytic results.
- Show a proof-of-concept prototype implementing a custom API for interaction with the designed software system.
- Review our findings and discuss future work in order to achieve our long-term targets.

1.4 Thesis Outline

We use two types of references throughout this thesis. Footnotes are used as references to websites and non-academic documents. Citations are used to reference books, official documentation of standards, and academic articles.

Chapter 2 Introduces technical concepts used in thesis, describes the requirements for our system implementation, and looks at how to realize the devised requirements by presenting a feasiblity analysis of available solutions.

Chapter 3 Gives a detailed description of the design of our system infrastructure and provides an in-depth presentation of the Web service paradigm.

Chapter 4 Discusses the implementation specific details of the developed software components of our proof-of-concept Web service.

Chapter 5 Reviews the software solution presented in Chapter 3 and outlines considerations of future work.

Chapter 6 Summarizes and concludes on the work presented in this thesis.

Chapter 2

Analysis

In this chapter we will conduct a thorough analysis of system models and back-end infrastructures available today, in order to find the most fitting solution model for the purpose of this project. However, in order to conclude on any specific solution, we must first come up with a set of requirements necessary in order for our system to fulfill the goals of the project.

2.1 Definitions

This section defines and describes some of the main terminologies and concepts used in the analysis.

2.1.1 Cloud Computing

Many interpretations of the term *Cloud Computing* exist and it is a topic of continuous discussion. However, we will use the following definition, inspired by Armbrust et al. (2):

"*Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS) ... The datacenter hardware and software is what we will call a Cloud.*"

Generally, we talk about two main types of Clouds, based on availability, although other variations exist. A *Private Cloud* refer to an internal datacenter of a corporation or other organization running various SaaS functions, which is not available to the outside world. However, if a Cloud is made generally available for rent or usage via a pay-per-use scheme, it is referred to as a *Public Cloud*. We will not be covering Private Clouds in this thesis.

2.1.2 Client-Server Model

A Client-Server architecture is a structure that distributes and divides computational tasks between two or more processes on either a single- or, most often, several machines. As detailed further in *Database Programming with JDBC and Java* (28):

"*Any database application is a client/server application if it handles data storage and retrieval in the database process and data manipulation and presentation somewhere else. The server is the database engine that stores the data, and the client is the process that gets or creates the data. The idea behind the client/server architecture in a database application is to provide multiple users with access to the same data.*"

Several client-server architecture variations exist, as will be discussed in Section 2.3.1. A great example of a Client-server architecture is the *Web browser/Web server* and *email/mail server* paradigm.

2.1.3 Infrastructure-as-a-Service

Infrastructure as a Service (IaaS) is a provision model, a part of the Cloud Computing business notion, in which an organization outsources the equipment used to support operations, including storage, hardware, servers and networking components. The service provider owns the equipment and is responsible for housing, running and maintaining it, and in return the client typically pays on a per-use basis for usage of said equipment[8].

2.1.4 Platform-as-a-Service

Platform as a Service (PaaS) is a way to rent hardware, operating systems, storage and network capacity over the Internet, most typically as a Cloud Computing implementation[9]. It differs from IaaS, primarily, by allowing the customer to lease pre-configured computing platforms and solutions stacks, providing all of the facilities required to support the complete life cycle of building and delivering web applications and services entirely available from the Internet[10].

2.1.5 Software-as-a-Service

Software as a Service (SaaS) is a software distribution model in which applications and their related data are hosted centrally, e.g. Clouds, by the service provider and are accessed by users, most often, through the Internet via thin clients such as web browsers. An example integration of the SaaS model is "Salesforce.com"[11] that provides business software, such as CRM[12], online via Cloud hosting[13].

[8] http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-a-Service-IaaS
[9] http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS
[10] http://http://en.wikipedia.org/wiki/Platform_as_a_service/
[11] http://www.salesforce.com/
[12] Customer Relationship Management
[13] http://en.wikipedia.org/wiki/Software_as_a_service

2.2 Requirements

Based on the goal of this project and several meetings with the other groups involved in the context-awareness research programme at Milab, we have devised a set of preliminary requirements for the back-end system. It is worth noting that all requirements are on a high level of abstraction, due to the complexity and uncertainty of future development needs. Effectively, these requirements along with our finished server design will act as a basic infrastructure framework, that can be extended with more functionality later on if required. The requirements are summarized as follows:

1. **Scalability and Performance** The system must be able to allocate resources dynamically, such that performance will not be a limiting factor, nor will it be unexploited.
2. **Extendability** The system must be susceptible to changes in which features are deployed, making it easy to increase the functionality of the system applications.
3. **Platform independency** The system must be able to interoperate with several types of client platforms, such as iPhone iOS, Android OS, and Symbian.
4. **Multi-language support** The system must be compatible with a wide range of programmatic languages and frameworks, e.g. mathematical tools such as Matlab, or a languages such as Python in order to ease the process and lower the transaction times for new prototyping- and proof-of-concept projects for students participating in the research programme.
5. **Cost** Deployment- and operational costs must be kept at a minimum and should flexible in accordance with the usage and scalability of the system. This is to ensure a lower risk level during the start-up of the research programme.

We acknowledge the importance of security when choosing a back-end infrastructure. Since the system will be handling personal and sensitive information, steps must be taken to ensure this data is out of reach for unauthorized parties. Furthermore, the system must also have some physical security, such as data back-ups, as well as network security making the infrastructure resistant against cracking attempts such as DDoS and Man-in-the-Middle attacks.

However, due to this preliminary status of the research programme and parallel projects, the topic is currently out of scope of this thesis, but will be discussed in brief in Section 2.3.1.2 and Section 5.3.

2.3 Realization of Requirements

In order to come up with a satisfying solution to our target goal, we must first take a closer look at the available technologies, which can fulfill the before-mentioned requirements for our system. This section covers the large scale decisions regarding our choice of system architecture and hosting platforms. A detailed look at the incorporated system design is available in Chapter 3.

2.3.1 System Model

Today, two generic types of system architecture for client-server applications exists; the two-tier architecture and the three-tier architecture, along with variations of these models depending on specific implementations and needs of the developer. Both architectures have their advantages and disadvantages, as we will explore in this section. Please note that we are omitting details on the single-tier architecture, since it has no use in development of client-server applications.

2.3.1.1 Two-tier Architecture

First of all, we have the two-tier architecture, which is the simplest structural form of a client-server architecture. It is built of two components, the *presentation layer* and the *data layer*. The presentation layer, most often illustrated as *clients*, which connects directly to the data layer containing all system viable data stored in a database (28). This relationship is illustrated in Figure 2.1.

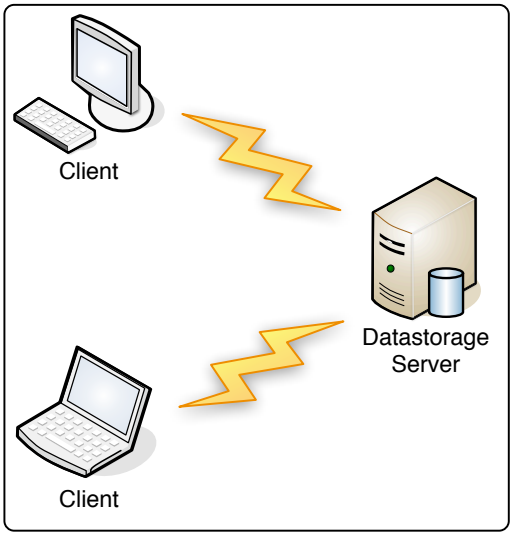

Figure 2.1: The two-tier client-server architecture.

The two-tier architecture is mainly used for small scale database systems and simple websites, since there is no need for an interleaving application layer. All actions made by the users are directly updated in the database. The model works very well with simple applications and non-scaling systems.

However, this system model has several limitations. As soon as one needs to implement more complex data processing, the two-tier architecture will fall short in the long run (28), because the client is directly tied to the data layer. Very often, what happens is that one will end up with what is called *a Fat Client*, in which all the processing is done on the client side, possibly implementing a lot of overhead features and functionalities unrelated to the task of the current user. Therefore, two-tier systems and fat clients are known to scale very badly, as data volumes and the userbase grow. For this reason, this system model will not be a feasible solution for this project.

2.3.1.2 Multi-tier Architecture

The multi-tier architecture, also called n-tier architecture, is built on the same principles as the two-tier system model but it has one core difference, isolation of dataprocessing. It is the preferred software architecture for modern Web applications (31). In its most basic form, known as the three-tier architecture, the direct connection between the presentation- and data layer has been replaced by a so-called *application layer*, as shown in Figure 2.2. The three-tier architecture can be expanded to a n-tier model by extending the number of servers and business logics on the various tiers. However, this requires a more complex implementation due to the need of load balancing and advanced data management.

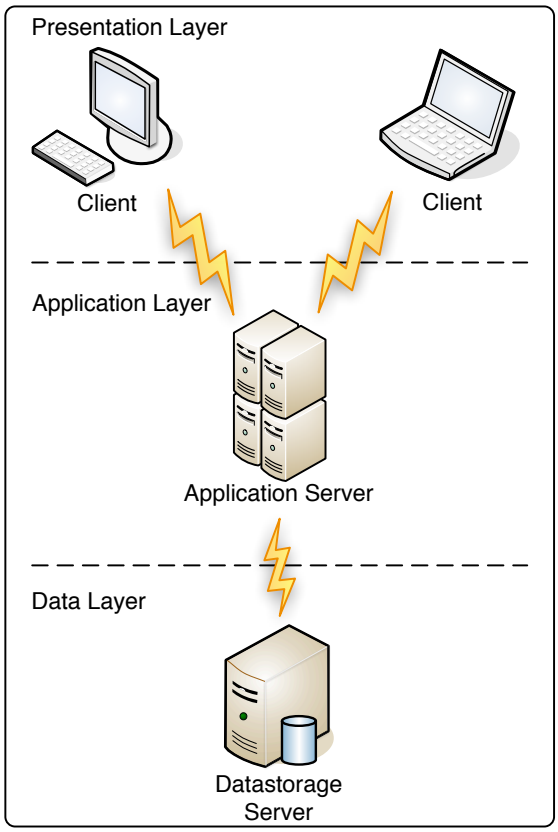

Figure 2.2: The three-tier client-server architecture.

The multi-tier design has several advantages. First, a system utilizing a multi-tier architecture is essentially built by smaller modules that can be upgraded or changed relatively easy, due to the system flexibility caused by separating the presentation- and data layer (31). In a multi-tier modelled system, the presentation layer, or client, can only communicate with the application server hosting the business logic. It has no way of communicating directly with the data layer, hence it does not care how the database is implemented, where it is located or if the database is distributed among a cluster of servers. Furthermore, this separation ensures a higher level of security and safety, since it is easier to control client access to the data layer, which may store sensitive information.

Secondly, the modularity ensures fairly straightforward scalability of the system on all tiers. For example, if a developer wants to extend his web application to a specific new operation system, he will only have to write a new piece of software for the presentation layer on the given platform, which supports the API of the application layer (28). Likewise, one can with reasonably low efforts extend the number of application servers to deal with increased data traffic by users and have a load balancer distribute the processing requests throughout the available servers. This enhanced management of one's system infrastructure is one of the key advantages of using a Cloud Computing service, such Amazon Web Services, for hosting multi-tier software systems, as discussed in Section 2.3.2.

Nevertheless, the flexibility of the multi-tier architecture comes with a disadvantage. It takes more work to plan and set up a system based on a multi-tier architecture. In addition, the increased complexity of integration and communication between the given components can make it more difficult and time-consuming to maintain.

2.3.2 Cloud Computing and Dedicated Servers

When deciding on a host solution for our system we have two options. First, one can buy, or rent, a dedicated server. This has the advantage of the developer being the sole user of the server, which allows for full control of hardware- and bandwidth resources and software configurations. This makes the system fully customizable and often very stable. Additionally, using a dedicated server makes the developer less reliant on third-party interventions, such as maintenance and data handling.

While being the sole owner of a dedicated server has clear benefits, there are several considerations to be made. First, one is faced with high upfront deployment costs when buying, or renting, a dedicated server. In extension of this, there is a relatively high level of fixed monthly expenses[14], such as electricity and maintenance, even if the server resources are not being utilized to its full extent. Secondly, owning only one dedicated server, makes it rather difficult to implement a scalable three-tier software architecture as we intend to do, due to the prominent cost of upgrading and extending the number of disposable servers.

