Institute of Architecture of Application Systems
University of Stuttgart
Universittsstrae 38
D–70569 Stuttgart

Studienarbeit Nr. 2446

Improving and updating the Nefolog system

Na Chen

Course of Study:	Computer Science
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dr. Vasilios Andrikopoulos
Commenced:	December.12, 2013
Completed:	June.13, 2014
CR-Classification:	D.2.1, D.2.9, H.3.3, H.5.2

Abstract

An increasing number of companies are offering nowadays Cloud Computing services. For example, they offer different services like doing calculations on virtual machines with different CPU cores and RAM sizes. A growing number of enterprises take advantage of these services for efficient data processing. The Nefolog system helps users to identify the most cost efficient cloud provider for their needs. However, the existing system provides RESTful services and does not offer a graphical interface for direct interaction with the users. This work designs and implements such an interface as a Web application, so that users can get information out of the Nefolog system in an easy to use manner.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Cloud computing develops quickly from day to day. For many companies, this brings advantages like resource conservation and reduction of costs because it is not necessary to run an own computer center. Another benefit is that the use of cloud computing normalizes the IT-System and gets it safely without having to do run redundant hardware or doing backups by the company itself. This chapter introduces the motivation, the specific problems, as well the outline of this thesis.

1.1 Motivation

Nowadays, almost all companies need IT-support independent of if the size of the company is large, medium, or small. IT-support is also needed forall kinds of business, both in the IT domain, but also in other domains, e.g. chemical companies etc. IT-technical support plays an important role for the company infrastructure. A successful IT-support can also bring profit for the company However, building the IT-department it costs the company relatively much investment, employees, equipment, and space. For large size companies using cloud services or having own equipment is possible. Such companies anyway have a large computing department with professional IT-engineers and advanced equipment. However, for a small size of companies, they don't have a professional team for the IT-system. Here cloud computing could be a good solution for them, so they don't need to employ people for the IT-service by themselves, also they don't need a large server for storing the data. Another advantage is that the hardware needed for processing can be scaled with demand. If the company at peak times needs more performance than usually this can be easily handled using clould services. Secondary but very important services like doing backups of the data are included in the service.

For example, if small company uses Google Apps and Google Docs, not just for data processing, like to creating tables or powerpoint, the data are also always saved in the same place and can be accessed wherever you are. The advantage of giving the data to the provider is that the provider takes care about that the resources in the computing center are fit for the need as well as data protection is taken care about. Small or Mid-size companies may not have the time to pay attention on such details. Instead of that such companies can concentrate on their main business safely end efficiently.

Because of that Mid-size companies are the main users for cloud computing. First those companies have own IT technical personal but they do not pay attention on how to build the basic technical infrastructure. They are challanged on how to integrate the IT into the business processes to maximize the benefit for the company. Secendly mid sized companies are often in the state of expansion. In this process the focus is mostly on the own business and not on the IT-Department.

These days, more and more cloud computing providers are bringing lots of different offerings, main like data processing and data storage. The cloud market is very segmented with many kinds of configurations, performance and location offerings. At different providers all of those might be called differently. The user wants to find the best solution for the problem with the lowest costs. As the market for Cloud computation is growing rapidly the prices of the services are changing fast. In addition, new offerings are coming up with higer performances. Because of this rapid changes the knowledge base of any decision support system for application migration needs to be kept up-to-date by steadily updating the database with new information from the different providers. In this way the user can find the exact solution in the database.

For the purposes of this work, we assume that the users are searching for their solutions in the Nefolog system database.[1] An interface that the users can directly use is necessary for this purpose. For getting a user friendly interface it is planed to create a dynamic web page where the user can easily find database entries. This should be possible without the user manually having to query the database for offerings.

1.2 Problem Definition and Goal

The user can give a request with it's needs to the system. The visualization system then takes the request and summarizes all the information from the providers. The system should then give an overview so that the user can find the best offering. In the end the system should calculate the costs resulting about using a solution. The Nefolog database which is used by the visualization contains the web services defined by providers and offerings. The offerings are segmented into service type and configuration.

1.3 Outline

This thesis consists of six chapters, the first chapter is introducing the main idea of this work. Chapter two introduces the technology background. Like the Figure1.1 shows, Chapter three to Chapter five present the specification of the system, the design and the details of the , as well as the evaluation of this work. The thesis ends with some conclusions and future work.

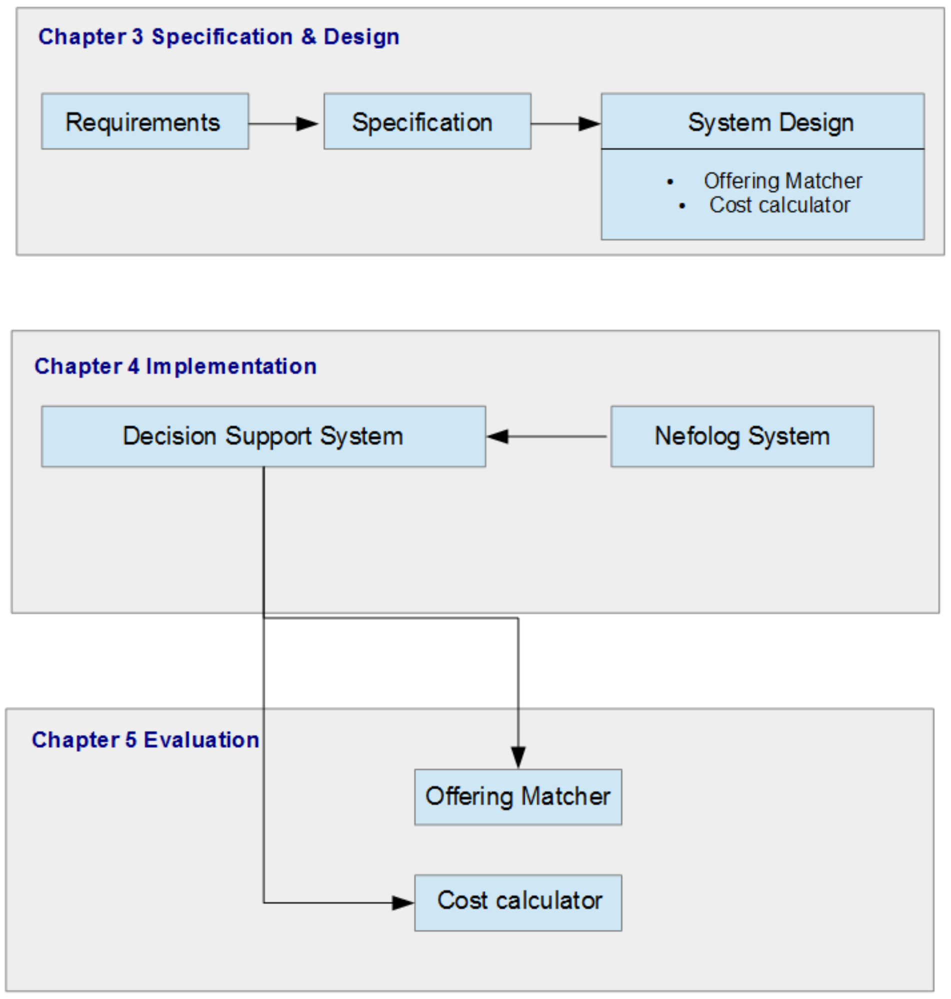


Figure 1.1: Structure of Thesis

2 Background

In this chapter will introduce some background about cloud computing, like what is cloud, the different types of cloud, three kinds of Cloud Service Models.

2.1 Fundamentals

Cloud computing portends a major change in how store information and run applications, instead of running programms and data on an individual desktop computer, everything is hosted in a nebulous assemblage of computers and servers accessed via the internet. [2] That is the integration of the development of Grid Computing, Distributed Computing, Parallel Computing, Network Storage Technologies. The core idea is to run a program on many connected computers at the same time. The user may be using a different own computing system. The user does not need to understand how the infrastructure is built like. The cloud offers the possibility to call programs or applications or to store data or doing other computing tasks through the internet.

The idea of cloud computing went through four phases at first the Power Plant model [2]:The idea of cloud computing was to have a cheap replacement for an uninterruptible power supply (UPS). The second phase was the Utility Computing: in the 1960s the computing devices were very expensive, not every company could afford them, the idea of utility computing should solve the problem. The server, storage system and application were integrated from different locations to share them to the users. The user could then pay dependent on usage. The idea of Grid Computing is to distribute a big computing prolem to small segments. These segments can then be calculated in parallel on devices with low compution capabilities.

2.2 Cloud Types

Currently Cloud-based applications provide a wide range of solutions to the user. The National Insitute of Standards and Technology (NIST) describes the cloud solutions based on the system's Deployment Models and the Cloud Service Models, as shown in Figure 2.1.

- Community Cloud: Two or more organizations or companies from the same branch share the private cloud of each other which only can be accessed by the members of the community.
- Hybrid Cloud: A Cloud that consists of two or more public, private or community clouds.

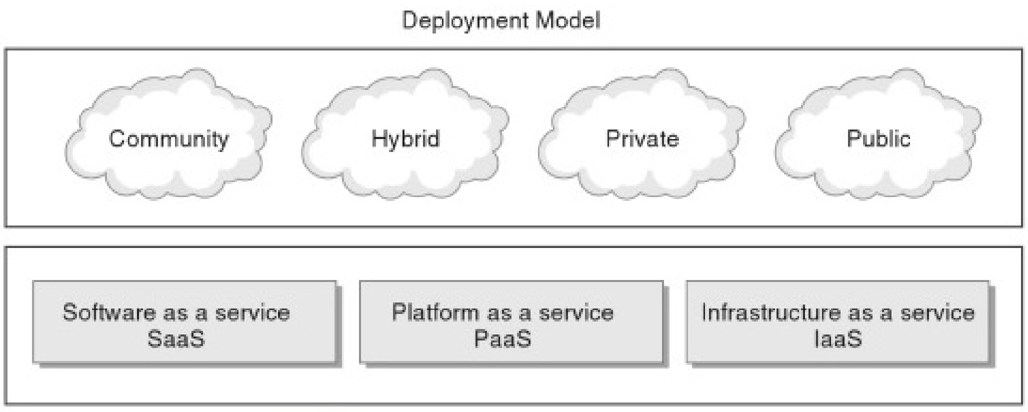

Deployment Model
Community
Hybrid
Private
Public
Software as a service SaaS
Platform as a service PaaS
Infrastructure as a service IaaS

Figure 2.1: Cloud Deployment Models [3]

- Public Cloud: Makes resources, such as applications and storage available to the general public over the internet. Using a public cloud is least intensive, even could be free, but that is less secure.
- Private Cloud: Owned by a specific entity and normally used only by that entity or one of its customers. A private cloud offers increased security at a greater cost. [3]

There are three kinds of Cloud Service Models available: Software as a Service (SaaS), the application for the user is web-based. Platform as a Service (PaaS) which provides a computing platform and a solution as service. A PaaS solution includes hardware, operating systems, development tools and administrative tools. Infrastructure as a service(IaaS) is the most basic cloud service model, the provider offers computers or virtual machines or storage. Those three models are independent, but also have some dependency relationships.

- Software as a Serice (SaaS): As the Figure 2.2 shows, SaaS is the most common and most earliest cloud computing service. Through the internet browser the user can directly connect with application on cloud. The provider offers to maintain the hardware and software of cloud. The cost is depend on the way the service is used and the amount of the users. For example like Google Apps, that included Gmail, Google Calendar, Google Docment, and other online-based office tools. The advantages of the SaaS solution for the user is that it is easy to use. When there is an internet connection available the user can access the Saas service at any time. The user does not need to care about the technology behind the service. Another advantage of this is safety of the storage and security of data transfers can be granted. This solution uses the basic web technologies like HTML, JavaScript and CSS.
- Platform as a Service (PaaS): The user creates the software, uses the tools and libraries from the cloud provider. The user also cares about controlling the software developing and configuration settings, but doesn't care the maintrainance

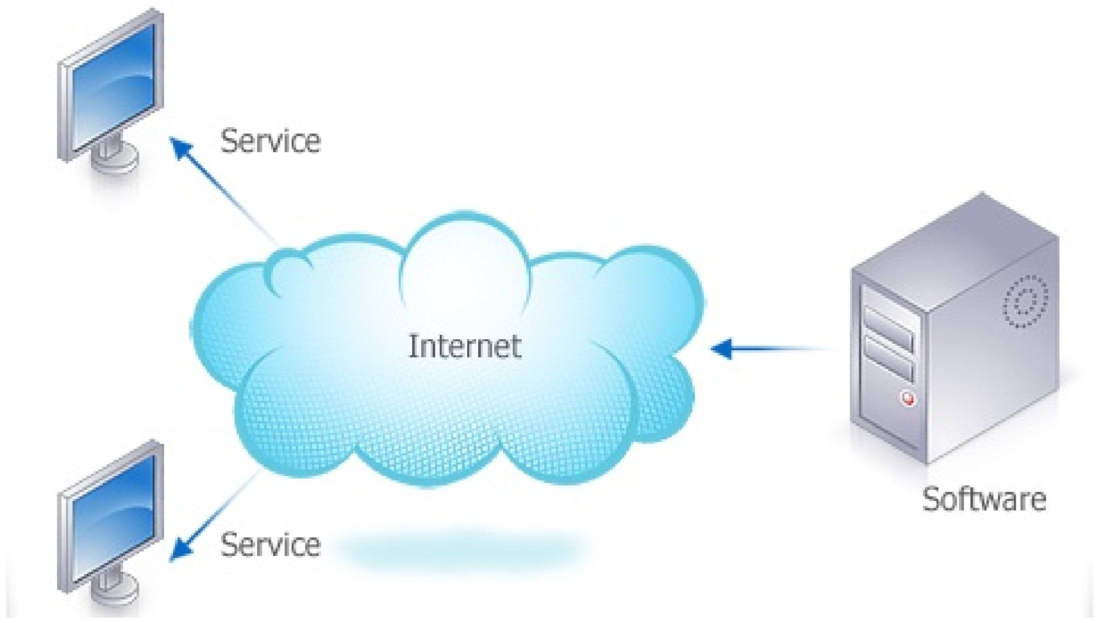


Figure 2.2: Software as a Serice (SaaS) [4]

of the servers, storage, and operating systems, like the Figure 2.3 shows. Examples for this are the Google App Engine, Windows Azure Plattform and Heroku. The Windows Azure Plattform is the PaaS resolution product from Microsoft, it provides a cloud services operating system and a set of services to support easy development and operation of applications for the platform. The platform provides the functionality to build and manage applications that span from consumer Web to enterprise scenarios. The solution under this are usally technologies like REST (Representational State Transfer), parallel processing and distributed cache.

- Infrastructure as a service(IaaS): In this case the user gets infrastructure as service. The provider offers a datacenter where the client gets one or more physical or virtual machines with storage and network connectivity. The user is then still responsible for installing and maintaining the software on the system. Dependent on the technology used this can be also a virtual machine, so that the acquisition cost will be reduced. The advantage of using this technology is that the IT infratructure can scale with the need of the customer. If the customer for instance needs more CPUs this can be simply realized by changing the configuration of the virtual machine. The same applies for the needed memory or mass data storage, like the Figure 2.4 shows. Examples for this are: Amazon EC2, IBM Blue Cloud, Cisco UCS and Joyent. Distributed storage system like Google BigTable can be used for mass data storage.