The second option is to host the system in the emerging market of *Cloud Computing*. Analogous to the advantages of owning a dedicated server, Cloud Computing provides several strong points, though typically not equivalent to those of dedicated server hosting. While dedicated servers introduce a high level of overall control, Cloud Computing takes a different approach by offering pay-on-demand usage of computational resources. For example, a processing task taking 1000 hours on a single server can be done for the same cost in 1 hour on 1000 servers hosted in the Cloud, assumming that the programs can scale (2). There are no long-term commitments and very low deployment costs involved, which makes this hosting solution flexible in accordance with developer needs.

Furthermore, Cloud Computing offers several features such as automatic scaling of hardware resources, as data traffic increases or decreases, including scalable data storage and load balancing functionality.

[14] http://www.serverschool.com/dedicated-servers/how-much-does-a-dedicated-server-cost/

This functionality makes it straightforward to implement multi-tier system architectures on Cloud Computing host platforms, due to the design of these platforms themselves (23).

However, this flexible business model might be considered an achilles heel of Cloud Computing, as it can be difficult to predict the total long-term costs of using the pay-as-you-go scheme, due to interrelating service fees and the complexity of the Cloud Computing infrastructure. Moreover, one depends on the Cloud provider to ensure stable server uptime and reliable maintenance of physical hardware systems. Lastly, some Cloud Computing hosting services require the developer to implement proprietary APIs, which can cause data lock-in.

Recent studies of Cloud Computing, as put forward by e.g. Armbrust et al. (2), Berriman et al. (22), and Kossmann et al. (23) focus on Cloud Computing as an alternative solution to current system implementations. A general consensus regarding research results in the mentioned articles, is that Cloud Computing has several benefits over traditional server solutions. However, the benefits may vary, depending on the actual workload requirements of the developer and the current configuration. For example, if a business has large amounts of data stored in a data center, it might not be prudent and economically feasable to migrate to a Cloud hosting solution (2). We refer to the above mentioned articles for an in-depth cost-benefit analysis of the Cloud Computing paradigm.

On the other hand, we are developing a novel server system, hence we are not concerned with compatibility, costs of transferring large quantities of data from existing storage solutions, and other migration related issues.

Furthermore, due to the nature of this project and current academic interest in the Cloud Computing field, we believe it will be rewarding to pursue this solution model, considering the low entry-barriers of financial burdens and deployment.

2.3.3 Amazon Web Services

Amazon Web Services, or AWS, was released in 2003 as the first public Cloud Computing host service and it is therefore the most mature product on the market today. Amazon Web Services, in its core form, provides a range of *Infrastructure as a Service (IaaS)* solutions with an adaptable pay-as-you-go price model[15]. Amazon's main product is *Elastic Compute Cloud (EC2)*, which is a virtual computing environment that allows developers to launch a number of Virtual Machines, known as *Instances*, using either template images or custom configurations to fit requirements of the developer. This service is elastic, meaning that one can increase or decrease system capacity within minutes to comply with varying data traffic, and that this management of resources can be controlled automatically by the Cloud itself using features such as *Auto Scaling* and *Elastic Load Balancer*.

In conjunction to the elastic Virtual Machines, two types of database services are available. A traditional database solution, *Amazon Relational Database Service (Amazon RDS)* is offered. It is a Web service providing easy setup and deployment of Cloud based relational databases, currently supporting MySQL and Oracle.

[15] http://aws.amazon.com

Alternatively, one can make use of *Amazon SimpleDB*, which is a flexible and scalable non-relational database, optimized for high availability. In addition to database services, AWS offers scalable data storage solutions such as *Amazon Simple Storage Service (S3)* and *Amazon Elastic Block Store (EBS)*, which are off-instance storage volumes. These storage block volumes, equivalent to physical harddrives, can be mounted directly on EC2 instances where they can be used for storing server images and back-up of databases among other things.

As described, AWS provides a scalable, easy to deploy, IaaS product giving the developer full control over every aspect of his server system. Adding to this, AWS has a very flexible pay-as-you-go business model with a wide range of prices, depending on developer requirements. Currently, AWS also offers a one year free-usage tier to new customers[16].

Nevertheless, AWS has its drawbacks. First off, being a highly flexible IaaS product, means that all system maintenance- and operational responsibilites, such as software updates and problem solving, are now in the hands of the developer, neglecting the hardware maintenance done by Amazon. Second, the developer is solely in charge of integrating AWS services to comply with one's needs, whereas other Cloud Providers such as Google App Engine take care of that aspect for the clients. Lastly, the pricing model of AWS, despite being versatile, can make it hard to predict long-term operational costs as argumented for in Section 2.3.2.

However, to address this issue, Amazon provides an Excel spreadsheet for calculation the annual costs of running an EC2 instance cluster on their platform. A similar pricing calculator exists for Amazon RDS.

2.3.4 Google App Engine

Google App Engine (GAE), initially released in 2008, is a development and hosting platform provided by Google, and it is targeted exclusively for development and deployment of web applications. GAE is considered a *Platform as a Service (PaaS)* opposed to Amazon AWS, being IaaS, as mentioned in Section 2.3.3, giving developers access to a specialized SDK and development tools. Like AWS, GAE comes with a free-tier usage plan as well as an expanded pay-per-use pricing scheme. Furthermore, a budgeting tool is provided, making it easy to control and limit the costs of running one's Web Application[17].

Once an application is built, Google takes care of deploying the service on one of their cloud instances. Furthermore, GAE automatically scales the available resources for a given application, depending on the workload. Effectively, this means that Google takes care of all underlying levels of the system infrastructure and administration of these.

This abstraction makes it easy for developers to quickly build and test web applications at a low cost, due to the simple SDK interfacing between the developer and the server system. In addition, the GAE SDK offers several out-of-the-box services, such as Mail, XMPP and Images[18] available through APIs. Currently, only

[16] http://aws.amazon.com/free/
[17] http://code.google.com/intl/da/appengine/docs/billing.html
[18] http://code.google.com/intl/da/appengine/docs/java/apis.html

Python, Java, and to some extend JVM based languages are supported on GAE, though this might change in the future. Nevertheless, this choice of programming languages for the platform makes it relatively quick to write and reuse code. An example of a Python application[19] on GAE is shown in Figure 2.3.

```
#Script code for HelloWorld.py

print 'Content-Type: text/plain'
print ''
print 'Hello, world!'

_____________________________
#Deployment Commands

application: helloworld
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
script: helloworld.py
```

Figure 2.3: Implementation of a simple *Hello World!* Python Web Application.

In terms of storing data, Google makes use of its own, non-relational, scalable storage database called option Datastore, similar to Amazon SimpleDB. It utilizes a custom simplified SQL dialect called GQL and supports both High Replication Datastore and Master/Slave Replication[20].

However, the ease of use comes with a cost. Unlike Amazon AWS, Google App Engine is restricted in numerous ways. First of all, you are effectively locked to Google's proprietary platform, APIs, and storage facilities, making it difficult and time consuming to migrate to other services later, if needed. Recently, several third-party APIs have surfaced, trying to ease the platform mobility of GAE, such as *TyphoonAE*[21].

Secondly, GAE has several constraints to ensure performance and scalabilty of its platform. One of the main considerations is that applications must be request-response based. If the request handler exceeds the maximum time restriction while generating a response, usually 30 seconds, it will be terminated. This precaution is implemented to ensure low CPU times and seamless autoscaling on the GAE platform[22].

[19] http://onlamp.com/pub/a/onlamp/2008/05/20/getting-started-with-the-google-apps-engine.html
[20] http://code.google.com/intl/da/appengine/docs/python/datastore/overview.html
[21] http://code.google.com/p/typhoonae/
[22] http://code.google.com/intl/da/appengine/docs/python/runtime.html

2.3.5 Microsoft Windows Azure

The Windows Azure Platform, publicly available in 2010, is Microsoft's take on a Cloud Computing platform for creating and hosting Web Applications[23]. Therefore Windows Azure is categorized as PaaS, like Google App Engine. When comparing the three platforms, Windows Azure is considered in-between AWS and GAE, meaning that it incorporates some of the flexibility of AWS while still separating the developer from the low-level architecture, e.g. hardware configuration and Operating System, of the platform. Like GAE, it is worth noting the constraints of using this platform, especially regarding being locked to Microsoft's proprietary SDK.

Development of applications on Windows Azure is done using the .NET libraries and are compiled to the Common Language Runtime[24], which offers flexibility in terms of programming language utilization, making it possible to write applications in Visual Studio languages, Java, PHP and Ruby. The Windows Azure platform also provides a collection of SaaS functionality for easy access to existing Microsoft products along with direct integration in Visual Studio, including a GUI and tools for easy management and deployment of applications, unlike Amazon that relies on command-line interfacing. Furthermore, Windows Azure offers two types of datastorage; SQL Azure, which is Microsoft's own implementation of SQL server and the non-relational Azure Storage Service, homologous to Google DataStore and Amazon SimpleDB (2).

Microsoft Windows Azure offers two types of business models for clients. One can either use the Pay-as-you-go formula or sign up for a montly subscription plan with a minimum duration of six months. The Subscription plan has the advantage of offering, potentially big, discounts on usage, and customized subscription plans can be tailored through a provided pricing calculator[25].

2.4 Summary

This section summarizes the results of our analysis and proposes a solution model for the system design of the infrastructure for the Context-Aware research programme at Milab DTU.

In order to ensure extendability we choose to make use of a multi-tier software architecture, because this model facilitates flexible changes to the system by separating the presentation layer, business logic, and data layer. This allows us to quickly add new features on request, without affecting the already established functionality on the server side. Adding to this design philosophy, the Web service paradigm, described in Chapter 3, is an obvious candidate for a practical implementation of this software architecture type.

We have opted for Cloud Computing as hosting solution due to the proficiency in scalability and perfomance utilization of this technology. The Cloud Computing

[23] http://www.microsoft.com/windowsazure/
[24] http://msdn.microsoft.com/en-us/library/ddk909ch(v=vs.71).aspx
[25] http://www.microsoft.com/windowsazure/offers/

	Amazon Web Services	MS Windows Azure	Google App Engine
Architecture	• IaaS	• PaaS	• PaaS
Computation model (VM)	• x86 Instruction Set Architecture (ISA) via Xen VM	• Microsoft Common Language Runtime (CLR) VM; common intermediate form executed in managed environment	• Predefined application structure and framework; programmer-provided "handlers" written in Python, all persistent state stored in MegaStore (outside Python code)
	• Computation elasticity allows scalability, but developer must build the machinery, or third party VAR such as RightScale must provide it	• Machines are provisioned based on declarative descriptions (e.g. which "roles" can be replicated); automatic load balancing	• Automatic scaling up and down of computation and storage; network and server failover; all consistent with three-tier Web app structure
storage model	• Range of models from block store (EBS) to augmented key/blob store (SimpleDB) • Automatic scaling varies from no scaling or sharing (EBS) to fully automatic (SimpleDB, S3), depending on which model used • Consistency guarantees vary widely depending on which model used • APIs vary from standardized (EBS) to proprietary	• SQL Data Services (re- stricted view of SQL Server) • Azure storage service	• DataStore(BigTable)
Networking model	• Declarative specification of IP-level topology; internal placement details concealed • Security Groups enable restricting which nodes may communicate • Availability zones provide abstraction of independent network failure • Elastic IP addresses provide persistently routable network name	• Automatic based on programmer's declarative descriptions of app components (roles)	• Fixed topology to accommodate three-tier Web app structure • Scaling up and down is automatic and programmer- invisible

Table 2.1: Comparing features of Cloud Providers (2)

providers mentioned offers several automated, low cost, components for effortless management of system asssets, such as automatic scaling of Virtual Machines, depending on the computational resources needed. Also, as described in Section 2.3.2, it is possible to delegate the role of distributing incoming data traffic among server instances to an automated agent, such as the Load Balancer feature in AWS. In comparison, dedicated servers do not offer any of such features out-of-the-box. Therefore, achieving such functionality requires much greater efforts.