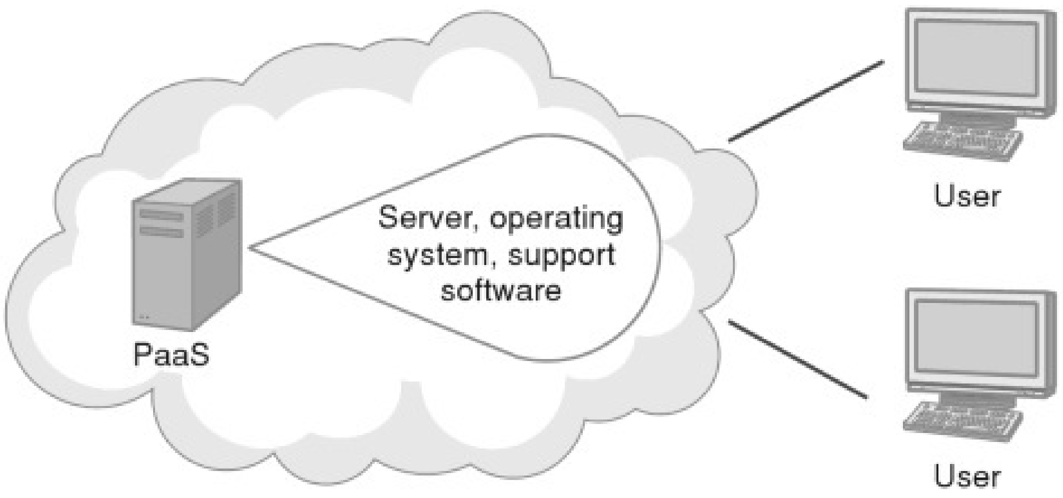


Figure 2.3: Platform as a Service (PaaS) [3]

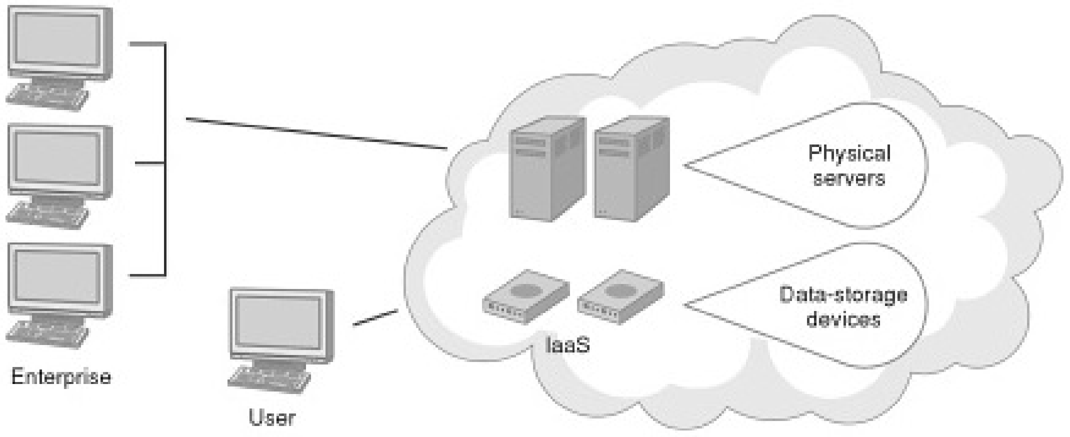


Figure 2.4: Infrastructure as a service(IaaS) [3]

2.3 Migration to the Cloud

2.3.1 Migration Type

As the research by Andrikopoulos et al. discusses[5], there are four types of migration:

- Type I: Replace component(s) with cloud offering: One or more components are replaced by cloud services, that is the least invasive type of migration. When switching to a cloud offering istead of a local component it is possibly necessary to rewrite some interfaces or doing other adaptations. For example when people use the Google App Engine Datastore instead of the local MySQL database.
- Type II: Partially migrate some of the application functionality to the Cloud: Those components are from one or more application layers and have an interconnection-relationship. For example instances to host data and business logic for HIC use Amazon SimpleDB and EC2.
- Type III: Whole software stack migration of the application to the cloud: that means to encapsulate the application of a whole software stack in the VMs on the cloud. That is also the most common type migration to the cloud.
- Type IV: Cloudify the application: The application is completely migrationed: The application is re-engineered as a composition of cloud services and runs exclusively in the cloud.

Besides to this also other decision systems like the CloudGenius and the Cloud Adoption Toolkit do exist. The CloudGenius decision system was developed by by Menzel et al. [6]. Regarding to Xiu [1] this is a continual evolutionary migration process. The Cloud Adoption Toolkit is another approach created by Khajeh-Hosseini et al. [7]. This toolkit includes a collection of tools which focus on both cost calculation and socio-technique. It is based on a checklist of questions.

2.3.2 Decision Support Systems

A DSS is an interactive, flexible and adaptable computer based information system that utilises decision rules, models, and model base coupled with a comprehensive database and the decision maker's own insights, leading to specific, implementable decisions in solving problems that would not be amenable to management science models per se. Thus, a DSS supports complex decision making and increases its effectiveness. [8]

The following five categories of DSS, there the Driven is the tool or component, that provide the domiant functionality in the Decision Support System.[9]

- Data-Driven DSS are the most common of those five category of the DSS. They are used for analysis of large amounts of structured data. The system provides operation with use the internal company data and some external data, like for example the data warehousing, analytical systems.

- Model-Driven DSS. A system that is using some kind of model like a financial model, an optimization model or a representational model. Model-Driven DSS emphasize access and manipulation of the model.
- Knowledge-Diven DSS, can suggest or recommend actions to the managers, and are person-computer systems with specialized problem-solving expertise. There the expertise like kind of to understand the problem from a particular domain and to solve the problem.
- Document-Driven DSS, these DSS is focused on helping managers to retrieve and manipulate unstructured information or documents from different electronic formats. Those DSS can be divided in oral, video and written.
- Communications-Driven and Group DSS, pays attention on communication get collaboration and coordination between groups of people, enables share the information and support the decision tasks from the group. For example the audio conferencing is supported from the group.

When the user wants to find the appropriate offering from the system, there will be some kinds of decisions, that they should be made. Between the different decisions, the user also need take care about the relationships and the influences. The research by Andrikopoulos et al [10] introduced the basic decisions that need to be made when migrating an application to the cloud. Like the Figure 2.5 shows, there is a model of decision support system, desision and task with different actions, the decisions (distribute application, select service provider/offering, defne multi-tenancy requirements and defne elasticity strategy). Those decisions are supported by seven different tasks (work load profling, compliance assurance, identifcation of security concerns, identifcation of acceptable QoS levels, performance prediction, cost analysis and effort estimation). The transparent arrows show the influences between them. For example in this figure that the performance prediction can descide the select service provider/offering, and effort estimation can also send feedbacks to the select service provider/offering.

2.4 The Nefolog Decision Support System

Like the Figure 2.6 shows, there are two parts, one is the web application and other is the Nefolog system which developed by Mingzhu Xiu in 2014 [1]. She built the web services and the database, implemented the offerings matcher, and costs calculator. The aim of the system is to help the customer to move components of applications to the cloud. It helps to identify a suitable cloud provider service. In the end the Nefolog system retrieves the values for the candidate offerings search service and offerings cost calculation service from the database which is connected to the Nefolog system.

2.4.1 Database of Nefolog system

All of the functions and database in the Nefolog system are designed based on the system MDSS [11], like this Entity-Relation Figure 2.7 shows. The Nefolog database

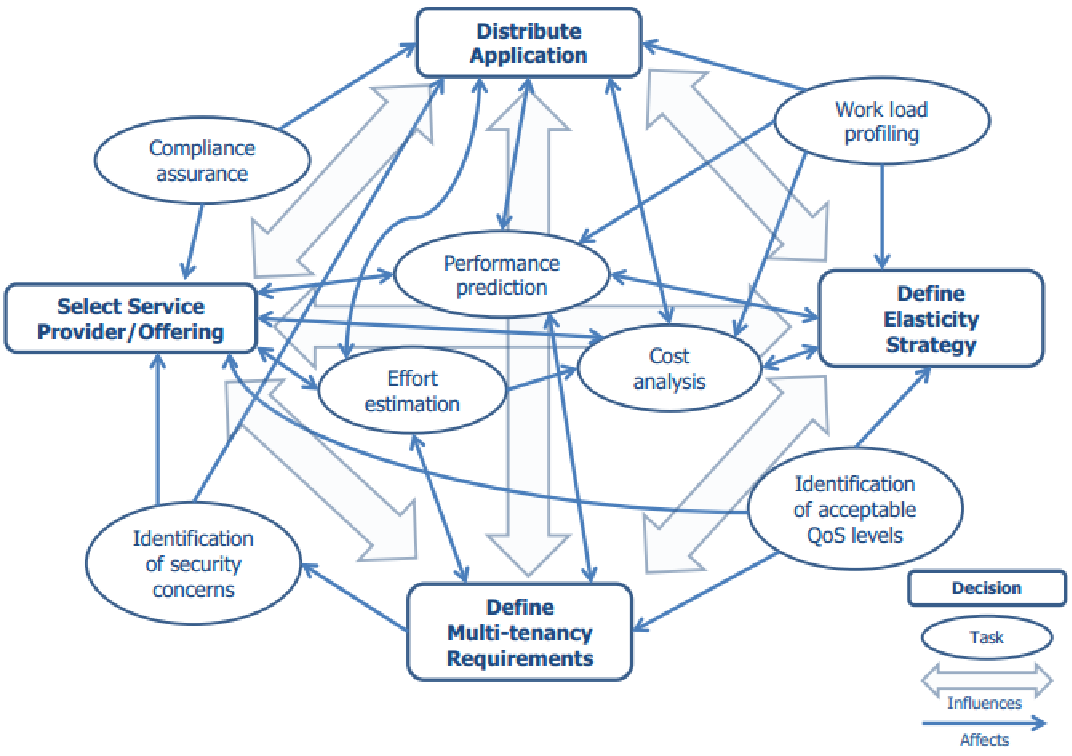

Distribute Application
Work load profiling
Compliance assurance
Performance prediction
Select Service Provider/Offering
Define Elasticity Strategy
Cost analysis
Effort estimation
Identification of acceptable QoS levels
Identification of security concerns
Define Multi-tenancy Requirements
Decision
Task
Influences
Affects

Figure 2.5: Conceptual Model of Decision Support System for Cloud Migration[10]

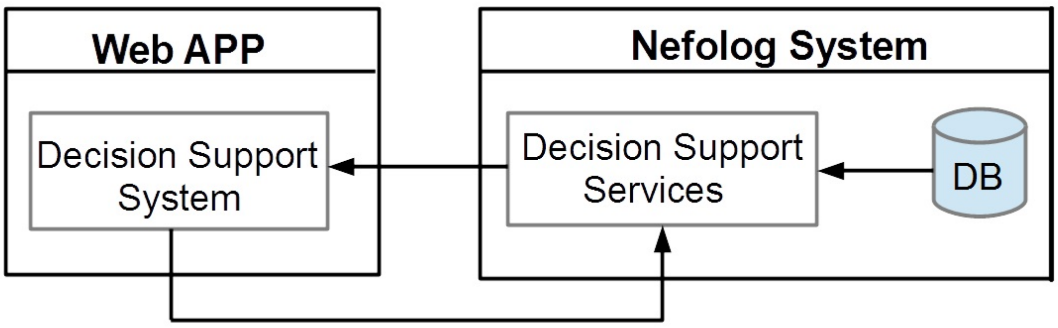


Figure 2.6: Nefolog system

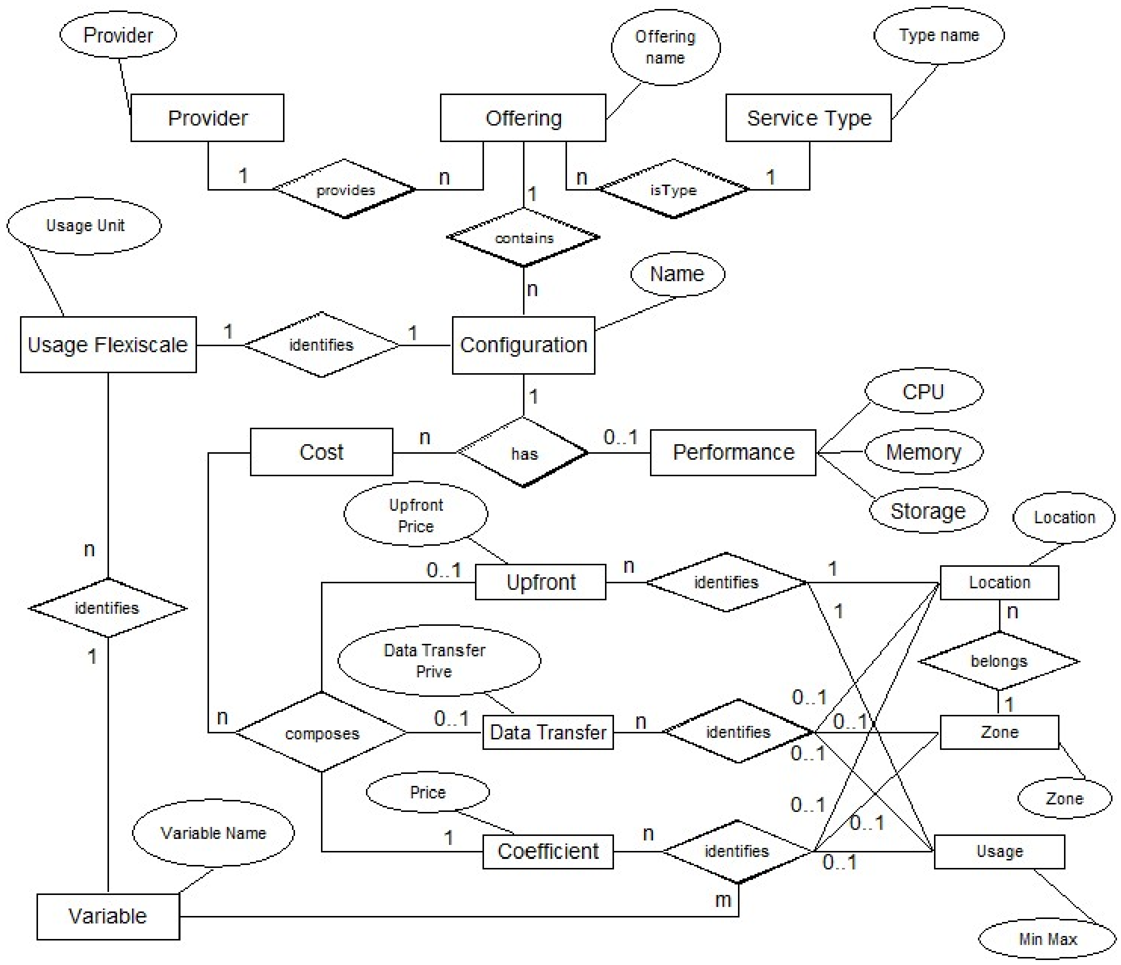


Figure 2.7: Entity-Relation Diagram of Nefolog System[11]

contains the data of six providers: Windows Azure, Amazon Web Services, Goolge, Hp Cloud, Flexiscale and Rackspace. All of those providers offer multiple kinds of offerings, which can be classifed in different 25 service types. One offering from one provider can have multiple configurations, which are sorted by kinds of performance. For example the Hp Cloud has offerings like HP Cloud Compute (Standard Instance Type), with different configurations like for standard extra small, standard small, standard medium and so on. The performace of the extra small cloud compute is with 1 HP computation unit, one virtual core, 1GB RAM, and 20GB of disk space. The customer can according to the self demand choose the configuration with compatible performance. The cost of configuration is calculated by each calculation formula, different location, and data transfer. The Nefolog database has a total 68 offerings, 547 performances, 656 configurations, 15 locations, 4890 different calculation methods (coefficient).

2.4.2 Decision Support Services

The advantage of the Web services is that they are platfom and program language-independent, the user doesn't need to worry about which kind of the platform or program language was used, either Windows or Linux, either C or Java. All of the resources from the database will be indentified by the URIs. The client side sends their request through the query with URI to the server side. The server side does for instance a calculation and sends the result back in standard representations like XML or JSON. In the Nefolog system there are two important services: The candidate search service and the cost calcultor service. Furthermore there are commands allowing to request the database content like the available providers etc.

- Candidate Search Service The candidate search service is like a function of the offering matcher. There are nonumerical performance characteristics and numerical performance characteristics. Non numerical performance characteristics are used for the operating system, the software license and IO. The other performance characteristics are stored as numerical performance characteristics. Before the user continues to the cost calculation service the information retrieved by the function can be evaluated by the user. [1]
- Cost Calculator Service The cost calculations service consists of two parts. There is a static calculator and a dynamic calculator available. The difference between the static calculator and the dynamic calculator is that the dynamic calculator needs additional logic. The total costs include the usage amout of costs, the data transfer costs and the upfront costs. It also needs to be checked whether they are served by the same data center, if not there could be a difference in the calculation. [1]

2.4.3 Basic knowledge for the web technology

The following list of web technologies are nowadays very commonly used for the development of interactive web pages. This is why they are also used for the implementation of this project:

- JavaScript JavaScript is a programming language built into the web browser, that is one of the best ways to add interactivity to website, because it's the only cross-browser language that works directly with web browsers[12].
- JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate [13].
- Cascading style sheets CSS is a modern standard for the website presentation, it can combined with a structural markup language like HTML, and XML. The cascading style sheets provide Internet browsers with the specified visual structure format [14].

2.5 Summary

This chapter introduced some basic knowledge for the rest of this work like some cloud types, migration types and also the Nefolog system. This background information of the Nefolog system is the basic knowledge for this new work, because this new system will be implemented as a web application based on Nefolog. In order to work it needs connection to the web services of the Nefolog system, which does the calculation and accesses the data from the database. The next chapter will explain the new web application with requirements and specifications.

3 Specification & Design

This chapter identifies the requirements for a decision support system extending the Nefolog system to a requirements specification and design specification document.

3.1 Project overview

The Nefolog system consists of a database as well as a java server page offering web services to access the data. It contains cloud computing providers with their offered services and prices. The Nefolog system also contains a cost calculator which can compare the price dependent on the services needed and which provider is used. The existing Nefolog system is the base for a decision support system whose graphical user interface which shall be described here in this document. It shall offer a user friedly access to the existing providers, the available services and the resulting prices.

3.2 Requirements

To realize this project the following requirements should be used. It is didived into functional and non functional requirements. The functional requirements shall describe what functionalities should be offered by the system. The non- functional requirements should describe the environmental requirements for the project e.g. the target operating system.

3.2.1 Non-Functional Requirements

The non-functional requirements shown in Table 3.1 have been defined for this project. HTML, javascript and CSS sheets should be used for the design of web application. The user can access the Nefolog database dynamically. The web presentation of the information is user-friendly.

3.2.2 Functional Requirements

Table 3.2 shows the functional requirements for the project. The visual Navigation of the database will go through the available providers or service types. The access should use the available RESTful API of the Nefolog system. A human oriented interaction with the offerings matcher and a cost calculator shall be built, so that the user can get the offering or price of offering efficiently.

Table 3.1: Non functional requirements

Requirement ID	Title	Description
NFR1	Web based implementation	The implementation of the system shall be done using Javascript, HTML and CSS sheets.
NFR2	User interface	The user interface should be implemented allowing the user to dynamically access date entries in the Nefolog database. This should be possible by using a web browser.
NFR3	Necessary user environment	In order to run the user interface for the user a browser needs to be available as well as a network connection to the server, or can be installed locally.
NFR4	User friendly presentation	The web based presentation of the information retrieved by the Nefolog shall be provided in a user-friendly representation.

Table 3.2: Functional requirements

Requirement ID	Title	Description
FR1	A Web GUI shall be available which provides dynamic interaction with the Nefolog system	Navigation through the available providers, service types etc. shall be possible. The access should build on the RESTful API provided by Nefolog.
FR2	Interaction with offerings matcher and cost calculator	Human oriented interaction with the the offerings matcher and cost calculator services shall be possible.

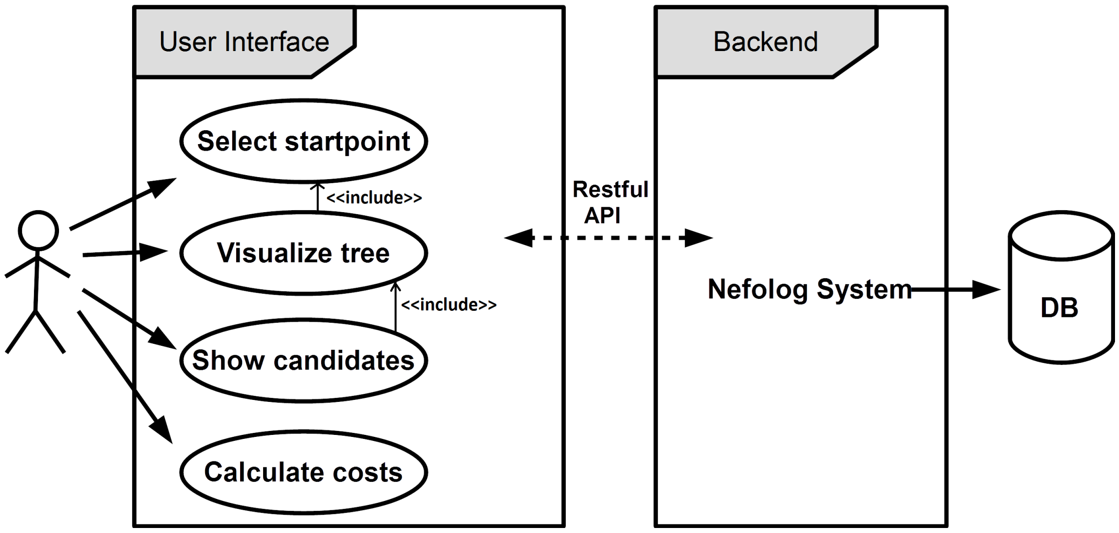


Figure 3.1: The Use Case Diagram

3.3 System Specification

3.3.1 Use Case Diagram

Figure 3.1 shows the use case diagram of the project. The right side of the diagram shows the Nefolog system which accesses the database. The Nefolog system itself is accessed via restful communication. Because this project does not focus on the functionality of the Nefolog system the right side of the figure is intentionally pictured without details. The left side of the diagram shows the interaction of the system with the user. The user can use the following features:

- Select a startpoint for browsing the Nefolog database
- Visualize the information in the shape of a tree, dependent on the startpoint
- Show the candidates matching the selected features
- Calculate the costs resulting on chosen specific provider with defined features

3.3.2 The graphical user interface

The graphical user interface is the main feature for this project. Therefore a user-friedly representation of the system is essential. It should enable to user to use the system without having to read a manual before. The user interface for this project shall be implemented as a lightweight web interface. This means that the calculation intensive operations should be running on the server. The communication to the client is based on TCP/IP as low level transport layer. Therefore the client can run on the same machine as as server but it can easily also run on another machine which has a network connection

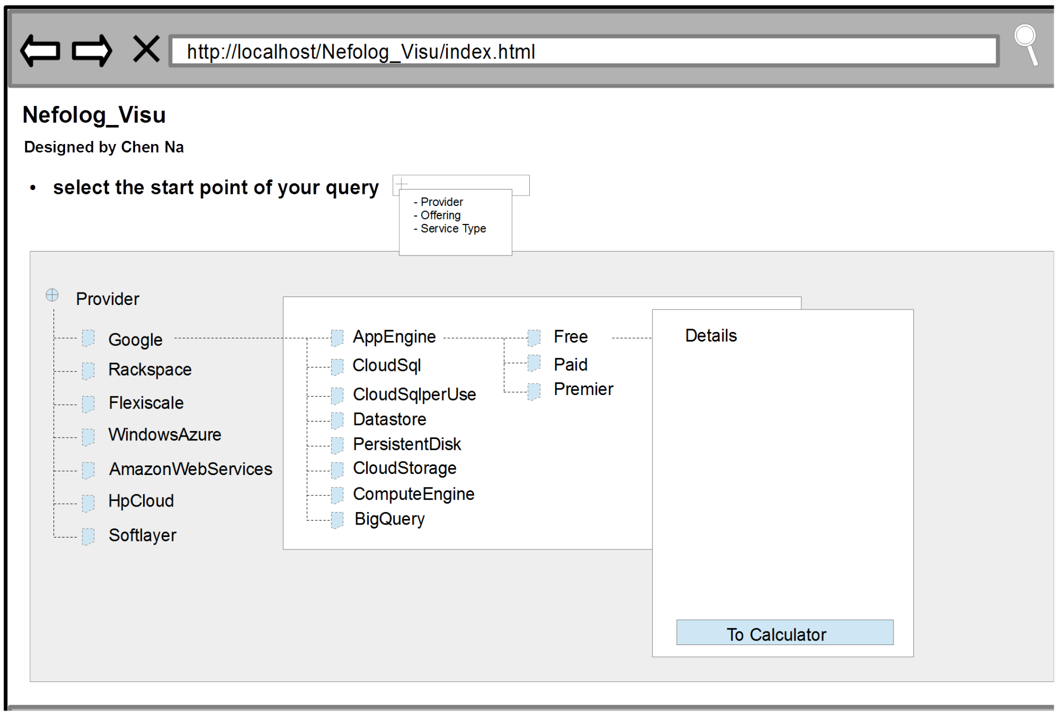


Figure 3.2: Mockup - Provider View

to the server. The protocol above is HTML, therefore no special application needs to be installed on the client computer. The advantage of this is also that this enables a cross platform usage. You can also use a client running any operation system as long as it offers a web browser supporting HTML5 and Java script. It would also be possible to render everything at the server and don't use a script language at the client but this would enhance the developlent effort of the system and would also lead to a higher latency in case of a slow network connection. The latency describes the reaction time between a user input and the response of the system. If this time is too long this has a negative effect on the user experience.

Provider View

The intention of the main screen is to provide an interface for the user which allows to easily search the database for providers. Afterwards it should be possible to trigger a cost calculation.

The start point for browsing for providers is shown in Figure 3.2. The user first can choose the startpoint for the intended search. This can be a provider an offering or a search type. Depending on the choice a tree is built. The advantage of a tree is that it gives a good top down overview. The user only can see the information based on the

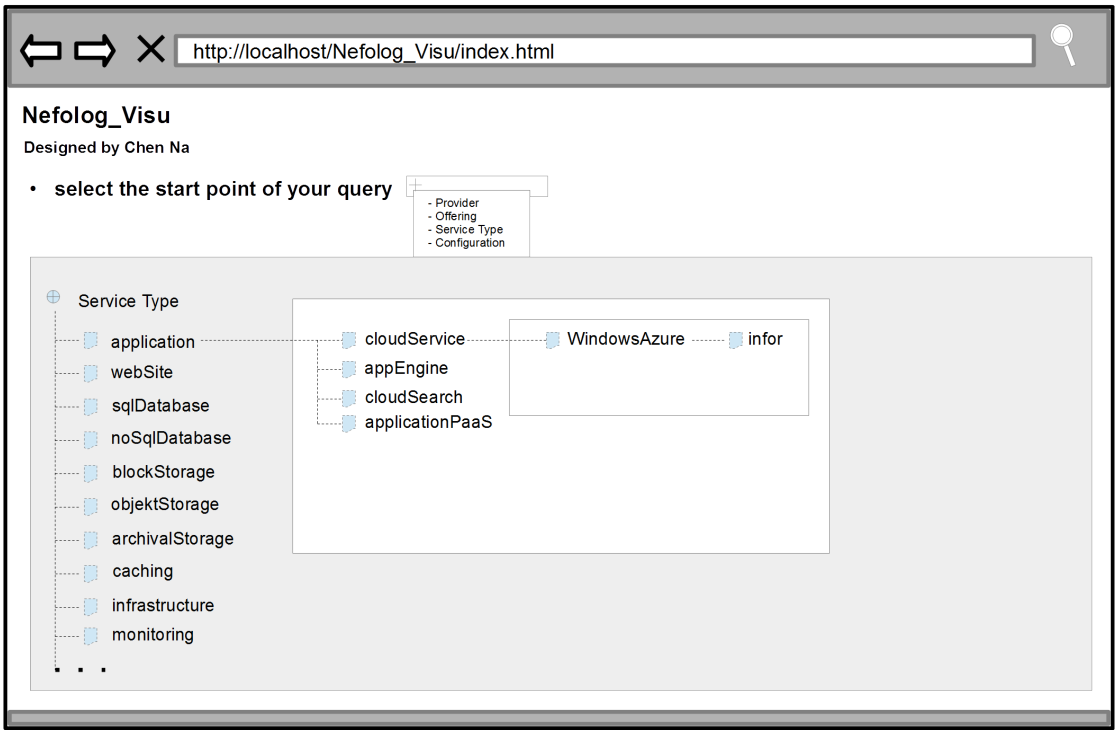


Figure 3.3: Mockup - Service Type View

choice which was done before. From the performance point of perspective it offers the possibility that the database query is done step by step as the user refines his choice. Alternativerly to that it is also possible to do a general database query and give the java script tree already all information.

In this screen the user firstly selected to start browsing from the providers. Therefor a tree spans with all providers listed in the database. Then the user can click on a provider. If different price models are available the user can choose one. From there it is then possible to calculate the price for the desired services.

Service Type View

The main screen in Figure 3.3 offers the selection of the start point for queries. This screen is active if "Service Type" was selected as a start point.

The service type start point is useful if a user already knows what service type is needed e.g. Application server, web server... etc. The user can choose one and then further refine the decision by choosing an offering. The result of the query is then a list of providers offering what is needed. When this choice is done it is possible again to calculate the costs.

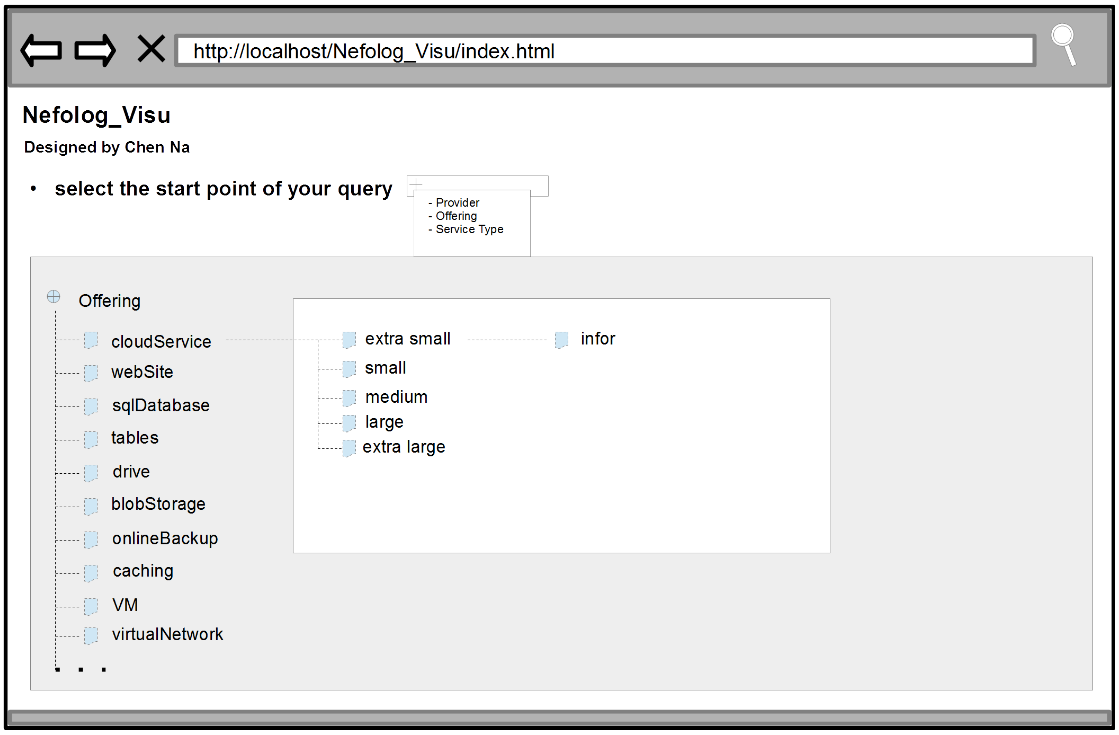


Figure 3.4: Mockup - Offering View

Offering View

The screen in Figure 3.4 is active if "Offering" was selected as a start point.

This view should be chosen if the needed offering is the base of the request. Offering means a specific cloud service, web site, sql database etc offered by a cloud provider, e.g. Amazon Web Services' Elastic Cloud Computing (EC2) solution. In the next step the query can be refined and then the calculation of the costs can be triggered.

Cost Calculation View

Choosing a provider from the main screen is possible in the different view modes mentioned above. Independent from which screen a provider was chosen it is possible to calculate the costs with the cost calculator screen (See Figure 3.5).

In order to make possible to calculate the costs it is necessary to enter more details in this screen. These are information like the number of CPU cores, the CPU speed, etc. From the implementation aspect it would be the easiest way to offer text fields where the user should enter the details. From the user point of view entering information in a text box is uncomfortable. This is the case even more because there are more and more computers in usage which don't have a keyboard any more like tablets and smart phones. Another disadvantage of a text box is that the user could enter anything there which might not be intendet. In this case a message needs to be shown to the user telling what has to be entered into the text box.