In extension, using Cloud Computing hosting presents us with a versatile cost layout. This will enable us to minimize costs while having optimal utilization of system resources, since the servers will automatically scale on request. For instance, if high peakloads are experienced in a two hour time-frame between 8AM and 10AM, the system can upgrade to a high-performance server instance, for an extra cost. When the data traffic returns to normal behavior, the system will detect the change and react accordingly. Consequently, we will only be charged for the two-hour usage of the high-performance server instance. This is not possible with a dedicated server, since you will pay for the resources, whether they are used or not and it takes a prolonged time to extend the hardware capabilities.

Furthermore, the initial deployment costs of using Cloud Computing are non-existing, because one is not responsible for buying and setting up the hardware needed, whereas with a dedicated server, one will have to pay an extensive sum of money up-front. That being said, we acknowledge, as put forward by Armbrust et al. (2), that Cloud Computing may not be the cheapest long-term solution. However, we believe that the features and potential of Cloud Computing, described in the analysis, outweight this realization.

Table 2.1 shows a comparison of the features provided by Amazon Web Services, Microsoft Windows Azure, and Google App Engine, as have been described earlier in Chapter 2.

Based on our study of Cloud Computing providers, we have concluded that neither Google App Engine or Microsoft Windows Azure is appropriate as a host solution for our system, even though both share adequate levels of performance and scalability.

Google App Engine, being PaaS, is exclusively aimed for web application development, do not provide a platform that is fully able to realize our requirements in terms of customization and functionality, due to its separation between the developer and the infrastructure itself. Also, GAE is limited by its selection of programming languages and its proprietary SDK. Similarly, Microsoft Windows Azure, still being a relatively new platform with unexplored potential, shares the same characteristics of GAE. It is considered PaaS and does not provide us with the opportunity to fully control every level of the system infrastructure and locks the development to Microsoft compliant programming languages only.

We have decided to use Amazon Web Services, mainly because of its high level of flexibility and scalability, which allows us to fully customize our system to our needs and requirements, both in terms of server resources, software architecture, and language- and platform support. Furthermore, its economic price model ensures a low-entry barrier in terms of both deployment- and operational costs, which is suitable for experimental project like this one.

Chapter 3

Design

3.1 System Architecture

This chapter will focus on the design of our back-end system, based on the results of our analysis in Section 2.4. Our design is a three-tier architecture Web service hosted on an Apache Tomcat Server deploying Apache Axis2. The service will be running on a cluster of Amazon EC2 instances controlled by an AWS Load Balancer, as seen in Figure 3.1.

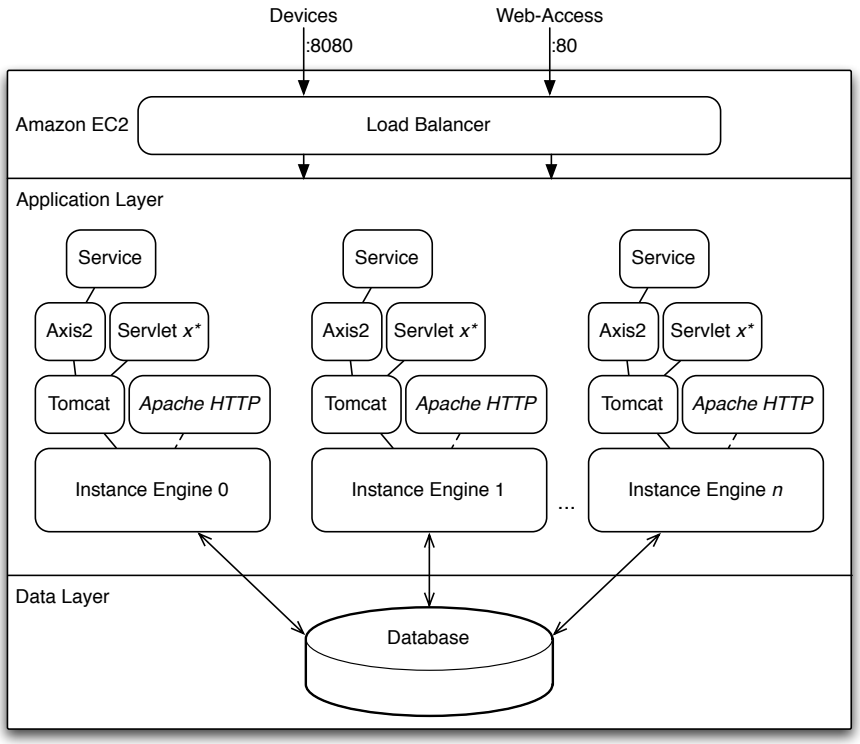

Figure 3.1: An illustration of the system design

An outline description of the depicted components is given in the list below:

- **Load Balancer** The Load Balancer is a feature of AWS. It distributes incoming traffic among a number of EC2 instances running our services, ensuring that traffic is only forwarded to instances with capacity to handle the incoming requests.
- **Service** A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. By using Web services we can distribute the logic to the back-end system while creating light-weight clients, capable of invoking these services.
- **Axis2** The Apache Axis2 project is a Java-based implementation of both the client and server sides of the Web services paradigm described in Section 3.2.
- **Servlet** A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model, such as a Web service. We can run a number of servlets simultaneously, all providing specialized functionality.
- **Tomcat** Tomcat is a popular and well-used open source servlet container, that implements the Java Servlet and JavaServer Pages (JSP) specifications. It is used to host our services including the Axis2 Web service.
- **Apache HTTP** If needed, we can quickly add support for a Apache HTTP Web Server. For example if a designated Web browser application is desired.
- **Instance** In this context, an instance refers to an instantiated Virtual Machine on EC2 running our implementation.
- **Database** The datastorage layer for the system, which can be implemented in several different ways e.g. AWS SQL, AWS RDS and so forth.

By using this architecture, our system is able to support a range of server-side applications, as Apache Tomcat is capable of hosting servlets executing code in languages such as Python and Matlab. This enables us to provide an API with extensive functionality to the mobile clients using the system, which can continuously be updated if necessary. Additionally, referring to Section 2.2, this design decision helps us realize the requirements of multi-language support.

The following sections will explain the Web service paradigm, on which our system is based to achieve compliance with the requirement of platform independency, as put forward in Section 2.2. We will also briefly explain the role and functionality of *Apache Tomcat* and *Apache Axis2*, along with a consideration of datastorage options and front-end design.

3.2 Web Service

The World Wide Web Consortium (W3C) defines a Web Service as:

"*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*" (7)

3.2.1 Web Services Architecture

Web services follows the concept of *Service-Oriented Architecture (SOA)*. SOA models applications are compositions of services provided by components that can be discovered and invoked dynamically. The SOA model defines three actors:

- **The Service Provider** The Service Provider acts as an interface for a system that manages a specific set of tasks.
- **The Service Requester** The Service Requester is an entity that can discover and invoke services.
- **The Service Registry** The Service Registry acts as a repository for the service interfaces published by the service providers.

The relationship between the three actors are shown in Figure 3.2. The concept of Web services following the SOA is that the service provider implements the service and describes the service interface. The provider then publishes the service to the service registry. The service requester then discovers the service, obtains its description and finally invokes the service.

The technologies used to achieve this architecture are, typically, HTTP for transport, XML for data description, SOAP for service invocation, and WSDL for service description. For service discovery, UDDI is used (29). In the following sections these technologies will be explained.

3.2.2 An Overview of XML Technologies

The Web service architecture uses the *eXtensible Markup Language (XML)* as a standardized way to represent data, in a structured, machine-readable way. From a Web service aspect, the most relevant parts of the XML are the XML 1.0 specification (10), namespaces in XML (9) and the XML schema (15).

The XML 1.0 specifications define the core XML as a set of rules for designing text formats for structured data. An XML document consists of markup, which is used to describe the structure, and elements, in which the actual data is contained. An XML document is text based and human readable, however the structure and the rules of the language ensure that a computer can generate and read the data.

The language however does not define any elements; the elements and their meaning are defined by the application. Namespaces in XML is a method for qualifying elements and attribute names to avoid collisions, and attach a specific semantic to them. The use of namespaces in XML makes it possible to define markup vocabularies which can by re-used in different documents.

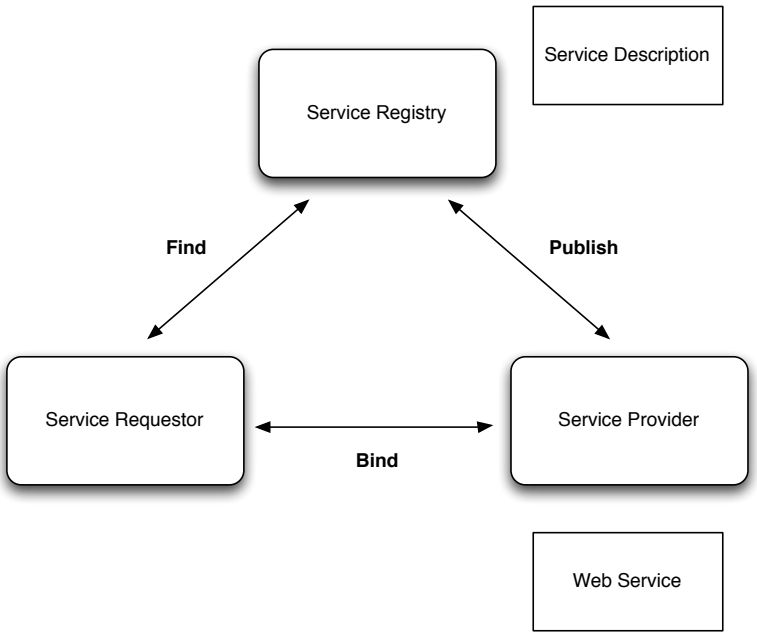

Figure 3.2: Relationship between SOA actors.

The XML scheme is used to describe and constrain the contents of XML documents. Informally put, a schema defines a class of documents. A document that suits the schema is an instance of that schema. Furthermore, the specification provides a standard set of data types which can be used in the schema.

3.2.3 SOAP Messages

The concept of SOAP is a stateless, one-way message exchange paradigm [26]. It is possible to create more complex interactions by combining several features provided by the underlying protocols. One of the key features of SOAP is that it is transport independent, unlike its predecessor, the XML-RPC (18) (26). The XML-RPC was originally created in order to create a light-weight system to serve as the message protocol. As more functionalities were introduced, XML-RPC evolved into SOAP [27]. The SOAP protocol consists of three parts:

[26] http://www.ibm.com/developerworks/xml/library/x-soapbx1/index.html
[27] http://www.xml.com/pub/a/ws/2001/04/04/soap.html

- An envolope, which defines what is in the message and how it should be processed.
- Rules for encoding; expressing application-defined data types.
- A RPC representation for representing remote procedure calls and responses.

An example of a SOAP request over HTTP can be seen in Figure 3.3. As it is quite clear in the example, the SOAP envelope is located in the actual body of the HTTP request, which has its own header and body. The SOAP header blocks usually contain information usable by both the "middle-man" as well as information on the destination of the message. The body then contains information of the actual content of the message.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset= ‘‘utf-8’’
Content-Length: nnnn
SOAPAction:  ‘‘/StockQuote’’

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV= ‘http://schemas.xmlsoap.org/soap/envelope/’
   SOAP-ENV:encodingStyle= ‘‘http://schemas.xmlsoap.org/soap/encoding/’’
    <SOAP-ENV:Header>
              <t:transactionID xmlns:t= ‘‘http//www.stockquoteserver.com/headers’’
                               SOAP-ENV:mustUnderstand= ‘‘1’’>
              124345 </t:transactionID>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m= ‘‘http://www.stockquoteserver.com/methods’’>
            <symbol>DIS</symbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3.3: SOAP message embedded in a HTTP request (26)

An interesting entry in the given example is the "encodingStyle" attribute. This is used to specify the serialization rules used in the message. This example uses the standard SOAP encoding style, which supports primitive numeric, data and string types, arrays and vectors (18). It is possible to use user-defined encoding styles.