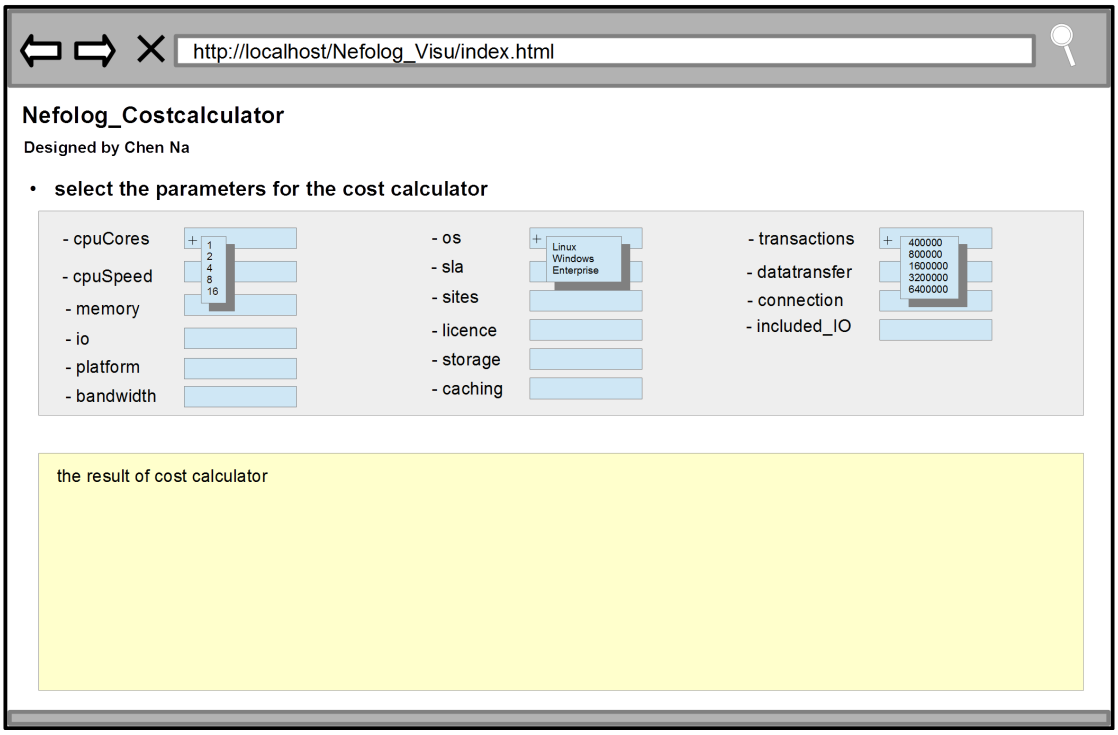
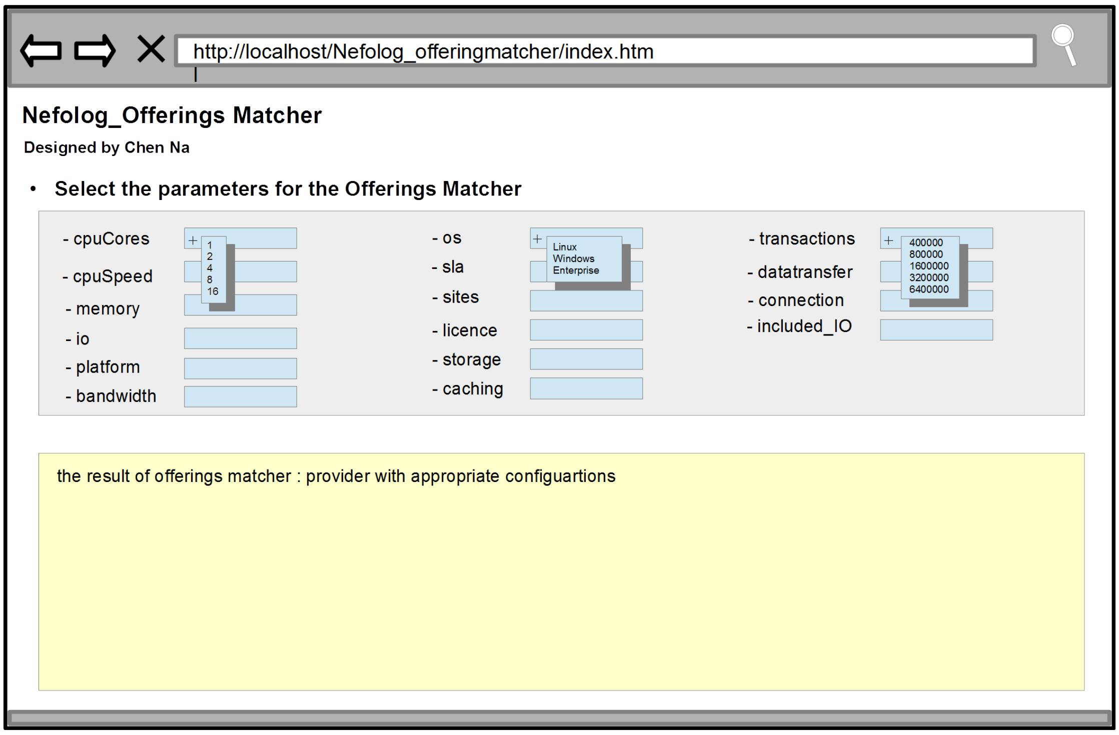


Figure 3.5: Mockup - Cost Calculation View

To make the entering of information as easy to use as possible one solution would be a drop down menu.

Offerings Matcher View

As Figure 3.6 shows, the user should give some parameters about the required performance, like CPU cores, memory, bandwidth and so on. Through the candidate search service in the Nefolg system the inappropriate configurations will be filtered out and the configuration IDs of the appropiate providers will be returned. There are parameters with numerical parameters and non-numerical parameters. Both searches for parameters will be built with the drop down boxes, because for the selection of non-numerical parameters can be more difficult, if the user writes the parameters different like it is called in the database. For instance if a user is searching for the string "Linux" but the String "Ubuntu" is stored in the database this could not be found with a query. In opposite to that it works if the user is choosing from a drop down box with the entries of the database.


http://localhost/Nefolog_offeringmatcher/index.htm

Nefolog_Offerings Matcher

Designed by Chen Na

- Select the parameters for the Offerings Matcher

the result of offerings matcher : provider with appropriate configuartions

Figure 3.6: Mockup - Offering Matcher View

3.4 Summary

In this chapter we designed the functional and non functional requirements. The functional requirement describe for example a web GUI shall be available which provides dynamic interaction with the Nefolog system. The non functional requirements describe which kind of technology will be used for implementation. In addition, a design of the system was discussed through mockups.

4 Implementation

This chapter describes the implementation and design of the graphical user interface. The chapter consists of three parts: one is the graphical user interface, based on Nefolog. The other two are the candidate search view and cost calculator, which use the RESTful APIs offered by Nefolog system.

4.1 Communication to Nefolog

As the requirements on chapter 3 Table 3.2 defined the communication to the Nefolog System shall be done using Restful web technology. This means that the communication to the server is done like the communication from a web browser to a web server. The client sends a HTTP request to the server and the server then returns the requested data in a representation format that is understood by the client. This could be XML or JSON for example. The Nefolog system returns the results as XML code. The advantage of this is that the communication is completely platform independent. The client does not need to know what operating system the server is running. Even the programing language used for writing the web service can be different from the one which is used at the client side.

The Nefolog system offers web services (See Table 4.1) for requesting information. These were used to build the GUI servlet.

4.2 The graphical user interface

It is the aim of a graphical user interface to build an easy to user interface between the user and the system. For this project a web based user interface was chosen using HTML, Javascript and CSS. Javascript is only used for smooth user interactions. Most logic is kept at a Java server page. The idea of this is to only have a low dependency of the used browser version and features. The Restful communication is going on between the gui-servlet and the Nefolog system. The reason for this is that the gui-servlet and Nefolog system are most probably running on the same server or at least having a high bandwith connection between each other. The connection speed to the browser is then not so significant for the reaction time of the system.

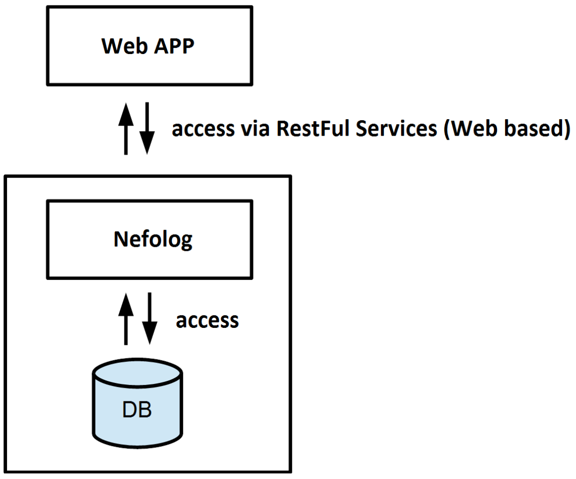


Figure 4.1: Restful Communication

Table 4.1: URL Table

URL	Description
`/nefolog/providers`	Returns the list of providers which are stored in the database.
`/nefolog/providers/{Provider}`	Returns the list of offerings and service types which are supported by a specific provider
`/nefolog/offerings/{Offering}`	Returns the list of configurations which do support the requested offering
`/nefolog/offerings/{Offering}/{Configuration_XYZ}`	Returns the supported performances covered by a given configuration like the number of available CPUs etc.
`/nefolog/costCalculator?configid={ConfigID}`	Returns the possible parameters for the selected configuration
`../costCalculator?configid={ConfigID}?{parN}=x&...`	Returns the calculated costs for the selected configuration

4.2.1 Building trees

In order to give a good overview of the available content of the Nefolog system a tree scheme was chosen. The user can first select where to start (Providers or Service Types), then a tree is built out of the data retrieved by the Nefolog system. The implementation of this is built on the XML scheme which always contains a link to following nodes as well as on a function written by me which does an HTTP request and returns the retrieved XML file as a list of nodes. The network communication and parsing is done centrally at this globally used function. The function is named getRestDataFromURL (See Listing 4.1). It expects one parameter which delivers the URL needed to request and a XML tag which is used to parse the XML data returned by the service.

Listing 4.1: Function getRestDataFromURL

```java
NodeList getRestDataFromURL(String url, String tag)
{
    CloseableHttpClient inst = null;
    CloseableHttpResponse resp = null;
    String result = "";
    NodeList nList = null;

    try {
        inst = HttpClientBuilder.create().build();
        resp = inst.execute(new HttpGet(url));
    }   catch (IOException ie) {
        ie.printStackTrace();
    }

    if (resp == null) {
        //out.write("Error connecting to url");
        return null;
    }

    try {
        final HttpEntity entity = resp.getEntity();
        if (entity != null) {
            result = EntityUtils.toString(entity);
        }

        resp.close();
    }
    catch (IOException ie) {
        ie.printStackTrace();
    }

    // Parse
    try {
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.
             newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

        Document doc = dBuilder.parse(new ByteArrayInputStream(result.
             getBytes(StandardCharsets.UTF_8)));

        doc.getDocumentElement().normalize();
        nList = doc.getElementsByTagName(tag);
    }
    catch (Exception ie) {
        ie.printStackTrace();
    }

    return nList;
}
```

From line 3 to line 40 of the code an HTML connection to the server is being established and the resulting XML data from the server is stored in a buffer. Line 33 to 44 are using the integrated DOM based XML parsing capabilities of JAVA. Using DOM parsing a XML document is nearly done with one function call. The information is then already available as a tree and can be easily processed further on. Another approach for parsing the XML files would have been SAX. In opposite to DOM it is not parsing the whole XML file as a block but it is calling functions while the XML data is beeing parsed. This also allows parsing of big amounts of data but makes the processing more complicated. The disadvantage of using DOM is that the complete XML data is processed and stored in the memory at once. Therefore it is not optimal to be used with huge amout of data. For the size of the single XML data blocks which have to be processed in this project it is a suitable solution.

Technical details

To build the tree starting from providers the first step is to do a REST request to the Nefolog system asking for the available providers. This is done by sending an HTTP GET request:
`http://localhost:8080/nefolog/providers` The Nefolog system is then answering with the following block of XML data:

Listing 4.2: http://localhost:8080/nefolog/providers

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<resource>
    <name>providers</name>
    <content>
        <provider>
            <name>Google</name>
            <uri>/providers/Google</uri>
        </provider>
    ...
    ...
    </content>
</resource>
```

The providers are specified with a name and an URI. This URI then leads to offerings that the chosen provider can do. In order to jump to the offerings this link has to be called and the provided content has to be parsed again:

Listing 4.3: http://localhost:8080/nefolog/providers/Google

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<provider>
    <name>Google</name>
    <content>
        <offering>
            <name>AppEngine</name>
            <uri>/offerings/AppEngine</uri>
            <servicetype>
                <name>application</name>
                <uri>/serviceTypes/applications</uri>
            </servicetype>
    ...
    ...
    </content>
</provider>
```

To get the available configurations starting from the previous point it is necessary to follow the URI specified in the <offering>section:

Listing 4.4: http://localhost:8080/nefolog/offerings/AppEngine

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<offering>
    <name>AppEngine</name>
    <content>
        <configuration>
            <name>free</name>
            <uri>/offerings/AppEngine/configuration_32</uri>
        </configuration>
    ...
    ...
    </content>
</offering>
```

To be able to render the content in the performance section of the window the URL given in the <configuration>section can be called and parsed. Here one can find parameters like the number of available CPUs etc.

Listing 4.5: http://localhost:8080/nefolog/offerings/AppEngine/configuration_33

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<resource>
    <name>paid</name>
    <configid>33</configid>
    <content>
        <performance>
            <name>sla</name>
            <value>0.99950000000000006</value>
        </performance>
    </content>
</resource>
```

In order to present the user a tree on the screen a the freely available Javascript package dtree was used. With this code a tree gets built by doing some initialization and then

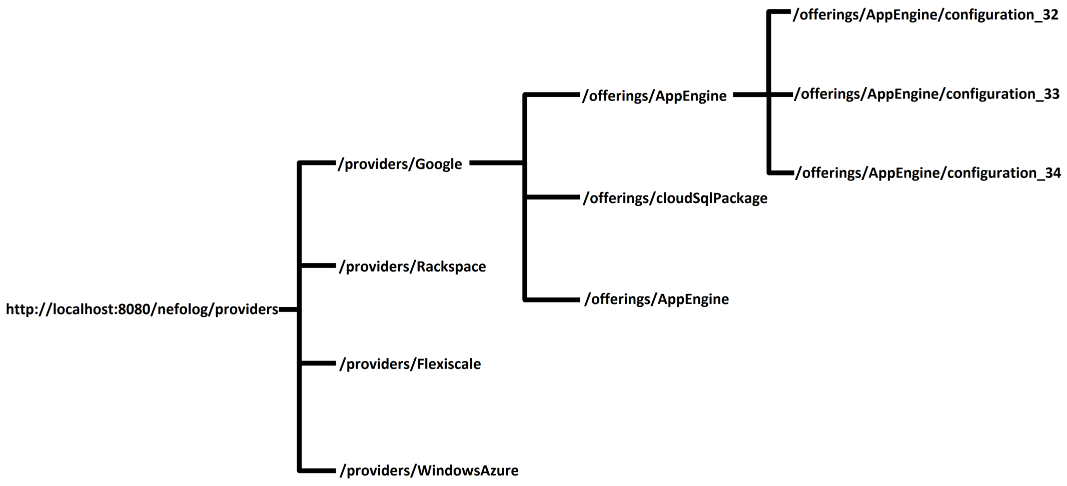

http://localhost:8080/nefolog/providers
/providers/Google
/providers/Rackspace
/providers/Flexiscale
/providers/WindowsAzure
/offerings/AppEngine
/offerings/cloudSqlPackage
/offerings/AppEngine
/offerings/AppEngine/configuration_32
/offerings/AppEngine/configuration_33
/offerings/AppEngine/configuration_34

Figure 4.2: Tree starting from providers

calling [add.(Unique ID, Subnode_of)]. When the tree is built it can be drawn. Therefore the JSP page goes recursably threw the whole tree in order to read the text fields and the links necessary for building the tree. Figure 4.2 shows a fraction of the links needed to build a visualisation tree.

Listing 4.6 shows the source code of the java server page building the tree starting from providers. This code implements the communication flow described earlier in this section. As is can be seen the code contains three nested for loops. Dependent on the number of database entries this takes processing time. The advantage of this is that this is only done once when the page is called. Then everything is available as Javascript code in the browser which leads to quick response times even with a slow network connection.

Listing 4.6: source code the tree from provider

```java
if (startpoint == 0) // Provider as startpoint
{
    // Get providers via Rest
    NodeList pList = getRestDataFromURL(PROVIDER_URL, "provider");

    for (int i = 0; i < pList.getLength(); i++) {
        Node pNode = pList.item(i);
        if (pNode.getNodeType() == Node.ELEMENT_NODE) {
            Element pElement = (Element) pNode;

            lastpNode = listID;
            out.println("d.add("+(listID)+",0,'"+pElement.
                getElementsByTagName("name").item(0).getTextContent()+"
                ','');");
            // Increase ID for next tree entry
            listID++;

            NodeList oList = getRestDataFromURL(BASE_URL+pElement.
                getElementsByTagName("uri").item(0).getTextContent(), "
```

```
                    offering");
            for (int i2 = 0; i2 < oList.getLength(); i2++) {
                Node oNode = oList.item(i2);
                if (oNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element oElement = (Element) oNode;
                    lastcNode = listID;
                    out.println("d.add("+(listID)+","+(lastpNode)+",'"+
                         oElement.getElementsByTagName("name").item(0).
                         getTextContent()+"','');");
                    // Increase ID for next tree entry
                    listID++;

                    NodeList cList = getRestDataFromURL(BASE_URL+oElement
                         .getElementsByTagName("uri").item(0).
                         getTextContent(), "configuration");
                    for (int i3 = 0; i3 < cList.getLength(); i3++) {
                        Node cNode = cList.item(i3);
                        if (cNode.getNodeType() == Node.ELEMENT_NODE) {
                            Element cElement = (Element) cNode;
                            out.println("d.add("+(listID)+","+(lastcNode)
                                 +",'"+cElement.getElementsByTagName("name
                                 ").item(0).getTextContent()+"','"+
                                 GET_PERFORMANCE_URL+BASE_URL+cElement.
                                 getElementsByTagName("uri").item(0).
                                 getTextContent()+"');");
                            // Increase ID for next tree entry
                            listID++;
                        }
                    }

                }
            }
        }
    }
}
```

Listing 4.7: Code block to fill the performance block

```
// Get performance from URL
String performanceURL = request.getParameter("perfID");
if (performanceURL != null)
{
    NodeList nList = getRestDataFromURL(performanceURL, "performance");
    for (int temp = 0; temp < nList.getLength(); temp++) {
        Node nNode = nList.item(temp);
        if (nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;
            out.println("name : " + eElement.getElementsByTagName("name")
                 .item(0).getTextContent());
            out.write("<br>");
            out.println("value : " + eElement.getElementsByTagName("value
                 ").item(0).getTextContent());
            out.write("<br><br>");
        }
    }
}
```

After having parsed the tree up to to configuration nodes it is possible to read the supported functions out of the Nefolog system. These are then shown in the HTML block displaying the performances of the selected configuration.

Listing 4.8: Code block to get necessary parameters for cost calculator

```
List<String> parameters = new ArrayList<String>();

NodeList pList = getRestDataFromURL(COST_CALC_URL+configID, "content");
for (int i = 0; i < pList.getLength(); i++) {
    Node pNode = pList.item(i);
    if (pNode.getNodeType() == Node.ELEMENT_NODE) {
        Element pElement = (Element) pNode;

        int i2=0;
        String result = null;
        while(pElement.getElementsByTagName("variable").item(i2) !=
            null)
    {
      String parameter=pElement.getElementsByTagName("variable").item(
          i2).getTextContent();

      // Store value in array list for building the request to the
          calculator
                   parameters.add(parameter);
      i2++;
    }
  }
}
```

The visualization created in this work is based on Java Server Pages creating and using HTML, Javascript and CSS code. CSS was used for changing the appearance of the Javascript tree as well as for dividing the screen into different view. Listing 4.9 shows one CSS file that was used. For instance of the thinkness of the borders needs to be adapted this can be done easily by changing the value in the CSS file. As it can be seen absolut coordinates have been used for implementing the user interface. In order to support a wider range of screen resolutions it would also make sense to dynamically divide the screen into sections.

Listing 4.9: css structure

```
\label{listing_visu_css}
/*------------------------------------------------|
| Nefolog visualization                              |
|-------------------------------------------------|
| Copyright (c) 2014 Chen                         |
|-------------------------------------------------*/

#Performance {
   position:absolute;
   top:130px;
   left:800px;
   width:420px;
   padding:10px;
   margin:0px;
   border:7px solid #0000EE;
}
```

In order to do calculations different parameters are necessary for different configurations. To know these parameters it takes one request to the Nefolog system with the configuration ID as parameter. The Nefolog system is then returning the possible parameters for this configuration. Since later this parameter list is also needed for other purposes the result of the query is stored in a list called parameters.

Listing 4.10: Dynamical building of the input form for the calculator

```
<form action "" method="get">
 <br>
 configID: <input type "text" name="configID" value="<%out.print(configID
      );%>" readonly><br>
<%
 for (String current: parameters) {
    out.println(current+": "+"<input type=\"text\" name=\""+current+"\"><
        br>");
    }
%>
<button type="submit" value="Submit">Submit</button>
</form>
```

Since the name of parameters as well as the number of parameters can be different in each configuration it is necessary to build the HTML user interface dynamically. The code in Listing 4.10 shows how this was done. Like all of the code of the visualization one can see here JSP code building HTML / Javascript code.

Listing 4.11: Build URL to Calculator and get the result

```

    String calcURL = COST_CALC_URL;
    String paramN = "";

    calcURL += request.getParameter("configID");

    for (String current: parameters) {
      if(current != null)
    {
        paramN = request.getParameter(current);
      if(paramN != null)
    {
          if ((paramN.contains("null") == false) && (paramN != ""))
        {
            calcURL += ("&"+current+"="+paramN);
        }
    }
    }
    }

    //out.print(calcURL);

    NodeList rList = getRestDataFromURL(calcURL, "result");

    for (int i = 0; i < rList.getLength(); i++) {
        Node rNode = rList.item(i);

        if (rNode.getNodeType() == Node.ELEMENT_NODE) {
            Element rElement = (Element) rNode;
      out.print(rElement.getTextContent());
    }
  }
```

To be able to call the Nefolog cost calculation service is is helpful to use again the previously stored list of parameters specific to this function. This function is building dynamically a request URL dependent on the parameter values entered in the browser. If a field is empty this parameter is not included in the URL. This enables the user for instance to let the Nefolog system calculate the costs also when the user does not already know for what region. The Nefolog system then simply returns the costs for all regions. With this information the user can then do a query using more field in order to get less results.

4.2.2 Candidate search view

The candidate search view view works similar as the calculation view. The main challenge of the implementation of the candidate search view is the fact that the parameters which are used for the calculation can differ dependent on what services are included in the database. Therefore it is necessary to query the Nefolog system for the existing parameters for dynamically building a HTML and Javascript based input screen for the web browser. To accomplish this task the following http request is sent to the Nefolog system `http://localhost:8080/nefolog/candidateSearch`. The Nefolog system then returns the following result generated out of database entries(See Listing 4.12).

Listing 4.12: Query for getting variables for offering matcher

```
<?xml version="1.0" encoding="UTF-8"?>
-<resource>
    <name>inputParameters</name>
    -<content>
        <param>servicetype</param>
        <param>provider</param>
        <param>offering</param>
        <param>cpuCores</param>
        <param>cpuSpeed</param>
        <param>memory</param>
        <param>io</param>
        <param>platform</param>
        <param>bandwidth</param>
        <param>storage</param>
        <param>sla</param>
        <param>sites</param>
        <param>licence</param>
        <param>os</param>
        <param>caching</param>
        <param>transactions</param>
        <param>datatransfer</param>
        <param>concurrentConnection</param>
        <param>included_IO</param>
    </content>
</resource>
```

The visualisation system is then parsing the resturned XML data and building HTML / Javascript code out of it. Like for the cost calculator only field which are filled by the user are used to build a query for the Nefolog system asking for fiting offerings. If this would not be done the query string whould also contain entries like storage=null which would lead to an empty result list. This is because most probably there will be no service included int the database which contains the String null for storage.
When the user entered the desired combination of parameter the submit button can be pressed. This then generates an URL containing the names of the entered parameters as well as their value. This query is then sent to the Nefolog system, for instance (`http://localhost:8080/nefolog/candidateSearch?os=linux&cpuCores=16`). This searches in the database for configurations running the linux operating system with 16 or more cores available. The system is then responding again with XML data. This data is then parsed and displayed as the result of the candidate search.

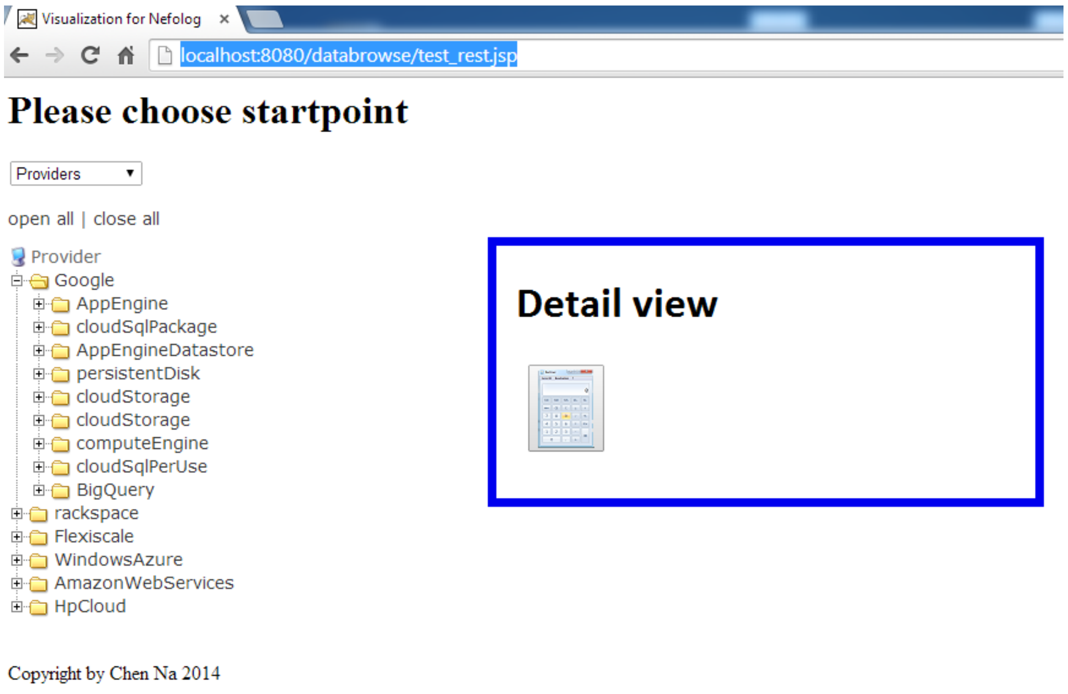
4.2.3 Implementation of the graphical user interface

Result of visualization - navigation view starting from providers

Figure 4.3 and Figure 4.4 shows that this page consists of two parts. One is the tree constructure, the tree shows the visualization of the database. Another is the performance view, that shows the performance of offerings. At first the user chooses one of the starting points. From providers or service types, according the starting point, our system is building the tree structure. If the user chooses the starting point with provider, then they can click the document to find the offering details that are offered from the provider. For example there are six different providers. If the user chooses the provider Google the user then can find the offering of google, like AppEngine, cloudSqlPackage, BigQuery and so on. At the next step, if the user chooses the offering AppEngine the user then can find the different pay models (or called configuration) like free, paid or premier. For example when the user clicks Provider-Google-AppEngine-paid, at the end the right side shows the parameters of performance of this configuration.

Result of visualization - navigation view starting from service types

It could happen that the user does not want to depend on the provider to find the final configuration. Because of this it was also designed that the user can choose services as starting point. If service types is selected the user can get all of the offerings, classified by service types in the database. For example service type Application has offerings like ApplicationPaas from provider HpCloud, AppEngine from provider Google, CloudService from WindowsAzure. In this way the user can also get the information to choose another provider. At the next step the user can find the details of several different configurations. For example the user can choose the offering cloudService from provider WindowsAzure with a1, then the user can get the performance of this offering on the



Visualization for Nefolog

localhost:8080/databrowse/test_rest.jsp

Please choose startpoint

Providers

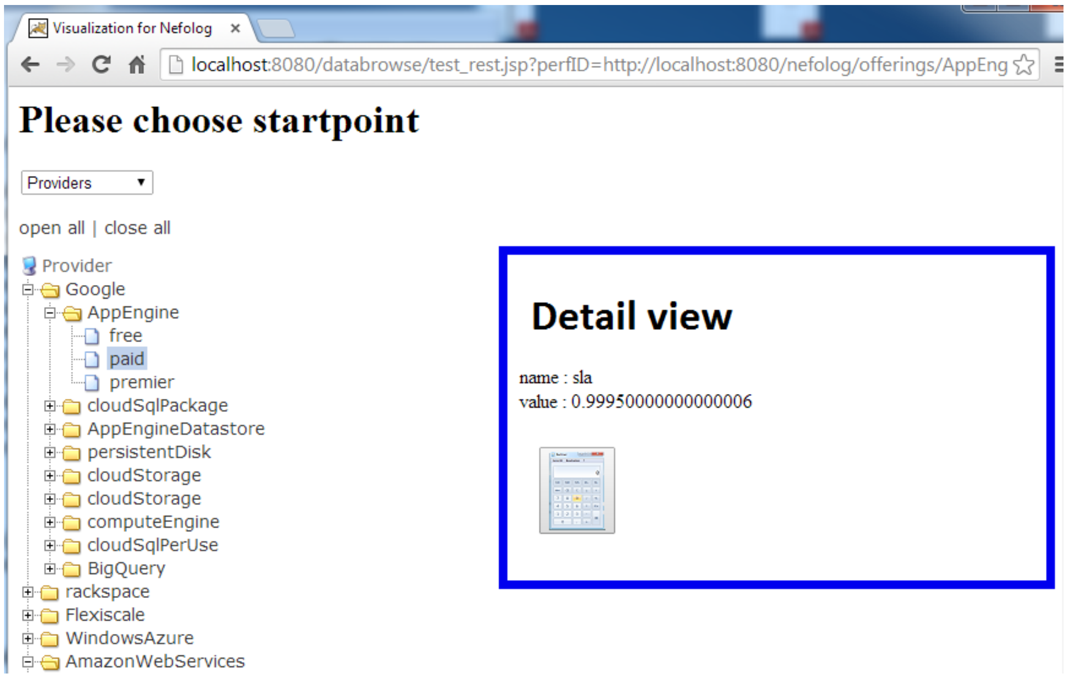
open all | close all

- Provider
  - Google
    - AppEngine
    - cloudSqlPackage
    - AppEngineDatastore
    - persistentDisk
    - cloudStorage
    - cloudStorage
    - computeEngine
    - cloudSqlPerUse
    - BigQuery
  - rackspace
  - Flexiscale
  - WindowsAzure
  - AmazonWebServices
  - HpCloud

Detail view

Copyright by Chen Na 2014

Figure 4.3: screenshot starting point from provider



Visualization for Nefolog

localhost:8080/databrowse/test_rest.jsp?perfID=http://localhost:8080/nefolog/offerings/AppEng

Please choose startpoint

Providers

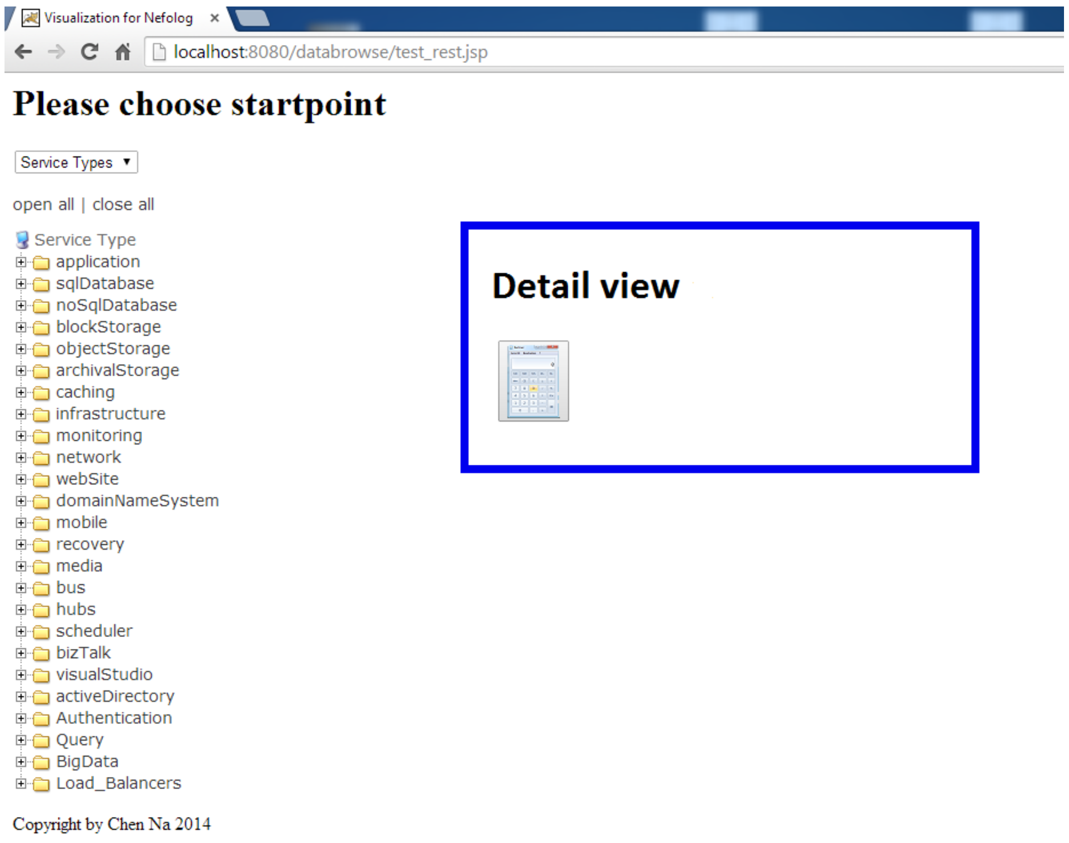
open all | close all

- Provider
  - Google
    - AppEngine
      - free
      - paid
      - premier
    - cloudSqlPackage
    - AppEngineDatastore
    - persistentDisk
    - cloudStorage
    - cloudStorage
    - computeEngine
    - cloudSqlPerUse
    - BigQuery
  - rackspace
  - Flexiscale
  - WindowsAzure
  - AmazonWebServices