In the SOAP RPC mechanism, a method call is a compound data element or struct named after the method to be invoked. In the above example, the "GetLastTradePrice" is the method to be invoked. In this example, the method also contains a "symbol" element as a parameter. A *Universal Resource Identifier (URI)* is used in order to identify the *End-Point Reference (EPR)*. SOAP has no way of conveying the URI, however it relies on the transport protocol to do so. When using HTTP binding, the RPC maps to a HTTP request and respond with SOAP payloads, while the URI is used as the communication EPR.

The use of XML and HTTP for transport makes SOAP available on any platform that is able to handle and process these technologies, which makes it a perfect candidate for the Web Service paradigm.

3.2.4 Web Service Description

SOAP defines a wire protocol for messaging, however, it does not define a way to describe what kind of messages are to be transmitted and where to. In order to address this problem, *Web Service Description Language (WSDL)* is used. WSDL provides a structured way of describing the communication scheme and it can be seen as an interface definition language for Web services. WSDL is an XML grammar for describing network services as a collection of communication endpoints, which are capable of exchanging information. According to the W3C, Web Services are defined by six major elements (11):

- **types** which provides data type definitions used to describe the messages exchanged.
- **message** which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **portType** which is a set of abstract operations. Each operation refers to an input message and output messages.
- **binding** which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- **port** which specifies an address for a binding, thus defining a single communication endpoint.
- **service** which is used to aggregate a set of related ports.

All listed types are described by the WSDL. An illustration of how WSDL structures these elements is shown in Figure 3.4.

3.2.5 Discovery

In the previous section, we discussed WSDL and its ability to describe a service. When looking back at the Web service architecture, the SOA, we still need to define a way to find, or discover, a service. From a Web service perspective, discovery means the process of locating the service provider as well as obtaining the information necessary for the service to be invoked. There are many ways of obtaining this information, some of the simplest being requesting the description documentation from a known location via HTTP or FTP.

However, there is another solution which is much more flexible, and it is very dominantly used (17). This approach is called *Web Services Inspection (WS-Inspection)*, as described in Section 3.2.6. It utilizes a list of references to several service descriptions by using a standard format. This makes it possible to store categorized data about the provided services and the necessary information needed to access said services, and to make queries for the information. Such functionality is provided by *Universal Description, Discovery and Integration (UDDI)*, which will be discussed in Section 3.2.7.

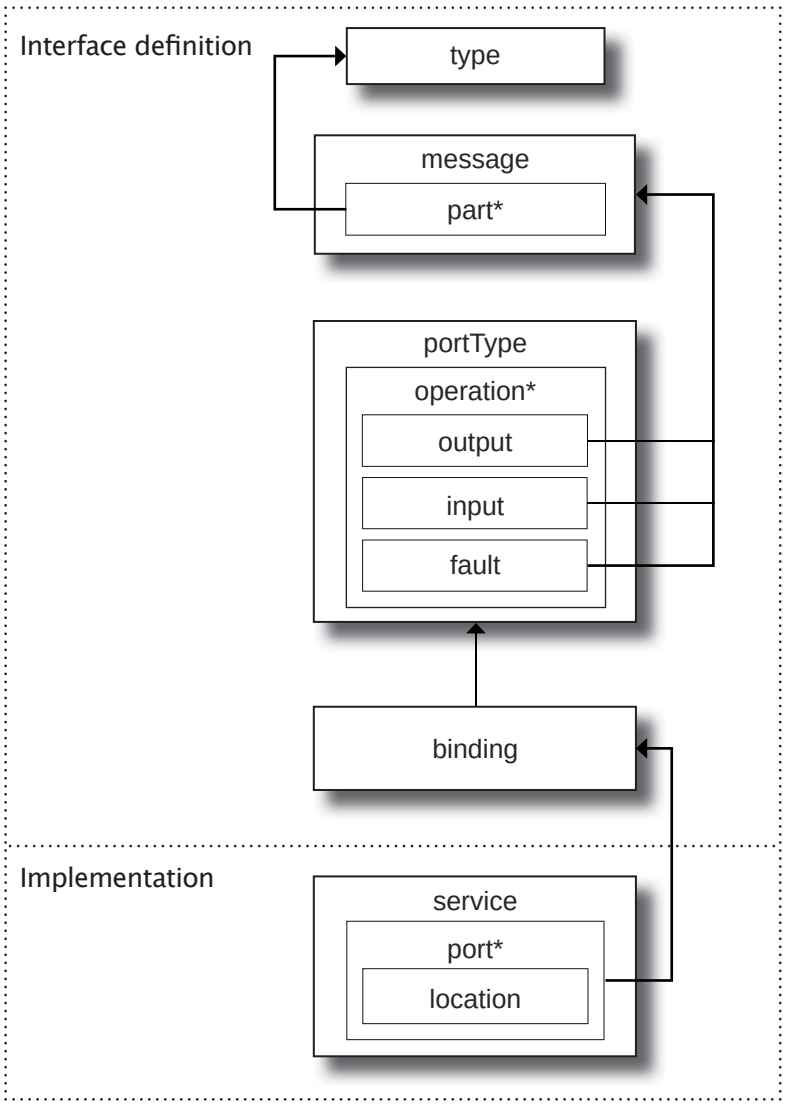

Figure 3.4: The structure of a WSDL definition (26)

3.2.6 Web Service Inspection

Web Service Inspection, also known as WS-Inspection provides an XML document for listing references to service descriptions [28]. A WS-Inspection document will contain one or more *service* and *link* elements. A service element will contain one or more references to different types of service descriptions for the same service. A link element may contain references to one type of service description. Figure 3.5 shows an example of a Web Service Inspection document.

```
<?xml version=‘‘1.0’’?>
<inspection xmlns=‘‘http://schemas.xmlsoap.org/ws/2001/10/inspection/’’>
  <service>
    <abstract>
       An example service
    </abstract>
    <name>BroadcastService</name>
    <description referencedNamespace=‘‘http://schemas.xmlsoap.org/wsdl/’’
                 location=‘‘http://example.com/service.wsdl’’ />
  </service>
  <link
    referencedNamespace=‘‘http://schemas.xmlsoap.org/ws/2001/10/inspection/’’
    location=‘‘http://example.com/additionalservices.wsil’’/>
</inspection>
```

Figure 3.5: An example of a XML WS-Inspection document (26)

The example contains a service element and a link element. The service element contains some basic information, such as the name and a short description of the service, and then it contains a reference to a service description, the reference to the WSDL document. The link element refers to another WS-Inspection document.

3.2.7 Universal Description, Discovery and Integration

UDDI is a set of specifications for defining a standard method for publishing and discovering the network based software components of a SOA (27). UDDI makes it possible to publish information about services and service providers to a central repository, and obtaining said information. IBM and Microsoft used to have public UDDI registries, however, they were discontinued [29]. UDDI defines several core types of information, forming the necessary information needed to use a particular Web service. The core information types are illustrated in Figure 3.6.

Furthermore, UDDI versions 2 and 3 each added an additional data type to facilitate registry affiliation (5). These data types are defined as:

- **publisherAssertion** which defines relationships among entities in the registry.
- **subscription** which defines requests to track changes to a list of entities.

[28] http://www.ibm.com/developerworks/Webservices/library/ws-wsilover/
[29] http://uddi.microsoft.com/about/FAQshutdown.htm

To elaborate on Figure 3.6, it shows that UDDI contains information about service providers; the **businessEntity**, *what services are provided?*, the **businessService**, *where are the services located?*, the **bindingTemplate**, and references to information on how they can be invoked, the **tModel**.

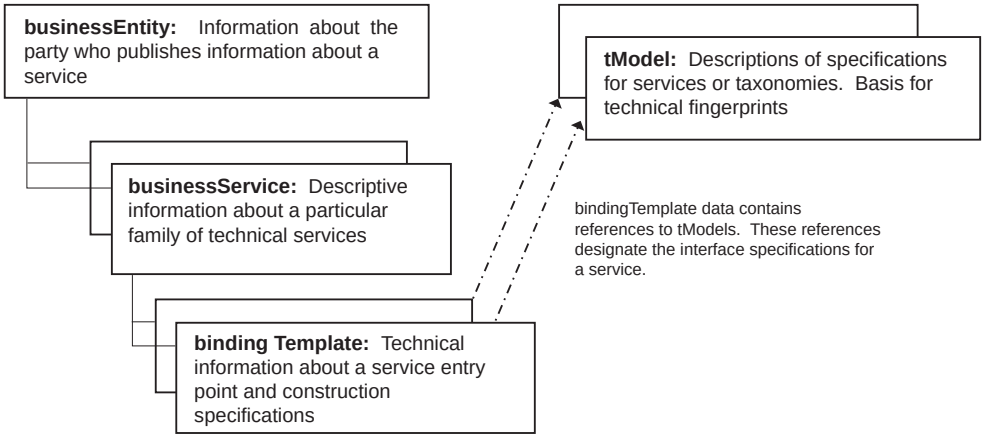

Figure 3.6: An illustration of UDDIs core data types (27)

An UDDI registry can be used to store references to several types of service descriptions, including WSDL descriptions. SOAP is need to access such a registry (5). In order to refer to a WSDL description, the WSDL document is divided into an interface definition and the implementation, as illustrated in Figure 3.4. A tModel data type is then instantiated to reference the interface definition, and the reference to the tModel, as well as the actual service location, which is stored in a bindingTemplate (12).

3.2.8 Summary

Figure 3.7 illustrates how the Web service architecture relates to the technologies discussed above.

The figure shows the service provider implementing the service, and uses a WSDL document to describe its interface. The service provider then publishes the service to the UDDI registry, shown in step 1, which represents the service registry in the SOA. After the service has been published, the service requester can find the service, by querying the UDDI registry, shown in step 2.

The information retrieved from the UDDI registry contains the description of the service, as well as the location of the WSDL interface document. The service requester is then able to retrieve the WSDL document, shown in step 3, and establish the required functionality for accessing and utilizing the service. When these steps have taken place, the service requester is then able to invoke the service, as shown in step 4.

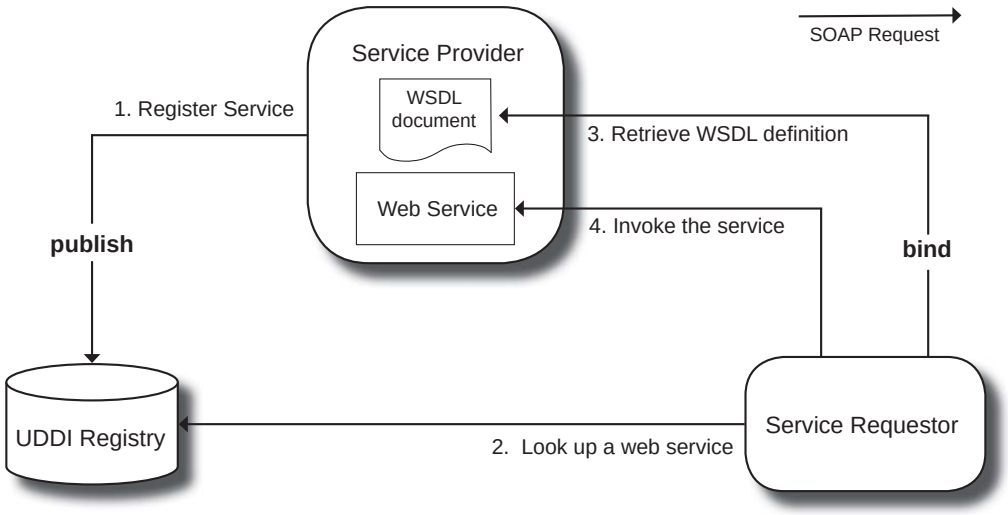

Figure 3.7: A SOA Web Service with Web Service Technology (26)

3.3 Back-end Design

This section gives a brief description of the main components of the design, namely Apache Tomcat and Apache Axis2. Also, we briefly touch on the matter of data storage.

3.3.1 Apache Tomcat

Apache Tomcat is a popular and well-used open source servlet container, that implements the *Java Servlet* and *JavaServer Pages (JSP)* specifications (24). Servlets are Java classes used in order to introduce dynamic content to a Web server. JSP allows developers to "mix" HTML and Java code, enabling the creation of dynamically generated Web sites. Due to the fact that Tomcat will purely be used to host the Axis2 servlet, these technologies will not be described in detail, however, we will describe Axis2 in Section 3.3.2.

3.3.2 Apache Axis2

Axis2 is a core engine for Web services. It is built on a modular architecture which consists of core and non-core modules as seen in Figure 3.8.

The core engine of Axis2 is a pure SOAP engine (21). Axis2 can handle both SOAP message as well as non-SOAP messages. However, because of the nature of the Axis2 core engine at transport level every message has to be converted into a SOAP message[30].

[30] http://www.developer.com/java/ent/article.php/3606466

Additionally, Axis2 uses the *AXIOM* document model for XML message handling. For further information on *AXIOM*, we refer to IBM's documentation on the topic[31].

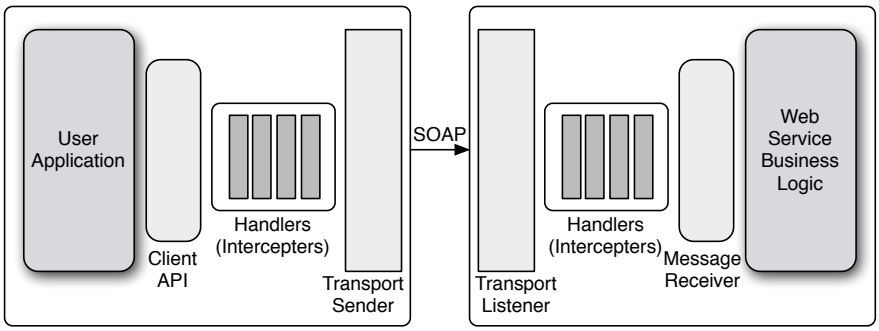

Figure 3.8: Axis2 architecture (16)

Axis2 was originally designed following three key rules[32]:

- Seperation of logic and state to provide a stateless processing mechanism. This is done because Web services are stateless, as discussed in Section 3.2.
- A single information model in which the system is able to suspend and resume runtime.
- Ability to support newer Web service specifications with minimal changes made to the core architecture.

Figure 3.9 illustrates the key core and non-core components in the Axis2 architecture. We will now elaborate on the modules and functionalities of Axis2, which are relevant for our thesis. For further information on Axis2, we refer to the official Apache Axis2 documentation (16).

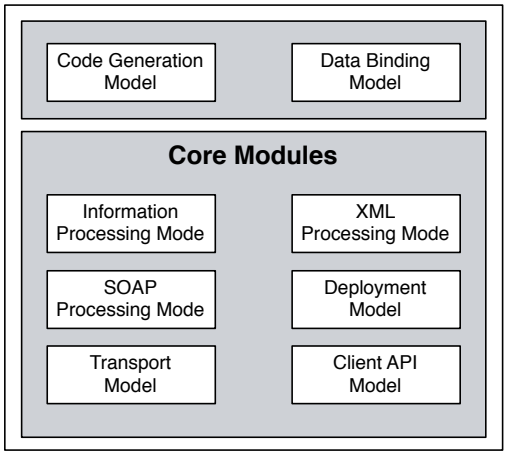

Figure 3.9: Axis2 core modules (21)

[31] http://www.ibm.com/developerworks/Webservices/library/ws-java3
[32] http://onjava.com/pub/a/onjava/2005/07/27/axis2.html

The SOAP processing model can be identified as two basic actions; sending and receiving SOAP messages. The Axis2 architecture provides two "pipes" to perform these basic actions, which are named **in**Pipe and **out**Pipe. The complex *Message Exchange Patterns*, or MEPs, are created by combining these two pipes. The SOAP processing model is provided through handlers. When a SOAP message is being processed, only the handlers registered will be executed; the handlers can either be registered in global, service, or operational scope. The handlers act as interceptors of the SOAP messages. They usually work on the SOAP headers, however, they are also able to access and change the SOAP body as well. The ultimate handler-chain is calculated combining the handlers from all the scopes. This feature is key in Axis2's scalability.

When a SOAP message is sent through the client API, an **out**Pipe activates. This pipe activates the corresponding handlers and ends with a Transport Sender, as sketched in Figure 3.8. The SOAP message is then intercepted by a Transport Receiver which reads the SOAP message and thus activates an **in**Pipe and the corresponding handlers. Finally the Message Receiver consumes the SOAP message[33].

For more information on the SOAP processing model of Axis2, we again refer to the official documentation by The Apache Software Foundation (16).

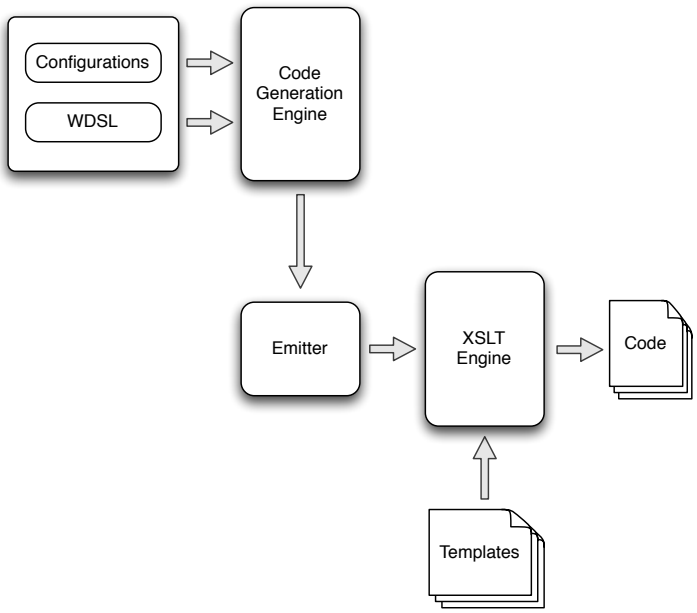

Figure 3.10: The Axis2 code generator (16)

The deployment model in Axis2 in designed to ease the process of deploying services to the system. First of all, Axis2 features a system where the developer can bundle all the library files, class files, resource files and so forth together as an archive file and deploy it by simply dropping it in a specified location. Additionally, Axis2 introduces the features "hot deployment" and "hot update", which its predecessor did not support[34].

[33] http://www.developer.com/java/ent/article.php/
[34] http://www.developer.com/services/article.php/3557741/Understanding-Axis2-Deployment-Architecture.

These features are not a new technical paradigm for the Web service platform, but as mentioned it is new for the Axis platform. Hot deployment simply means that the developer is able to deploy services to the system without having to restart the runtime itself. Intuitively, hot update refers to the ability to make changes to existing Web services without having to restart the runtime. This feature is essential for developers and eases the process of uploading code in a testing environment.

Lastly, the code generation module of Axis2 uses XSL templates, which enables the code generator the flexibility to generate code in multiple languages. The code generator of Axis2 works by generating an XML file and parsing it with a template to generate the code. Figure 3.10 illustrates how the code generation process works.

First an AxisService (16) is populated from a WSDL document. The code generation engine is then able to extract relevant information from the AxisService, in order to create a language independent XML document. By using a XSL template, it is then possible to generate code in the language specified in the provided template. This method allows for code generation in any language, as long as a XSL template can be provided.

We have chosen Axis2 for a number of reasons. It is a Web service engine, which provides us with a better SOAP processing model, with remarkable increases in performance, when compared to Axis 1.x as well as other existing Web service engines. It also provides us with the code generation module, which eases the process of creating a client application, capable of invoking services. Furthermore, it offers the possibility of generating code in other languages, making it much easier to develop client applications for a variety of platforms.

3.3.3 Data Storage

Since this is a preliminary study, we have not currently decided on a final solution for an implementation of the data layer. This choice is highly dependent on requirements of projects running parallel development to this thesis project. Nonetheless, we have several data storage technologies available, of which AWS RDS running MySQL and AWS EBS block storage are likely to suit our needs, due to Amazon's scalable integration of these services alongside EC2.

However, in order to fully solve this problem, further research and development is needed.

3.4 Front-end Design

The emphasis of this thesis is on the overall infrastructure design of system capable of supporting various MoSoNet- and Context-aware applications. Therefore, the actual design of the presentation layer is out of our scope. However, due to the implemented three-tier system architecture, we are able to provide easy integration between numerous client types and the application layer as long as each application implements our Web service API, for which the basis components can be automatically generated by Axis2 via WSDL as described in Section 3.2.4 and Section 3.3.2.

Chapter 4

Implementation

In this chapter we will describe the implementation of a proof-of-concept web service, deployed on the system described Chapter 3, along with an accompanying client application.

4.1 Service

We have used a MySQL database for our proof-of-concept prototype. The service code uses the JDBC Connector/J drivers provided by the MySQL foundation[35]. In order for the reader to fully understand our implementation, it is important to understand what SQL statements are. This will briefly be explained in the following.

SQL stands for *Structured Query Language*. By using SQL, one can display, add, edit and remove records from a table in a database. The SQL queries for these are, respectfully:

```
SELECT, INSERT, UPDATE, DELETE
```

The first example we will show is the *insertText* method, seen in Figure 4.1. JDBC requires a database connection string represented by an URL in order to establish connection to the database. As this implementation will serve as a proof-of-concept prototype, and due to time limitations, we have chosen to implement the MySQL database on-drive on our EC2 instance. Thus, the database URL used is:

```
String url = ‘‘jdbc:mysql://localhost:3306/JavaDB’’
```

However, before actually connecting to the database, the JDBC drivers needs to be properly loaded, which is done in line 4 of Figure 4.1. Now, JDBC should be ready to establish the connection with the database, which is done in line 5, by using *java.sql.DriverManager*'s *getConnection* method. The parameters used in

[35] http://www.mysql.com/products/connector/

this method are the url mentioned above and a username and password, which in this case are *root* and *dontscrewup* respectively.

The parameters of this *insertText* method are the name of a table, the name of a column and some text to put in the database. These parameters are needed for the SQL query used in line 7. The query we use against the database to insert something in the database is defined as:

```
1   INSERT INTO <table>(<column>) VALUES ('<input>')
```

For example, if the service was requested to insert *s082714* into column *studentID* in the table *students*, the query would look like this:

```
1   INSERT INTO students(studentID) VALUES('s082714')
```

In order to execute the query, we first use the *createStatement()* method, which returns an empty JDBC statement object. Then we use the *executeUpdate()* method on this statement, which executes the query. The *executeUpdate()* method is used for the SQL statements listed above. To use the `SELECT` statement, however, one would need to use the *executeQuery()* method instead. After the query has been made, it is necessary to close the connection, which is done in line 8 of Figure 4.1 using the Connection object's *.close()* method.

```
1   public void insertText(String table, String column, ...
        String text){
2   try{
3    Statement stmt;
4    Class.forName("com.mysql.jdbc.Driver");
5    Connection con = DriverManager.getConnection(url,"root","...
        dontscrewup");
6    stmt = con.createStatement();
7    stmt.executeUpdate("INSERT INTO " + table + "(" + column +...
         ") VALUES('" + text + "')");
8    con.close();
9   }catch (Exception e) {e.printStackTrace();}
10  }
```

Figure 4.1: insertText service

Another example we feel is relevant to present is the *showColumns* service. The code for this service can be seen in Figure 4.2. This service follows the same principles as the *insertText* service and uses the JDBC drivers. However, two new objects are introduced here; the *ResultSet* object and *DatabaseMetaData* object. To fully understand how the *showColumns* service works, it is important to understand the properties of these objects, as will be described next.

A *ResultSet* object is a table of data representing a database result set[36]. The *ResultSet* object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. By using the *next()* method on the

[36] http://download.oracle.com/javase/1.4.2/docs/api/java/sql/ResultSet.html

object, the cursor is moved to the next row. When the cursor is at the end of the data table, calling the *next()* method will return a false.