Detail view

name : sla
value : 0.99950000000000006

Figure 4.4: screenshot starting point from provider with configuration selected


Visualization for Nefolog

localhost:8080/databrowse/test_rest.jsp

Please choose startpoint

Service Types

open all | close all

Service Type
- application
- sqlDatabase
- noSqlDatabase
- blockStorage
- objectStorage
- archivalStorage
- caching
- infrastructure
- monitoring
- network
- webSite
- domainNameSystem
- mobile
- recovery
- media
- bus
- hubs
- scheduler
- bizTalk
- visualStudio
- activeDirectory
- Authentication
- Query
- BigData
- Load_Balancers

Detail view

Copyright by Chen Na 2014

Figure 4.5: screenshot starting point from service type

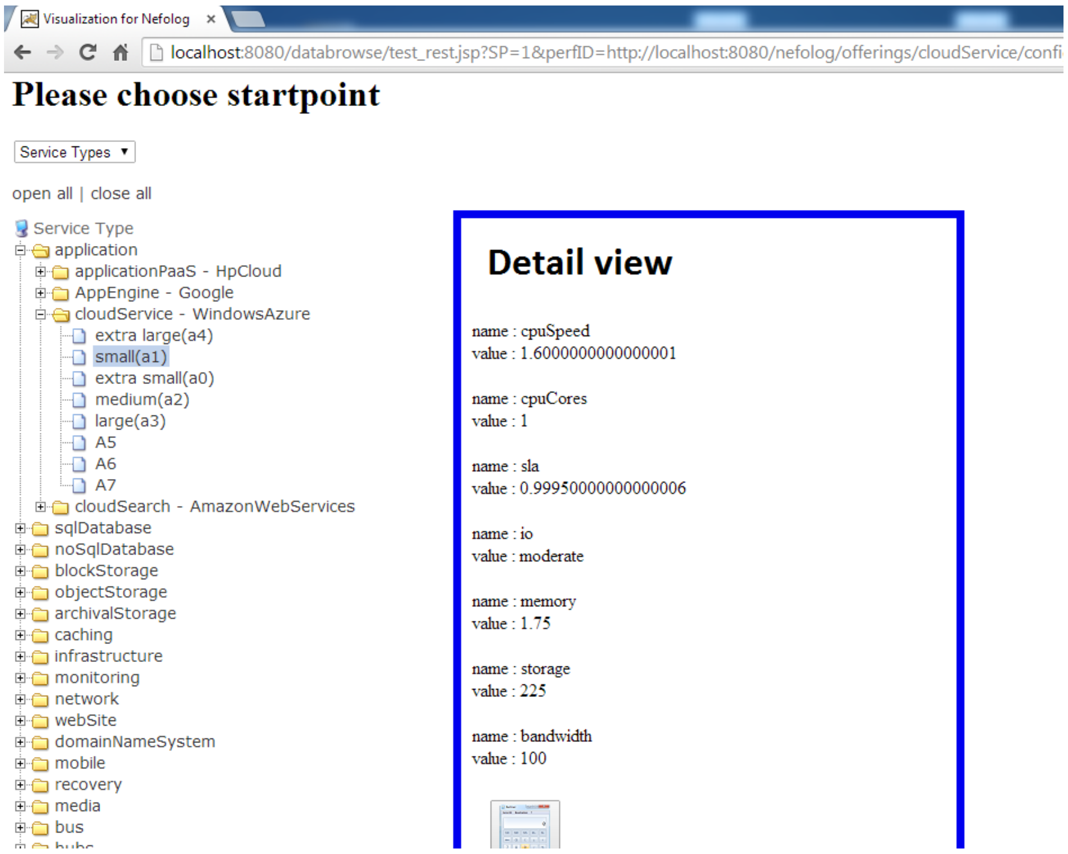
right side of the page containing information like cpu speed, cpu cores, sla, io, memory, storage, and bandwidth.

Result of visualization - costcalculator function view

The performance view shows the performance information, the user clicks the costcalculator button then goes to other page. In this page shows the dynamically parameters, that the user needs to calculate the price. Like Figure 4.7 shows, for example for calculate CloudService from WindowsAzure with small pakage, we need the parameter how many hour and month, how many GB enxternal netzwork egress, which location the users desire, and also usage pattern. The first parameter with configID is read only, show theID of the this configration in Nefolog.

Result of visualization - costcalculator result view

Figure 4.7 shows the calculation result. That means that when the user finished to fill the parameters for the cost calculator and the Submit button is clicked that the calculation result view gets populated. Figure 4.8 and Figure 4.9 show the result from same configuration with different parameters. If the user knows which location to use,



Visualization for Nefolog

localhost:8080/databrowse/test_rest.jsp?SP=1&perfID=http://localhost:8080/nefolog/offerings/cloudService/confi

Please choose startpoint

Service Types

open all | close all

- Service Type
  - application
    - applicationPaaS - HpCloud
    - AppEngine - Google
    - cloudService - WindowsAzure
      - extra large(a4)
      - small(a1)
      - extra small(a0)
      - medium(a2)
      - large(a3)
      - A5
      - A6
      - A7
    - cloudSearch - AmazonWebServices
  - sqlDatabase
  - noSqlDatabase
  - blockStorage
  - objectStorage
  - archivalStorage
  - caching
  - infrastructure
  - monitoring
  - network
  - webSite
  - domainNameSystem
  - mobile
  - recovery
  - media
  - bus

Detail view

name : cpuSpeed
value : 1.6000000000000001

name : cpuCores
value : 1

name : sla
value : 0.99950000000000006

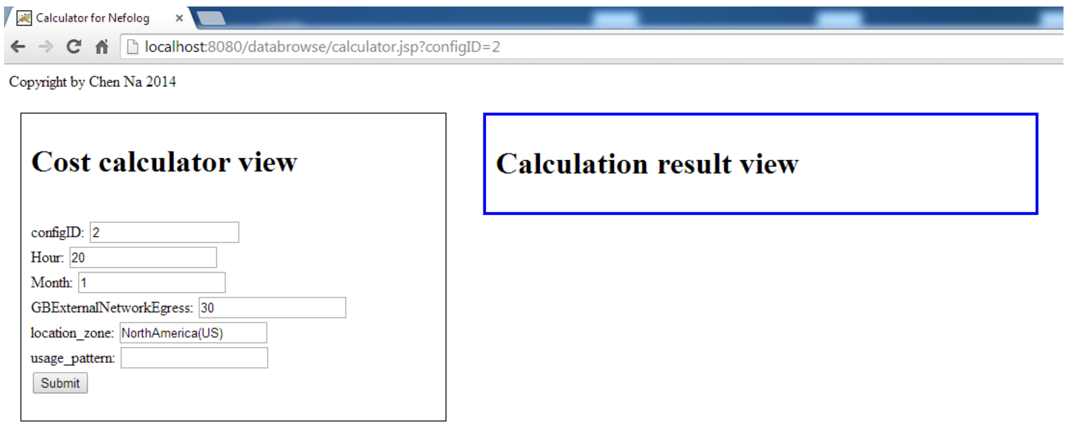
name : io
value : moderate

name : memory
value : 1.75

name : storage
value : 225

name : bandwidth
value : 100

Figure 4.6: screenshot showing performance view



Calculator for Nefolog

localhost:8080/databrowse/calculator.jsp?configID=2

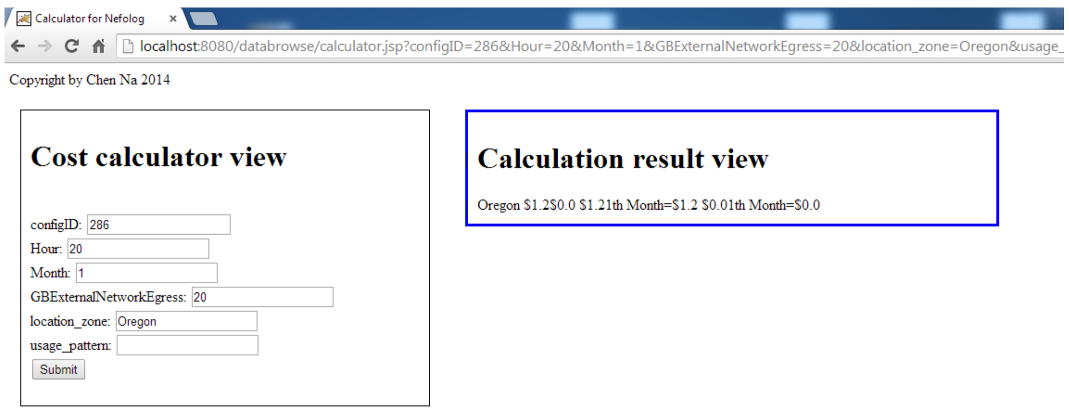
Copyright by Chen Na 2014

Cost calculator view

configID: 2
Hour: 20
Month: 1
GBExternalNetworkEgress: 30
location_zone: NorthAmerica(US)
usage_pattern:
Submit

Calculation result view

Figure 4.7: screenshot of costcalculator function



Calculator for Nefolog

localhost:8080/databrowse/calculator.jsp?configID=286&Hour=20&Month=1&GBExternalNetworkEgress=20&location_zone=Oregon&usage_

Copyright by Chen Na 2014

Cost calculator view

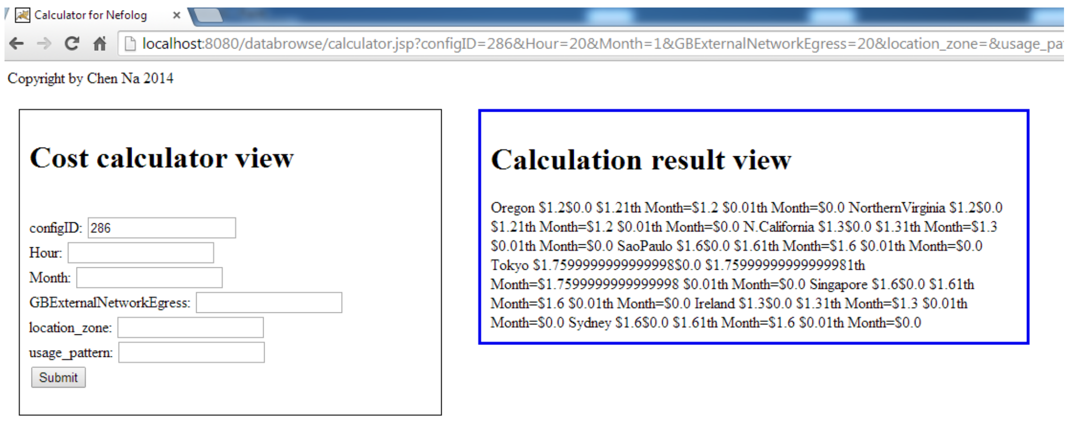
configID: 286
Hour: 20
Month: 1
GBExternalNetworkEgress: 20
location_zone: Oregon
usage_pattern:

Submit

Calculation result view

Oregon $1.2$0.0 $1.21th Month=$1.2 $0.01th Month=$0.0

Figure 4.8: screenshot of costcalculator result



Calculator for Nefolog

localhost:8080/databrowse/calculator.jsp?configID=286&Hour=20&Month=1&GBExternalNetworkEgress=20&location_zone=&usage_pa

Copyright by Chen Na 2014

Cost calculator view

configID: 286
Hour:
Month:
GBExternalNetworkEgress:
location_zone:
usage_pattern:

Submit

Calculation result view

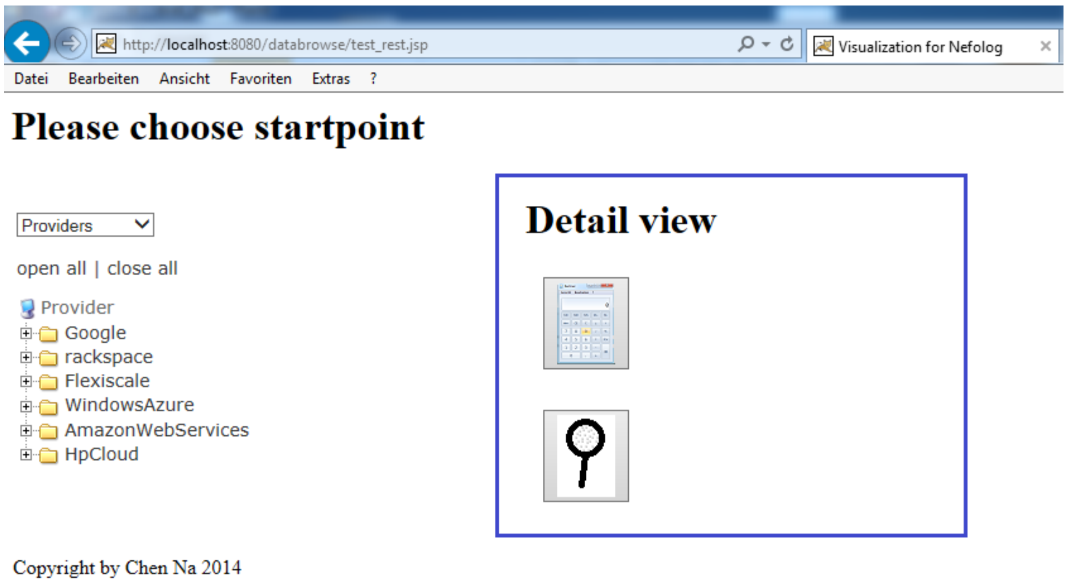
Oregon $1.2$0.0 $1.21th Month=$1.2 $0.01th Month=$0.0 NorthernVirginia $1.2$0.0 $1.21th Month=$1.2 $0.01th Month=$0.0 N.California $1.3$0.0 $1.31th Month=$1.3 $0.01th Month=$0.0 SaoPaulo $1.6$0.0 $1.61th Month=$1.6 $0.01th Month=$0.0 Tokyo $1.7599999999999998$0.0 $1.75999999999999981th Month=$1.7599999999999998 $0.01th Month=$0.0 Singapore $1.6$0.0 $1.61th Month=$1.6 $0.01th Month=$0.0 Ireland $1.3$0.0 $1.31th Month=$1.3 $0.01th Month=$0.0 Sydney $1.6$0.0 $1.61th Month=$1.6 $0.01th Month=$0.0

Figure 4.9: screenshot of costcalculator result2

for example like North America, it will only show the cost of this location, like Figure 4.8 shows. But if the user doesn't know the location, then it will look like Figure 4.9, that all of the locations with this offering will be calculated.

Result of visualization - Candidate search view

The intention of the candidate search view is to to search the Nefolog database for suitable providers dependent on the selected offering parameters. Like in the cost calculator view only fields where a value was entered are transmitted to the search algorithm. This means that the more precise the input is the less results are shown to the user. As Figure 4.10 shows the candidate search view can be called from the main window by clicking on the lens icon. It is independent of if the service provider or service type was selected as start point.
Figure 4.11 shows the candidate search view populated with results. The results are implemented with HTML links leading to the cost calculator concatinated with the con-


http://localhost:8080/databrowse/test_rest.jsp
Visualization for Nefolog
Datei Bearbeiten Ansicht Favoriten Extras ?
Providers
open all | close all

Please choose startpoint

Provider
- Google
- rackspace
- Flexiscale
- WindowsAzure
- AmazonWebServices
- HpCloud

Detail view

Copyright by Chen Na 2014

Figure 4.10: screenshot of candidate search

figuration id. In this way the cost calculation view is called also by clicking on a link in the candidate search view.

Unit names and descriptions for fieldnames

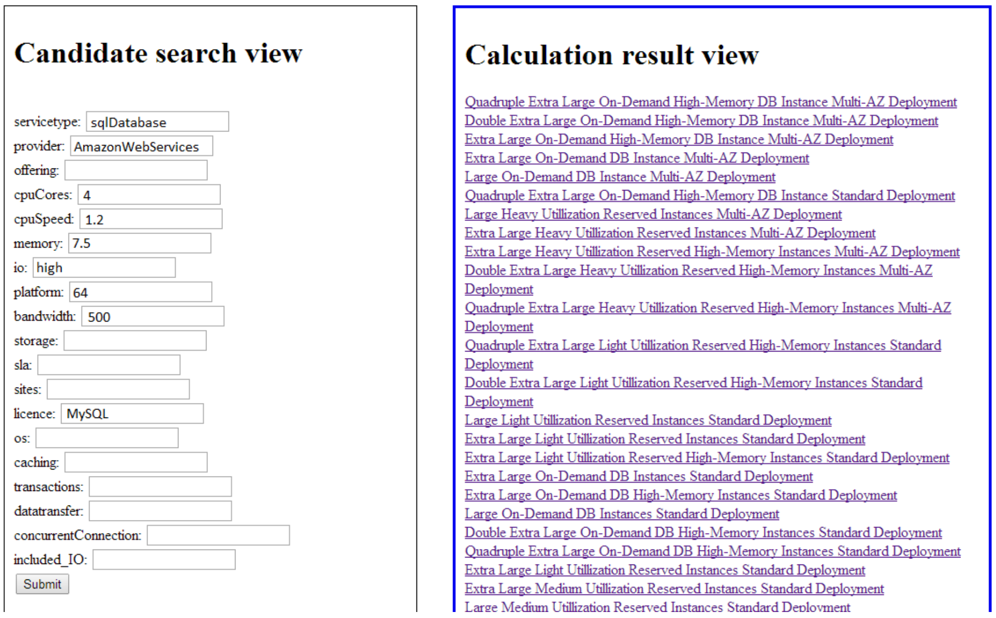
The cost calculation view as well as the candidate search view depend on that the user enters values into fields. Unfortunately the Nefolog database currently only offers short names of the field names and no description to what it is. For instance the field `cpuSpeed`. The user doesn't know what unit is used, is it MHz or GHz? Another example for this is the `platform` field. It expects the values 32 or 64. Since this is not self explaining a hint needs to be shown to the user.

In order to realize this two approaches are possible. One is to adapt the database layout, add the contents to the database and to add Rest server functions to Nefolog system. Then it is possible to use the same approach to access the fields names as well also on the descriptions to the field names. Because of time constraints another solutions was chosen for the implementation. It is a local function using a map returning the suitable description for a given field name. See function refgetUnitFromFieldname. The advantage of this solution is that this solution can be easily replaced by a Rest function as soon as the Nefolog system also supports descriptions.

Listing 4.13: Function getUnitFromFieldname

```
String getUnitFromFieldname(String fieldname)
{
   String retval = null;
   Map candidate_units = new HashMap();

```



Candidate search view

servicetype:	sqlDatabase
provider:	AmazonWebServices
offering:	
cpuCores:	4
cpuSpeed:	1.2
memory:	7.5
io:	high
platform:	64
bandwidth:	500
storage:	
sla:	
sites:	
licence:	MySQL
os:	
caching:	
transactions:	
datatransfer:	
concurrentConnection:	
included_IO:	

Calculation result view

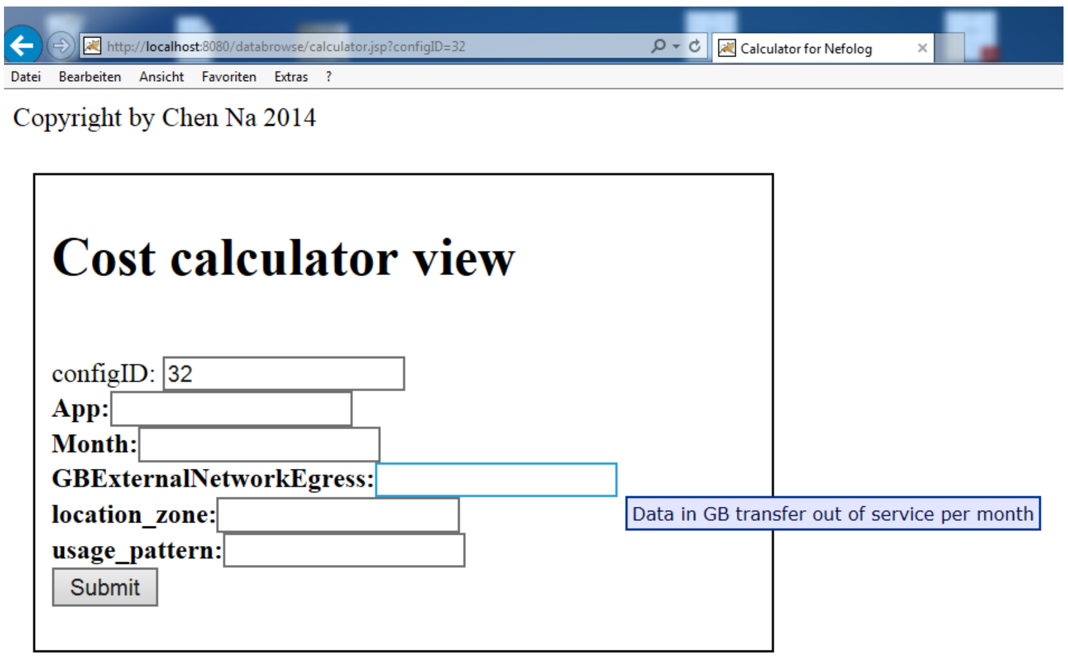
Quadruple Extra Large On-Demand High-Memory DB Instance Multi-AZ Deployment
Double Extra Large On-Demand High-Memory DB Instance Multi-AZ Deployment
Extra Large On-Demand High-Memory DB Instance Multi-AZ Deployment
Extra Large On-Demand DB Instance Multi-AZ Deployment
Large On-Demand DB Instance Multi-AZ Deployment
Quadruple Extra Large On-Demand High-Memory DB Instance Standard Deployment
Large Heavy Utilization Reserved Instances Multi-AZ Deployment
Extra Large Heavy Utilization Reserved Instances Multi-AZ Deployment
Extra Large Heavy Utilization Reserved High-Memory Instances Multi-AZ Deployment
Double Extra Large Heavy Utilization Reserved High-Memory Instances Multi-AZ Deployment
Quadruple Extra Large Heavy Utilization Reserved High-Memory Instances Multi-AZ Deployment
Quadruple Extra Large Light Utilization Reserved High-Memory Instances Standard Deployment
Double Extra Large Light Utilization Reserved High-Memory Instances Standard Deployment
Large Light Utilization Reserved Instances Standard Deployment
Extra Large Light Utilization Reserved Instances Standard Deployment
Extra Large Light Utilization Reserved High-Memory Instances Standard Deployment
Extra Large On-Demand DB Instances Standard Deployment
Extra Large On-Demand DB High-Memory Instances Standard Deployment
Large On-Demand DB Instances Standard Deployment
Double Extra Large On-Demand DB High-Memory Instances Standard Deployment
Quadruple Extra Large On-Demand DB High-Memory Instances Standard Deployment
Extra Large Light Utilization Reserved Instances Standard Deployment
Extra Large Medium Utilization Reserved Instances Standard Deployment
Large Medium Utilization Reserved Instances Standard Deployment

Figure 4.11: screenshot of candidate search 2

```
    // Since units are not stored in database
    ...
    candidate_units.put("cpuSpeed", "GHz");
    candidate_units.put("license", "MySQL; Oracle; SQLServer; SQLWeb;
        SQLStandard");
    ...
  // Get unitname from map
  retval = (String) candidate_units.get(fieldname);

  return retval;
}
```

To display the descriptions to the user it was chosen to display them to the user when a mouse over event over the field is detected. For displaying the tooltips the public available package `wz_tooltip.js` was bound into the project. The result of these steps looks like this (See Figure 4.12).



Copyright by Chen Na 2014

Cost calculator view

configID: 32
App:
Month:
GBExternalNetworkEgress:
location_zone:
usage_pattern:

Data in GB transfer out of service per month

Figure 4.12: screenshot of tooltip for cost calculator

5 Evaluation

In this chapter, we first validate our work using the same example as in Xiu's thesis[1]. The validation process is straightforward: Compare the result from this system and MiDSuS should get the same result. In order to do the comparison there would be at least two possible approaches. One is to downgrade the database to the initial contents without updates and compare the results of the visualization described in this document with the results out of Xiu's thesis.

Another approach is to use the latest database and rerun the tests on MiDSuS and compare then the results to the system described in this documentation. Since both systems are accessing the same database and both are using Nefolog services to access the database the results should be the same. For this section it was chosen to take the second approach because in this way it is also indirectly tested if the updates of the database are working well. We then demonstrate the user-friendliness of the developed system by comparing it with the equivalent functionality in the MiDSuS system.

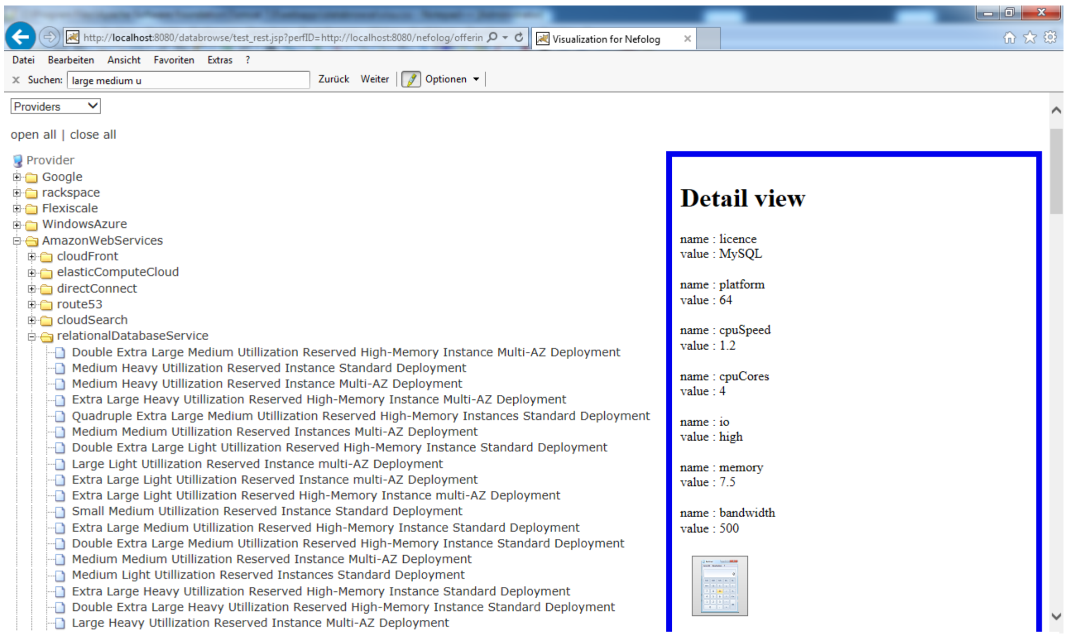
5.1 Nefolog Validation

In the Nefolog system, there have two decision support services, candidate search service and cost calculator service. In Xiu's thesis, to test the cost calculator service with Provider Amazon Web Service, they take the offering RDS (relationalDatabaseService), and configuration is Large Medium Utilization Reserved Instances Multi AZ Deployment. In this configuration, the dynamiclly generated parameter have been enterned with the following example values: 100 hours, 2 months, io operation 10000, 1000GB storage and 1000GB external network egress.

Figure 5.1 shows the main screen starting from providers. The user was started at the provider AmazonWebServices and then expanded the tree at the position relationalDatabaseService in order to then select the defined test configuration. After clicking on the configuration the Detail view section of the screen gets updated with the technical details of the configuration like the number of available CPU cores etc.

The user can then check the details view section of the configuration offers and necessary features. If this is the case the user can click on the calculator button. This causes the calculator.jsp servlet to be called. The URL in the call includes the selected configuration so it there does not need to be selected again.

Figure 5.2 shows the calculator view where the test values have been entered. Since the submit button was not yet pressed the calculation result view is still empty.


http://localhost:8080/databrowse/test_rest.jsp?perfID=http://localhost:8080/nefolog/offerin

Visualization for Nefolog

Datei Bearbeiten Ansicht Favoriten Extras ?

Suchen: large medium u Zurück Weiter Optionen

Providers

open all | close all

- Provider
  - Google
  - rackspace
  - Flexiscale
  - WindowsAzure
  - AmazonWebServices
    - cloudFront
    - elasticComputeCloud
    - directConnect
    - route53
    - cloudSearch
    - relationalDatabaseService
      - Double Extra Large Medium Utillization Reserved High-Memory Instance Multi-AZ Deployment
      - Medium Heavy Utillization Reserved Instance Standard Deployment
      - Medium Heavy Utillization Reserved Instance Multi-AZ Deployment
      - Extra Large Heavy Utillization Reserved High-Memory Instance Multi-AZ Deployment
      - Quadruple Extra Large Medium Utillization Reserved High-Memory Instances Standard Deployment
      - Medium Medium Utillization Reserved Instances Multi-AZ Deployment
      - Double Extra Large Light Utillization Reserved High-Memory Instance Standard Deployment
      - Large Light Utillization Reserved Instance multi-AZ Deployment
      - Extra Large Light Utillization Reserved Instance multi-AZ Deployment
      - Extra Large Light Utillization Reserved High-Memory Instance multi-AZ Deployment
      - Small Medium Utillization Reserved Instance Standard Deployment
      - Extra Large Medium Utillization Reserved High-Memory Instance Standard Deployment
      - Double Extra Large Medium Utillization Reserved High-Memory Instance Standard Deployment
      - Medium Medium Utillization Reserved Instance Multi-AZ Deployment
      - Medium Light Utillization Reserved Instances Standard Deployment
      - Extra Large Heavy Utillization Reserved High-Memory Instance Standard Deployment
      - Double Extra Large Heavy Utillization Reserved High-Memory Instance Standard Deployment
      - Large Heavy Utillization Reserved Instance Multi-AZ Deployment

Detail view

name : licence
value : MySQL

name : platform
value : 64

name : cpuSpeed
value : 1.2

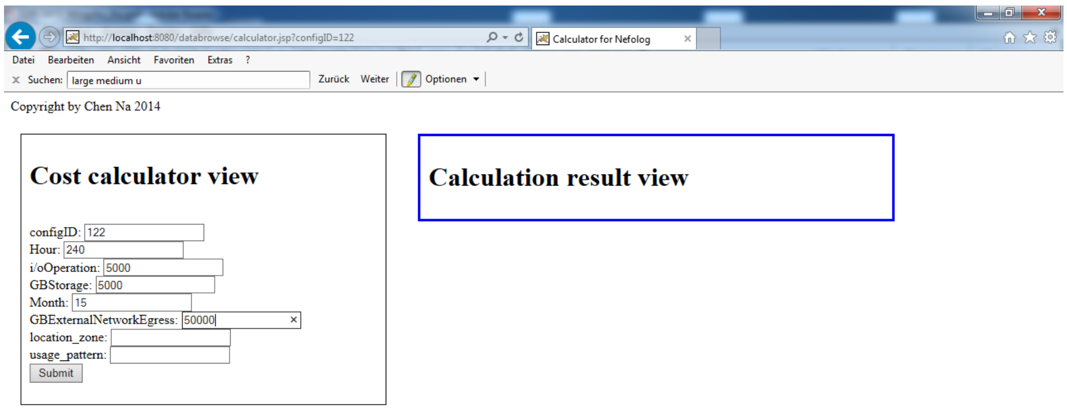
name : cpuCores
value : 4

name : io
value : high

name : memory
value : 7.5

name : bandwidth
value : 500

Figure 5.1: Evaluation - Cost calculation screen 1 - Search and select offering


http://localhost:8080/databrowse/calculator.jsp?configID=122

Calculator for Nefolog

Datei Bearbeiten Ansicht Favoriten Extras ?