A *DatabaseMetaData* object contains comprehensive information about the database as a whole[37]. Certain *DatabaseMetaData* methods will return a *ResultSet* object, which is the case in this example. Also, *DatabaseMetaData* contains methods that can search through its information about the database. The parameters of such methods are String patterns, and they act as search criteria. If a search pattern criterion parameter is set to null, said criterion will be excluded from the search.

```
1            public String showColumns(String table){
2           s = "";
3           try {
4            Class.forName("com.mysql.jdbc.Driver");
5            Connection con = DriverManager.getConnection(url,"root","...
                 dontscrewup");
6            ResultSet rs;
7            int i = 1;
8            DatabaseMetaData dbmd = con.getMetaData();
9            rs = dbmd.getColumns(null , null, table, null);
10           while (rs.next()){
11            s = s + "Column #" + i + ":\n\tName: "+ rs.getString("...
                  COLUMN_NAME") + "\n\tType: " + rs.getString("...
                  TYPE_NAME") + "\n\tMaximum entry length: " + rs.getInt(...
                  "COLUMN_SIZE");
12            i++;
13           }
14           rs.close();
15           con.close();
16          }catch (Exception e) {e.printStackTrace();}
17          return s;
18         }
```

Figure 4.2: showColumns service

As mentioned, the *showColumns* service follows the same principles as the *insertText* service. However, instead of using a SQL statement to obtain the requested information, a *ResultSet* object and a *DatabaseMetaData* object is used. The information is retrieved by using the *DatabaseMetaData* object's *getColumns* method. This method takes four String parameters as search criteria. as listed below:

```
getColumns(String catalog, String schemaPattern, ...
        String tableNamePattern, String columnNamePattern)
```

The method then performs a search in the entire database based on these parameters. In Figure 4.2 we only use the tableNamePattern parameter as a search criteria, as the rest are set to null.

The *getColumns* method returns a *ResultSet* object containing the results of the performed search. As mentioned earlier, when the cursor is at the end of the data table, calling the *next()* method will return a false. In order to retrieve all data

[37] http://download.oracle.com/javase/1.4.2/docs/api/java/sql/DatabaseMetaData.html

from the *ResultSet* object, we exploit this property by using a while loop to run through the entire *ResultSet* oject. The data extraction itself is done by using *getter* methods on the *ResultSet* object, passing the name of the columns in the data table as parameters for these methods. Finally the data is formatted into a String *s*.

We have now shown how the database can be accessed and manipulated by a service. However, we still need a client in order to invoke the service, which will be explained in Section 4.2.

4.2 Client

In the previous section, we showed how the Web service is implemented in Java, but we still need a client in order to be able to invoke operations of the service. Lets again look at the *insertText()* functionality of the Web service, as described in Section 4.1. However, this time we will take a look from the client's perspective. The implementation of the *insertText()* invocation operation is shown in Figure 4.3.

```
1          public static void insertText(String table, String column,...
               String text){
2         try{
3          DatabaseServiceStub.InsertText var = new ...
               DatabaseServiceStub.InsertText();
4          var.setTable(table);
5          var.setColumn(column);
6          var.setText(text);
7          stub.insertText(var);
8          System.out.println("'" + text +"' inserted in column " + ...
               column + " in table " + table);
9         }catch(Exception e){e.printStackTrace();}
10        }
```

Figure 4.3: insertText invocation method

We have used Axis2s code generation functionality to generate parts of the client code. For information on the process of code generation, we refer to Section 3.3.2. We have used *Axis2 Databinding Framework (ADB)* for code generation. While we acknowledge that ADB is not the most optimal method in terms of power or flexibility, it is the easiest to setup and it sufficiently serves our purpose of developing a proof-of-concept model[38].

However, it should not be perceived as if Axis2 provides an auto generated client, fully ready for invoking services. Instead it generates linkage code from WSDL documents, in order to relieve applications from working directly with *AXIOM*. The generated linkage code is in the form of a stub class, which defines access methods for the application to use, when invoking the services. To make use of this generated stub class, we first create an instance of the stub by using a constructor that takes an endpoint reference pointing to our service, in the form of an URI, as shown below:

[38] http://axis.apache.org/axis2/java/core/docs/userguide-creatingclients.html

```
stub = new DatabaseServiceStub ...
("http://ec2-184-73-93-22.compute-1.amazonaws.com:8080/axis2/services/DatabaseService");
```

Once the instance of the stub has been created, we can call the service-specific access methods to actually invoke the operations. When one of the service methods is called, the stub converts the request data objects to XML, as well as converts returned XML to response data objects for the client.

If we take a closer look at Figure 4.3, line 3 shows how an instance of the generated *InsertText* class is created. In line 4 through 6, we interact with this class, setting the variables to some user input. Finally in line 7, we use the *InsertText* instance as a parameter for the *insertText()* method call.

Figure 4.4 shows the client code for invoking the showColumns operation discussed in Section 4.1. The code follows the same principles as the implementation of invocation method for the *insertText* operation. However, the *showColumns* operation returns a *set* of information to the user.

```
1          public static void showColumns(String table){
2         try {
3          DatabaseServiceStub.ShowColumns var = new ...
               DatabaseServiceStub.ShowColumns();
4          var.setTable(table);
5          ShowColumnsResponse res = stub.showColumns(var);
6          System.out.println(res.get_return());
7         }catch(Exception e){e.printStackTrace();}
8        }
```

Figure 4.4: showColumns invocation method

As mentioned earlier in this section, the stub converts the returned XML into a response data object. Inspecting the code in Figure 4.4, we see in line 4 how an instance of a response object is created. Further, line 5 shows how we retrieve the information from said object.

We refer to Appendix A and B for the entire client- and server java source code. As seen in the examples described above, the process of creating a client code capable of invoking one's Web service becomes relatively trivial, by using the Axis2 generated linkage code.

Furthermore, as described in Section 3.3.2, our decision to useAxis2 makes it possible to generate client code, not only for Java, but for any programming language, as long as a corresponding WSDL-to-code generation tool can be provided.

Chapter 5

Evaluation

5.1 Testing

In this section we will show a small test of our proof-of-concept Web service and its accompanying client, as described in Chapter 4. We will also demonstrate how the AWS Management Console works and how to access its underlying features. The purpose of the test is to validate the postulated claims of rapid development and easy deployment of Web services using the proposed solution.

For testing purposes we have utilized the free-usage tier offered by AWS, mentioned in Section 2.3.3. This free-usage tier provides us with a *micro instance*. This type of instance provides a small and steady amount of CPU resources capable of bursting CPU capacity if supplementary computational power is needed. Even though the processing power of a *micro instance* is equivalent to that of a Nokia N900[39], the setup is sufficient for our testing purposes. For more information on the EC2 instance types available on AWS, we refer to the AWS website[40]. The Management Console interface for AWS is shown in Figure 5.1.

From the management console, it is possible to access all the features of AWS. The interface makes it very easy to create, change or remove instances, or make use of any of the other features that AWS provides. Figure 5.2 shows how one can change the type of an instance to another, through a few simple clicks. This makes it possible to change the type of an instance, for example, from a micro instance, with relatively low computational power, to a GPU cluster instance, capable of very high performance within a short period of time.

Figure 5.3 shows the start-up screen of our prototype client. As previously discussed in Chapter 4, our service and client are a very simple implementations, only capable of performing basic database manipulation, without any real functionality. To present this implementation, we have constructed a straightforward interface, making it possible to interact with our deployed services.

[39] http://www.phoronix.com/scan.php?page=article&item=amazon_ec2_micro&num=1
[40] http://aws.amazon.com/ec2/instance-types/

We refer to Appendix C for instructions in order to gain hands-on access to the client.

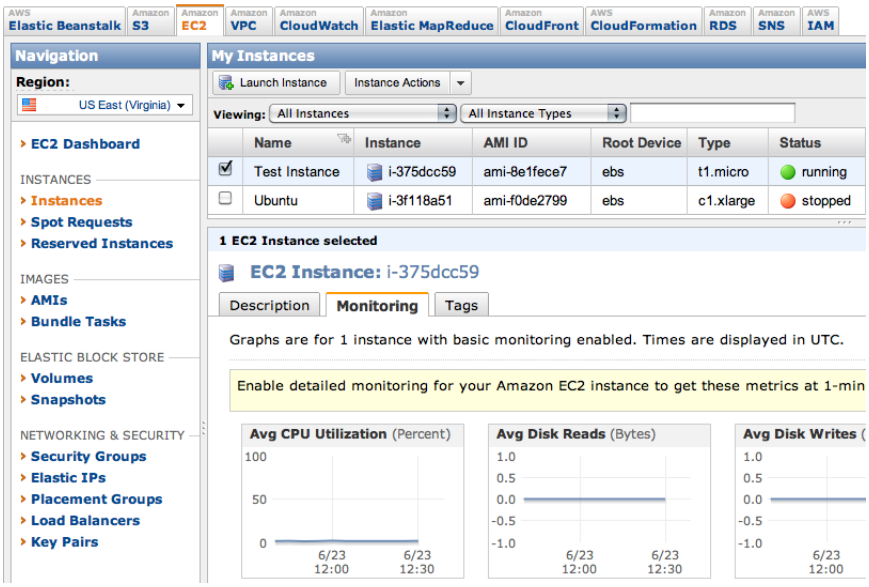

Figure 5.1: AWS Management Console showing running EC2 Instances

Even though the implemented Web service and client application is relatively simplistic and bare-boned, it serves as a noteworthy result for backing up our assesments outlined throughout the thesis. Development of the service and application itself was straightforward. However, we did experience a series of initial issues regarding the setup and configuration of the system infrastructure. For example, it proved to be challenging to configure our EC2 instance along with Tomcat and Axis2, as all interaction with the instance had to be done through command-line interfaces and SSH.

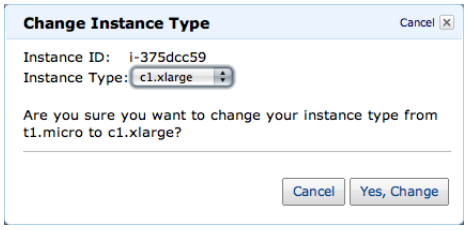

Figure 5.2: Upscaling from a micro instance to a larger machine

However, we acknowledge that the troubles we encountered with this may be a result of lacking experience with the tools. The initial configuration of Axis2, in order to make it operate within our requirements, also proved to be troublesome at first. For example, we had some challenges enabling the Hot Update functionality, as described in Section 3.3.2.

Nevertheless, once we had overcome these issues it was fairly easy to replicate the setup for later use. Additionally, it is possible to save the entire system setup as a

so-called *Amazon Machine Image (AMI)*[41], which is a preconfigured machine image for fast deployment of EC2 instances.

Figure 5.3: User interface of prototype application

Once the system was running, the actual development of services was only a matter of writing code for the wanted features and upload the service. Even services with complex functionalities, once developed, are relativelse trouble-free to deploy using the Axis2 tools and client invocation of said services can be created a minimum of efforts.

5.2 Discussion

Based on our test results and experiences learned throughout the project, we found the system to succesfully live up to our expectations and needs for fast and easy distribution of diverse Web services. We believe to have shown that our solution and current implementation forms a strong corner stone for future development of a framework for context-aware applications and advanced social group analysis, as visioned in Chapter 1.

Despite the positive outcome of our work, we acknowledge that the implications of our research results may not be applicable for a direct implementation of a full scale system, based on the design presented in Chapter 3.

First of all, it has not been possible to gather in-depth requirements from other participating groups in the Milab DTU research programme as intended, since their projects have been undertaken parallel to our work, resulting in a diffused inter-

[41] http://aws.amazon.com/amis/

pretation of specifications in terms of computational requirements, data workload, and network traffic loads. For this reason, our preliminary study of hardware needs has proved to be inclusive due to the lack of key parameters as mentioned above.

Therefore, instead of postulating claims based on abstract speculations, we have based our design and implementation decisions on results put forward in recent studies of similar issues, such as the work of Berriman et al. (22), Iosup et al. (19), and Akioka and Muraoka (1). Generally, these studies show a significant potential of utilizing Cloud Computing for high-performance computing, based on performance benchmarking of Amazon EC2 for a set particular scientific applications. However, we recognize that usage of Amazon EC2 may not be the best solution in all cases, as depicted by Jackson et al. (20).

It is our impression, that we do not currently hold the knowledge required to make a sufficiently tailored benchmark of the EC2 instances, in order to comprehend which hardware configuration would best accomodate the needs of a campus-wide deployment of the proposed system. Once we acquire this understanding, it will be possible to set up a suitable performance benchmark analysis using available software, such as *Phoronix Test Suite*[42]. An example of a benchmark of the available EC2 instance types, using Phonorix test tools is displayed in Figure 5.4.

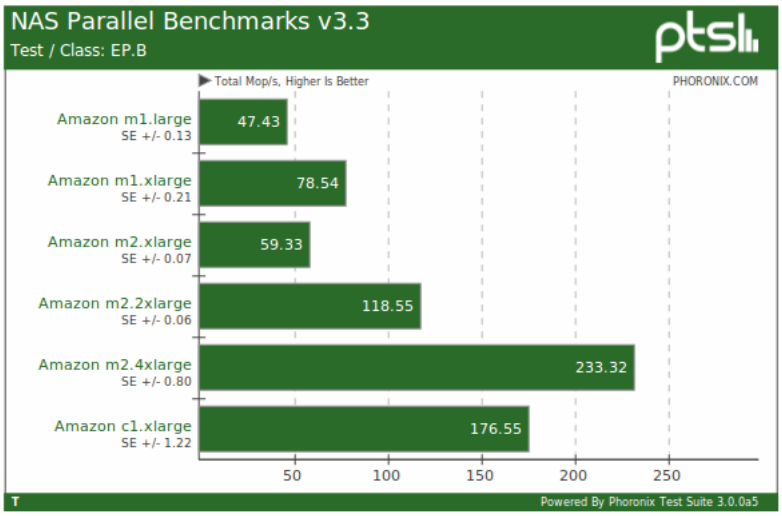

Figure 5.4: NAS Parallel Benchmark V3.3 - EP.B Test Class

This specific CPU benchmark, provided in a test package developed by NASA[43], analyses the multi-threading capabilities of Amazon EC2 instances, by generating independant gaussian random variates, measured in Million operations per second (Mop/s). For further details on the shown CPU performance test and related benchmarks, we refer to the full article, published on the Phoronix website[44].

Secondly, as we wanted to come up with a cost-effective solution for this system, a detailed cost analysis is needed order to gain a thorough understanding of the economic consequences of putting our system into existence. Regardless, as a cost

[42] http://www.phoronix.com/
[43] http://www.nas.nasa.gov/Resources/Software/npb.html
[44] http://www.phoronix.com/scan.php?page=article&item=amazon_ec2_exhaustive

analysis of utilized resources requires clear overview of involved hardware components, which is not currently available. Therefore, as we cannot ourselves provide an accurate estimate of the annual operational costs of hosting our framework on Amazon Web Services, we have to some extent ommitted this perspective in our decision making.

However, we acknowledge recent academic studies on the topic, such as the work of Kossmann et al. (23), in which a comphrensive overview of cost related to using Cloud Computing services provided Amazon, Google, and Microsoft is given. Additionally, we spent time familiarizing ourselves with the Amazon EC2 Pricing tool, mentioned in Section 2.3.3, in order to get an approximate of costs of various EC2 configurations.

For example, we calculated that the annual cost of running a large EC2 instance with a peakload compensation of an extra high-performance machine to be between $2031 and $3005, depending on the payment model used. We have attached a copy of the pricing calculator tool, with our inputted values, as a part the digital contents of this thesis, detailed in Appendix C.

5.3 Future Work

We have now taken the first steps towards establishing a framework for hosting context-aware web services. However, as reviewed in Section 5.2, there are still many issues to overcome in order to achieve the vision put forward in the opening statement. Based on the discussion in Section 5.2, we have summarized a set of tasks for future research below:

- Conduct a study in collaboration with essential project stakeholders from DTU in order to clarify the actual scale of required hardware resources and functionalities of our system, to facilitate a campus wide deployment such as average data traffic and expected user behavior.
- Coordinate a detailed performance benchmark test of the back-end system, tailored to comply with expected workloads and foreseen growth of users after deployment.
- Provide an in-depth cost analysis of actual expenses in regards deployment of the full scale system.
- Devise a common database- and data storage model to handle numerous different data sources and developer needs, while ensuring system scalability. Furthermore, this database model must be able to handle personal information and other sensitive data in a secure manner.
- Develop an API suitable for rapid development of context-aware mobile applications, providing easy access to data storage and tools for complex computational processing. In addition, a protocol for API extensions must be formulated.

Chapter 6

Conclusion

This thesis has reviewed the requirements needed in order to construct a framework for context-aware application development. We have studied recent research in context-awareness, Mobile Social Networks, and Cloud Computing in order to gain a perspective on the topic at hand.

Based on an initial literature study, we sat up a series high level requirements for a back-end system capable of supporting context-aware mobile applications and necessary data handling. Mainly, the system must be scalable, extendable, platform independent and support multiple programming languages. Furthermore, it should be cost efficient. In order to comply with the requirements specified, we have conducted a feasibility analysis of currently available solutions in software architecture and network technology.

The analytical results gave way to a system design consisting of a three-tier architecture Web service framework deployed on a Cloud Computing hosting platform, offered by Amazon Web Services. The internal components are based on an Apache Tomcat Server deploying Apache Axis2. The Web Service paradigm proved to be the most suitable solution for the implementation of our system, in regards to the preliminary requirements set up, but also due to its flexible design structure and low deployment costs.

We have developed and implemented a proof-of-concept prototype application on the proposed system that is capable of interacting with our deployed Web service through a custom API, in order to verify the hypothesis that our solution ensures rapid development times and easy service deployment.

Lastly, we reviewed our findings and proposed future work needed in order to successfully deploy this system in its full extent for the Milab Context-aware research programme. We found the system to succesfully live up to our expectations and requirements. However, we acknowledge that the implications of our research results may not be directly applicable for a full scale implementation of our solution.

In conclusion, we have successfully carried out the thesis objectives outlined in Section 1.3.

Bibliography

[1] Sayaka Akioka and Yoichi Muraoka. HPC Benchmarks on Amazon EC2. *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, pages 1029–1034, 2010.

[2] Michael Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and Others. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[3] Aaron Beach, Mike Gartrell, Sirisha Akkala, Jack Elston, John Kelley, Keisuke Nishimoto, Baishakhi Ray, Sergei Razgulin, Karthik Sundaresan, Bonnie Surendar, and Michael Terada. WhozThat? Evolving an Ecosystem for Context-Aware Mobile Social Networks. *Ieee Network*, (August):50–55, 2008.

[4] Aaron Beach, Mike Gartrell, Xinyu Xing, Richard Han, Qin Lv, Shivakant Mishra, and Karim Seada. Fusing Mobile, Sensor, And social Data To Fully Enable Context-Aware Computing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications - HotMobile '10*, page 60, New York, New York, USA, 2010. ACM Press.

[5] Tom Bellwood, Steve Capell, Luc Clement, John Colgrave, Matthew J. Dovey, Daniel Feygin, Andrew Hately, Rob Kochman, Paul Macias, Mirek Novotny, Massimo Paolucci, Claus von Riegen, Tony Rogers, Katia Sycara, Pete Wenzel, and Zhe Wu. UDDI Version 3.0.2. *UDDI Spec Technical Committee Draft*, 2004.

[6] G.B. Berriman, Gideon Juve, Ewa Deelman, Moira Regelson, and Peter Plavchan. The Application of Cloud Computing to Astronomy: A Study of Cost and Performance. In *2010 Sixth IEEE International Conference on e ?? Science Workshops*, pages 1–7. IEEE, December 2010.

[7] David Booth, Hugo Haas, Francis McCab, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. *W3C Working Group*, 2004.

[8] Dario Bottazzi, Rebecca Montanari, and Alessandra Toninelli. Middleware for Anytime, Anywhere Social Networks. *IEEE Intelligent Systems*, (October):23–32, 2007.

[9] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, and Henry S. Thompson. Namespaces in XML 1.0 (Third Edition). *W3C Recommendation*, 2009.

[10] Tim Bray, Jean Paoli, Eve Maler, and Francois Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C Recommendation*, 2008.

[11] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. *W3C Note*, 2001.

[12] F Curbera and D Ehnebuske. Using WSDL in a UDDI Registry, Version 1.07, UDDI Best Practice. 2002.

[13] Florian Daniel and Maristella Matera. Mashing up context-aware Web applications: A component-based development approach. *Web Information Systems Engineering-WISE 2008*, pages 250–263, 2008.

[14] Nathan Eagle, Alex Sandy Pentland, and David Lazer. Inferring Friendship Network Structure By Using Mobile Phone Data. *Proceedings of the National Academy of Sciences of the United States of America*, 106(36):15274–8, September 2009.

[15] David C. Fallside and Priscilla Walmsley. XML Schema Part 0: Primer Second Edition. *W3C Recommendation*, 2004.

[16] The Apache Software Foundation. Apache Axis 2 / Java Version 1.6.0 Documentation, 2011.

[17] John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos, and A. Tsakalidis. Web service discovery mechanisms: looking for a needle in a haystack? In *International Workshop on Web Engineering*, volume 38. Citeseer, 2004.

[18] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarka, and Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). *W3C Recommendation*, 2007.

[19] Alexandru Iosup, Simon Ostermann, N. Yigitbasi, Radu Prodan, Thomas Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011.

[20] K.R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, H.J. Wasserman, and N.J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. *2nd IEEE International Conference on Cloud Computing Technology and Science*, pages 159–168, November 2010.

[21] Deepal Jayasinghe. *Looking Into Axis2*, chapter Chapter 2. Number 2. Birmingham: Packt Publishing Ltd, 2008.

[22] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P. Berman, and Phil Maechling. Scientific workflow applications on Amazon EC2. *2009 5th IEEE International Conference on E-Science Workshops*, pages 59–66, December 2009.

[23] Donald Kossmann, Tim Kraska, and Loesing Simon. An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In *Proceedings of the 2010 international conference on Management of data*, pages 579–590, New York City, New York, USA, 2010. ACM.

[24] Budi Kurniawan and Paul Deck. *How Tomcat Works*. BrainySoftware.com, 2004.

[25] Jakob Eg Larsen and Kristian Jensen. Mobile context toolbox: an extensible context framework for s60 mobile phones. In *Proceedings of the 4th European conference on Smart sensing and context*, 2009.

[26] Sonera Plaza Ltd and MediaLab. Web Services White Paper, 2002.

[27] U. Oasis. Introduction to UDDI: Important features and functional concepts, 2004.

[28] George Reese. *Distributed Application Architecture*, chapter 7, pages 126–145. O'Reilly & Associates, second edition, 2000.

[29] Ian J. Taylor. *From P2P to Web services and grids: peers in a client/server world*. Springer, first edition, 2004.

[30] Thomas Springer Thomas Hamann, Gerald Hübsch. A Model-Driven Approach For Developing Adaptive Software Systems. In *Distributed Applications and Interoperable Systems*, pages 196–209. Springer, 2008.

[31] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):291, June 2005.

Appendix A

Client Source Code

```
1  package ws.client;
2
3  import java.util.Scanner;
4
5  import ws.client.DatabaseServiceStub.ShowColumnsResponse;
6  import ws.client.DatabaseServiceStub.ShowTablesResponse;
7
8  public class DatabaseClient {
9
10   static DatabaseServiceStub stub;
11   static Scanner console = new Scanner(System.in);
12   static String inputData, clientTable;
13
14   public static void main(String[] args) {
15     try {
16       stub = new DatabaseServiceStub
17       ("http://ec2-184-73-93-22.compute-1.amazonaws.com:8080/axis2/...
             services/DatabaseService");
18
19       /* Interface start */
20       System.out.println("\n\n...
             +******************************************
21       **************************+");
22       System.out.println("|Welcome to the test client for Emil ...
             Lysgaard Hansen's and Soeren Fuhr's  |");
23       System.out.println("|bachelor project. This client has been ...
             constructed as a simple          |");
24       System.out.println("|proof-of-concept model, enabling us to ...
             present our case in a hands-on   |");
25       System.out.println("|approach. This interface will present the...
              user with a series of choices,|");
26       System.out.println("|enabling the user to make queries against...
              the connected database.       |");
27       System.out.println("...
             +***********************************************
28       ***********************+\n\n");
29
30       System.out.println("The tables currently present in the ...
             database are:");
```

```
31        showTables();
32        //The user inputs name of table and column.
33        System.out.println("\nPlease input the name of the desired ...
              table:");
34        clientTable = console.next();
35        System.out.println("\nThe columns present in table " + ...
              clientTable + " are:");
36        showColumns(clientTable);
37
38        while(true){
39
40          //Present the user with choices
41          System.out.println("\n\n...
                +****************************************
42          ********************************+");
43          System.out.println("INFO:");
44          System.out.println("The currently selected table is '" + ...
                clientTable +"'");
45          System.out.println("The columns present in the selected ...
                table are:");
46          showColumns(clientTable);
47          System.out.println("...
                +****************************************
48          ***************************+\n\nPlease select an option:");
49
50
51          System.out.println("Press '1' to select another table.");
52          System.out.println("Press '2' to create a new table.");
53          System.out.println("Press '3' to view the columns in the ...
                selected table.");
54          System.out.println("Press '4' to view the content of a user ...
                specified column in the selected table.");
55          System.out.println("Press '5' to create a new column in the ...
                selected table.");
56          System.out.println("Press '6' to insert text in a user ...
                specified column in the selected table.");
57          System.out.println("Press '7' to delete a column in the ...
                selected table.");
58          System.out.println("Press '8' to delete a table.");
59          System.out.println("Press '0' to terminate the program.");
60
61          //Read choice of user
62          inputData = console.next();
63
64          //Analyze input and act upon
65          if (inputData.equals("1")) {
66            System.out.println("The tables currently present in the ...
                  database are:");
67            showTables();
68            System.out.println("\nPlease select one of the presented ...
                  tables:");
69            clientTable = console.next();
70          } else if (inputData.equals("2")){
71            System.out.println("The tables currently present in the ...
                  database are:");
72            showTables();
73            System.out.println("Please enter a name for the table:");
74            clientTable = console.next();
75            System.out.println("Please enter a name for the default ...
                  column:");
```

```
76            String column = console.next();
77            createTable(clientTable, column);
78          } else if (inputData.equals("3")){
79            System.out.println("The columns in " + clientTable + " are...
                  :");
80            showColumns(clientTable);
81          } else if (inputData.equals("4")){
82            System.out.println("Please select one of the following ...
                  columns:");
83            showColumns(clientTable);
84            String column = console.next();
85            printContentOfColumn(clientTable, column);
86          } else if (inputData.equals("5")){
87            System.out.println("Please input a name for the column:");
88            String column = console.next();
89            System.out.println("Please input the type to be contained ...
                  in the column:");
90            System.out.println("(If in doubt just input 'varchar')");
91            String type = console.next().toUpperCase();
92            System.out.println("Please input the maximum length of ...
                  entries in the column:");
93            String length = console.next();
94            createColumn(clientTable, column, type, length);
95            System.out.println("Column created!");
96          } else if (inputData.equals("6")){
97            System.out.println("Please select one of the following ...
                  columns:");
98            showColumns(clientTable);
99            String column = console.next();
100           System.out.println("Please input what to be stored in the ...
                  column:");
101           System.out.println("(No spaces allowed!)");
102           String text = console.next();
103           insertText(clientTable, column, text);
104           System.out.println("Text inserted!");
105         } else if (inputData.equals("7")){
106           System.out.println("Please select one of the following ...
                  columns to delete:");
107           showColumns(clientTable);
108           String column = console.next();
109           deleteColumn(clientTable, column);
110         } else if (inputData.equals("8")){
111           System.out.println("Please select one of the following ...
                  tables to delete:");
112           showTables();
113           String table = console.next();
114           deleteTable(table);
115         }
116         else if (inputData.equals("0")){
117           break;
118         }
119       }
120     }catch(Exception e){e.printStackTrace();}
121   }
122
123   public static void createTable(String table, String column){
124     try{
125       DatabaseServiceStub.CreateTable var = new DatabaseServiceStub....
              CreateTable();
126       var.setTableName(table);
```

```
127         var.setColumnName(column);
128         stub.createTable(var);
129       }catch(Exception e){e.printStackTrace();}
130
131     }
132
133     public static void deleteTable(String table){
134       try {
135         DatabaseServiceStub.DeleteTable var = new DatabaseServiceStub....
              DeleteTable();
136         var.setTable(table);
137         stub.deleteTable(var);
138       }catch(Exception e){e.printStackTrace();}
139     }
140
141     public static void createColumn(String table, String column, String...
          type, String length){
142       try {
143         DatabaseServiceStub.CreateColumn var = new DatabaseServiceStub....
              CreateColumn();
144         var.setTable(table);
145         var.setColumn(column);
146         var.setType(type);
147         var.setLength(length);
148         stub.createColumn(var);
149       }catch(Exception e){e.printStackTrace();}
150     }
151
152     public static void deleteColumn(String table, String column){
153       try {
154         DatabaseServiceStub.DeleteColumn var = new DatabaseServiceStub....
              DeleteColumn();
155         var.setTable(table);
156         var.setColumn(column);
157         stub.deleteColumn(var);
158       }catch(Exception e){e.printStackTrace();}
159     }
160
161     public static void showTables(){
162       try {
163         ShowTablesResponse res = stub.showTables();
164         System.out.println(res.get_return());
165       }catch(Exception e){e.printStackTrace();}
166     }
167
168     public static void showColumns(String table){
169       try {
170         DatabaseServiceStub.ShowColumns var = new DatabaseServiceStub....
              ShowColumns();
171         var.setTable(table);
172         ShowColumnsResponse res = stub.showColumns(var);
173         System.out.println(res.get_return());
174       }catch(Exception e){e.printStackTrace();}
175     }
176
177     public static void printContentOfColumn(String table, String column...
          ){
178       try {
179         DatabaseServiceStub.PrintContentsOfColumn var = new ...
              DatabaseServiceStub.PrintContentsOfColumn();
```

```
180         var.setTable(table);
181         var.setColumn(column);
182         DatabaseServiceStub.PrintContentsOfColumnResponse res = stub....
                printContentsOfColumn(var);
183         System.out.println("Contents of the column " + column + " in " ...
                + table + " are:\n" +res.get_return());
184       }catch(Exception e){e.printStackTrace();}
185     }
186
187     public static void insertText(String table, String column, String ...
          text){
188       try{
189         DatabaseServiceStub.InsertText var = new DatabaseServiceStub....
                InsertText();
190         var.setTable(table);
191         var.setColumn(column);
192         var.setText(text);
193         stub.insertText(var);
194         System.out.println("'" + text +"' inserted in column " + column...
                + " in table " + table);
195       }catch(Exception e){e.printStackTrace();}
196     }
197 }
```

Appendix B

Server Source Code

```
1  package ws.client;
2
3  import java.sql.*;
4
5  public class DatabaseService {
6    static String url = "jdbc:mysql://localhost:3306/JavaDB";
7    static String s;
8
9
10   public void createTable(String tableName, String columnName){
11     try {
12       Class.forName("com.mysql.jdbc.Driver");
13       Connection con = DriverManager.getConnection(url,"root","...
             dontscrewup");
14       Statement stmt;
15       stmt = con.createStatement();
16
17       stmt.executeUpdate("CREATE TABLE " + tableName + " (" + ...
             columnName + " char(15))");
18       con.close();
19     }catch (Exception e) {e.printStackTrace();}
20   }
21
22   public void deleteTable(String table){
23     try {
24       Class.forName("com.mysql.jdbc.Driver");
25       Connection con = DriverManager.getConnection(url,"root","...
             dontscrewup");
26       Statement stmt;
27       stmt = con.createStatement();
28       stmt.executeUpdate("DROP TABLE " + table);
29     }catch (Exception e) {e.printStackTrace();}
30   }
31
32   public void createColumn(String table, String column, String type, ...
       String length){
33     try{
34       Class.forName("com.mysql.jdbc.Driver");
35       Connection con = DriverManager.getConnection(url,"root","...
```

```
            dontscrewup");
36        Statement stmt;
37        stmt = con.createStatement();
38        stmt.executeUpdate("ALTER TABLE " + table + " ADD COLUMN " + ...
              column + " " + type + " (" + length + ")");
39      }catch (Exception e) {e.printStackTrace();}
40    }
41
42    public void deleteColumn(String table, String column){
43      try{
44        Class.forName("com.mysql.jdbc.Driver");
45        Connection con = DriverManager.getConnection(url,"root","...
              dontscrewup");
46        Statement stmt;
47        stmt = con.createStatement();
48        stmt.executeUpdate("ALTER TABLE " + table + " DROP COLUMN " + ...
              column);
49      }catch (Exception e) {e.printStackTrace();}
50    }
51
52    public void insertText(String table, String column, String text){
53      try{
54        Statement stmt;
55        Class.forName("com.mysql.jdbc.Driver");
56        Connection con = DriverManager.getConnection(url,"root","...
              dontscrewup");
57        stmt = con.createStatement();
58        stmt.executeUpdate("INSERT INTO " + table + "(" + column + ") ...
              VALUES('" + text + "')");
59        con.close();
60      }catch (Exception e) {e.printStackTrace();}
61    }
62
63    public String showTables(){
64      s = "";
65      try{
66        Class.forName("com.mysql.jdbc.Driver");
67        Connection con = DriverManager.getConnection(url,"root","...
              dontscrewup");
68        ResultSet rs;
69        int i = 1;
70        DatabaseMetaData dbmd = con.getMetaData();
71        rs = dbmd.getTables(null, null, null, new String[]{"TABLE"});
72        while (rs.next()){
73          s = s + "Table #" + i + ": " + rs.getString("TABLE_NAME");
74          i++;
75        }
76        rs.close();
77        con.close();
78      } catch (Exception e) {e.printStackTrace();}
79      return s;
80    }
81
82    public String showColumns(String table){
83      s = "";
84      try {
85        Class.forName("com.mysql.jdbc.Driver");
86        Connection con = DriverManager.getConnection(url,"root","...
              dontscrewup");
87        ResultSet rs;
```

```
88        int i = 1;
89        DatabaseMetaData dbmd = con.getMetaData();
90        rs = dbmd.getColumns(null , null, table, null);
91        while (rs.next()){
92          s = s + "Column #" + i + ":\n\tName: "+ rs.getString("...
              COLUMN_NAME") + "\n\tType: " + rs.getString("TYPE_NAME") +...
               "\n\tMaximum entry length: " + rs.getInt("COLUMN_SIZE");
93          i++;
94        }
95        rs.close();
96        con.close();
97      }catch (Exception e) {e.printStackTrace();}
98      return s;
99    }
100
101   public String printContentsOfColumn(String table, String column){
102     try {
103       Class.forName("com.mysql.jdbc.Driver");
104       Connection con = DriverManager.getConnection(url,"root","...
            dontscrewup");
105       Statement stmt;
106       ResultSet rs;
107       s = "";
108       int i = 1;
109       String temp;
110       stmt = con.createStatement();
111       rs = stmt.executeQuery("SELECT * from " + table + " ORDER BY " ...
            + column);
112       while(rs.next()){
113         temp = rs.getString(column);
114         s = s + " " + "Row #" + i + ": " + temp + "\n";
115         i++;
116       }
117       rs.close();
118       con.close();
119     } catch (Exception e) {e.printStackTrace();}
120     return s;
121   }
122 }
```

Appendix C

Digital Thesis Contents

The Digital Contents of this thesis, available at http://www.sorenfuhr.com/BSc2011.zip, contains a both the print- and digital version of this thesis, The source code and a runable instance of the client application. Furthermore, a copy of Amazons pricing calculator is also attached. An overview of the included files is seen below:

/BScLysgaardFuhr_thesisPrint
A print version of the thesis.

/BScLysgaardFuhr_thesisNet
A digital version of the thesis.

/SourceCode/
The directory containing the source code of the implemented Web service and client application.

/SourceCode/Client.jar
A runnable copy of the client application. To run the file, open a terminal window, locate the directory containing the client.jar file and type `java -jar client.jar`. The application should now be running and connected to the Web service hosted on Amazon EC2. Make sure to have the latest version of Java Runtime installed, before running the application.

/Amazon_EC2_Cost_Comparison_Calculator
The Amazon pricing calculator for devising annual costs of using various EC2 instances, as discussed in Chapter 5.

www.milab.imm.dtu.dk

Department of Informatics and Mathematical Modelling
Mobile Informatics Lab (Milab)
Technical University of Denmark
building 321
DK-2800 Kgs. Lyngby
Denmark
Tel: +45 45 25 33 51
Fax: +45 45 88 26 73
E-mail: milab@imm.dtu.dk