Suchen: large medium u Zurück Weiter Optionen

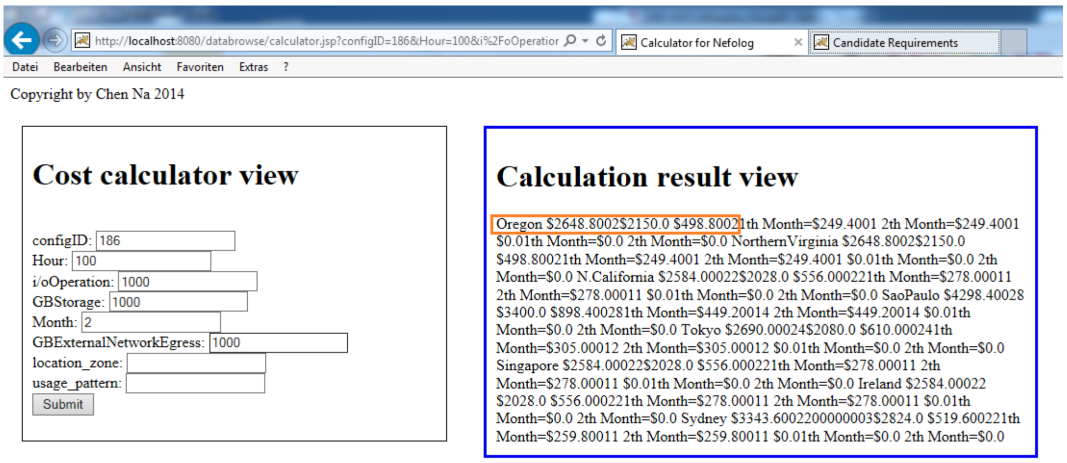
Copyright by Chen Na 2014

Cost calculator view

configID: 122
Hour: 240
i/oOperation: 5000
GBStorage: 5000
Month: 15
GBExternalNetworkEgress: 50000
location_zone:
usage_pattern:
Submit

Calculation result view

Figure 5.2: Evaluation - Cost calculation screen 2 - Entering of details


http://localhost:8080/databrowse/calculator.jsp?configID=186&Hour=100&i%2FoOperatior
Calculator for Nefolog
Candidate Requirements
Datei Bearbeiten Ansicht Favoriten Extras ?
Submit

Copyright by Chen Na 2014

Cost calculator view

configID: 186
Hour: 100
i/oOperation: 1000
GBStorage: 1000
Month: 2
GBExternalNetworkEgress: 1000
location_zone:
usage_pattern:

Calculation result view

```
Oregon $2648.8002$2150.0 $498.80021th Month=$249.4001 2th Month=$249.4001
$0.01th Month=$0.0 2th Month=$0.0 NorthernVirginia $2648.8002$2150.0
$498.80021th Month=$249.4001 2th Month=$249.4001 $0.01th Month=$0.0 2th
Month=$0.0 N.California $2584.00022$2028.0 $556.000221th Month=$278.00011
2th Month=$278.00011 $0.01th Month=$0.0 2th Month=$0.0 SaoPaulo $4298.40028
$3400.0 $898.400281th Month=$449.20014 2th Month=$449.20014 $0.01th
Month=$0.0 2th Month=$0.0 Tokyo $2690.00024$2080.0 $610.000241th
Month=$305.00012 2th Month=$305.00012 $0.01th Month=$0.0 2th Month=$0.0
Singapore $2584.00022$2028.0 $556.000221th Month=$278.00011 2th
Month=$278.00011 $0.01th Month=$0.0 2th Month=$0.0 Ireland $2584.00022
$2028.0 $556.000221th Month=$278.00011 2th Month=$278.00011 $0.01th
Month=$0.0 2th Month=$0.0 Sydney $3343.6002200000003$2824.0 $519.600221th
Month=$259.80011 2th Month=$259.80011 $0.01th Month=$0.0 2th Month=$0.0
```

Figure 5.3: Evaluation - Cost calculation screen 3 - Results of calculation

Table 5.1: Cost calculation service

Parameter name	Value of Xiu's sytem	Value of own system
Total Cost	$ 2648,80	$ 2649,8002
Upfront	$ 2150,00	$ 2150,00
Service Cost	$ 498,80	$ 498,8002

Figure 5.2 shows the calculation servlet after pressing the submit button. Therefore the calculation result view now contains the costs resulting of the provided input parameters. The red rectangle marks the costs for Oregon which will later be compared to the result of MiDSuS.

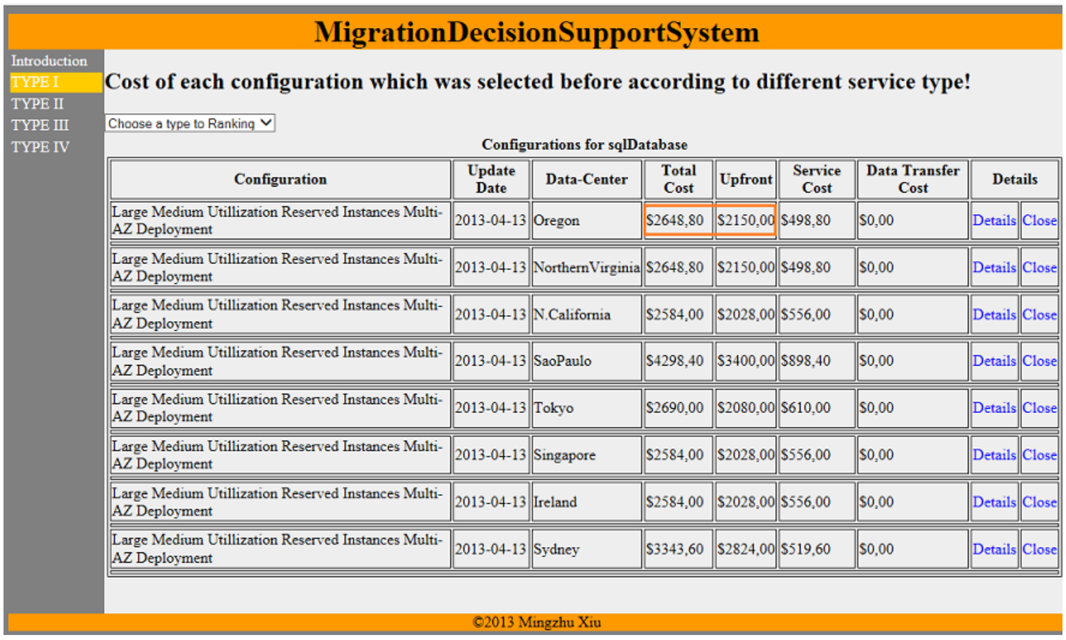
Figure 5.1 shows a screenshot of the output of MiDSuS. Table 5.1 shows the results of both systems:

As it can be seen the results do nearly equal. The only difference is that the values for total costs and service costs have been rounded at MiDSuS. Therefore the test is passed.

5.2 Comparison with MiDSuS

The Nefolog visualization system consists of a search tree starting from providers and starting from offerings. Starting from this it is possible to calculate the costs like tested in the section before. In this section the candidate search view shall be tested.

Figure 5.5 shows the steps for the candidate search in MiDSuS. First the needed requirements have to be entered then the system can find the suitable configurations



Cost of each configuration which was selected before according to different service type!

Configurations for sqlDatabase

Configuration	Update Date	Data-Center	Total Cost	Upfront	Service Cost	Data Transfer Cost	Details	
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	Oregon	$2648,80	$2150,00	$498,80	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	NorthernVirginia	$2648,80	$2150,00	$498,80	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	N.California	$2584,00	$2028,00	$556,00	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	SaoPaulo	$4298,40	$3400,00	$898,40	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	Tokyo	$2690,00	$2080,00	$610,00	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	Singapore	$2584,00	$2028,00	$556,00	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	Ireland	$2584,00	$2028,00	$556,00	$0,00	Details	Close
Large Medium Utillization Reserved Instances Multi-AZ Deployment	2013-04-13	Sydney	$3343,60	$2824,00	$519,60	$0,00	Details	Close

©2013 Mingzhu Xiu

Figure 5.4: screenshot of MiDSuS

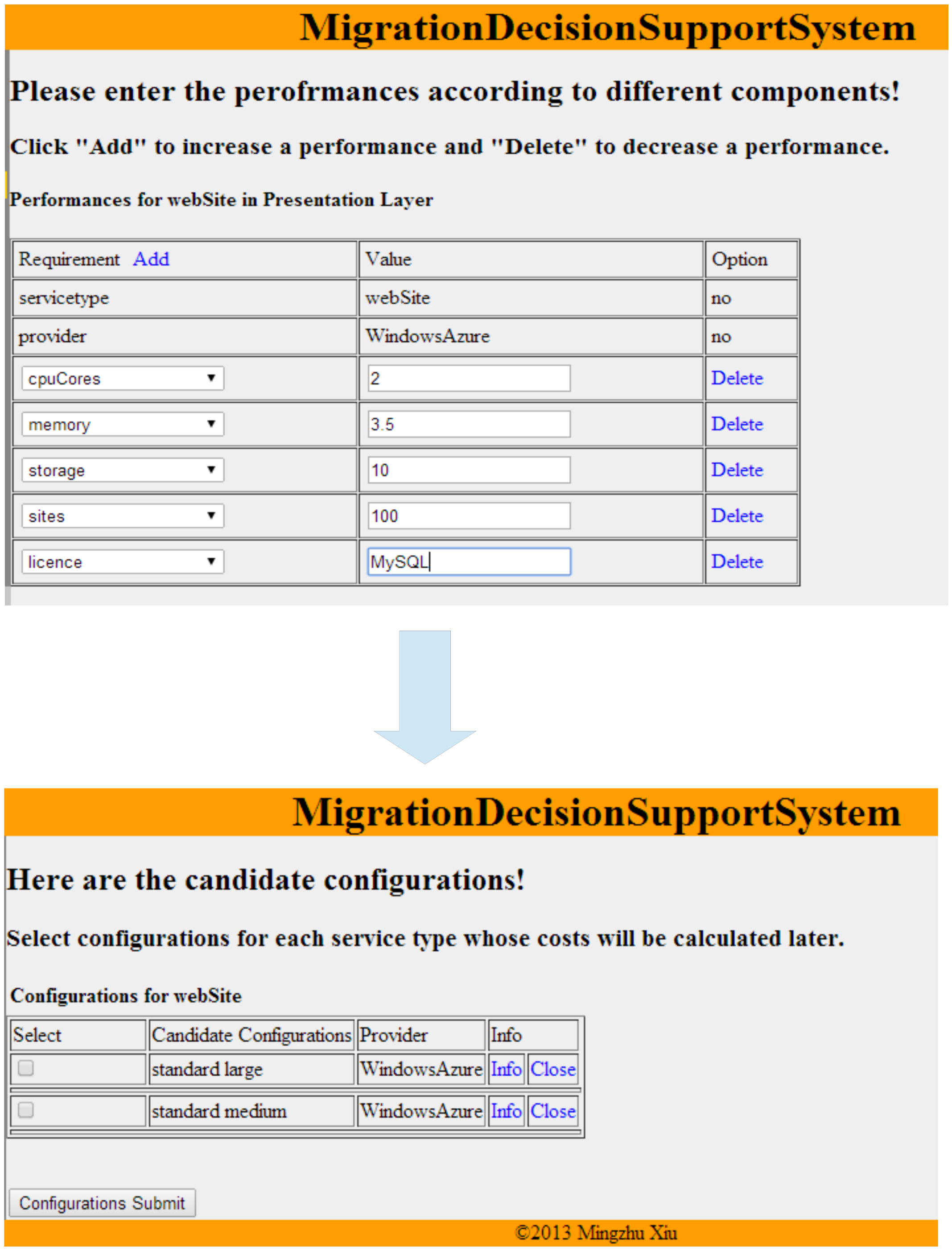
Table 5.2: Candidate search comparison

Value of MiDSuS	Value of own system
standard large	standard large
standard medium	standard medium

fitting to the requirements. For testing purposes the following requirements have been selected: Servicetype = WebSite, Provider = WindowsAzure, cpuCores = 2, memory = 3.5, storage = 10, sites = 100 and license = MySQL. The advantage of specifying the requirements with so many parameters is that this only leads to a short result list.

Figure 5.6 shows the same query entered into the system described in this document. When the submit button is pressed the calculation of finding candidates is triggered and the found configurations are displayed in the calculation result view. This is implemented as links to the calculator, so if the user clicks on configuration the calculation view containing the selected configuration is opened.

As Table 5.2 shows the systems do deliver the same candidates when having the same parameters as input. Therefore the tests of the candidate search view is passed as well as the test for the calculation view.



MigrationDecisionSupportSystem

Please enter the perofrmances according to different components!

Click "Add" to increase a performance and "Delete" to decrease a performance.

Performances for webSite in Presentation Layer

Requirement Add	Value	Option
servicetype	webSite	no
provider	WindowsAzure	no
cpuCores	2	Delete
memory	3.5	Delete
storage	10	Delete
sites	100	Delete
licence	MySQL	Delete

MigrationDecisionSupportSystem

Here are the candidate configurations!

Select configurations for each service type whose costs will be calculated later.

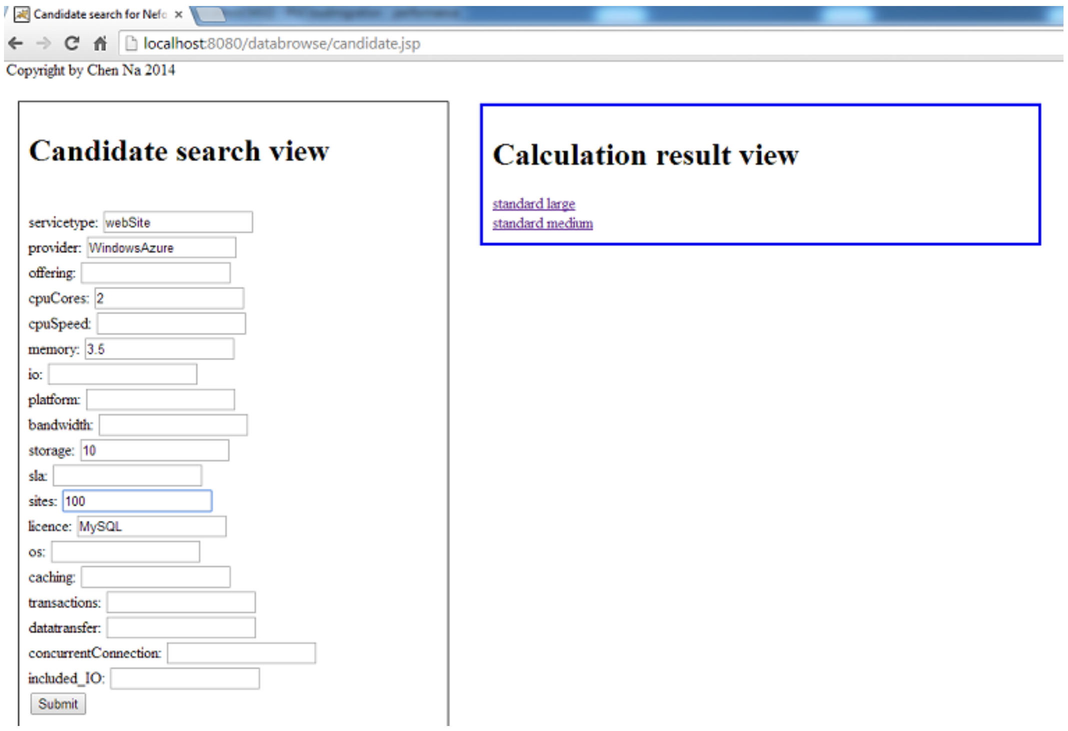
Configurations for webSite

Select	Candidate Configurations	Provider	Info	
☐	standard large	WindowsAzure	Info	Close
☐	standard medium	WindowsAzure	Info	Close

Configurations Submit

©2013 Mingzhu Xiu

Figure 5.5: Screenshot of MiDSuS


Candidate search for Nefo
localhost:8080/databrowse/candidate.jsp

Copyright by Chen Na 2014

Candidate search view

servicetype:	webSite
provider:	WindowsAzure
offering:	
cpuCores:	2
cpuSpeed:	
memory:	3.5
io:	
platform:	
bandwidth:	
storage:	10
sla:	
sites:	100
licence:	MySQL
os:	
caching:	
transactions:	
datatransfer:	
concurrentConnection:	
included_IO:	

Calculation result view

standard large
standard medium

Figure 5.6: Evaluation - Candidate search view

6 Conclusions

6.1 Summary

These days the clould market has many providers offering many different products. Therefore Nefolog as a search engine to find the right provider is quite useful. Xiu [1] built a RESTful interface for the Nefolog system that supports all features neccesary to find providers and offerings. It was one aim of this work to further more improve the usability of the system. It was another aim to update the database contents. This is a very challenging task, because the market is always in movement. New offerings arise and prices are changing regulary.

In order to enable to user to easily find the offering or provider which is searched the graphical user interface was constructed in the following way:

- A tree starting from Providers
- A tree starting from Offerings
- An offering matcher which is searching dependent in required characteristics

If the user already knows from what provider to start the first view is a very good starting point. It maps the whole content of the database to a tree. This enables the user to have a good overview and to easily find the desired configuration. With a click on the configuration the resulting costs can be calculated. With the same flow of operations a configuration can be found starting from offerings. The third starting point is the offering matcher. The user here has to enter the needed features and will then get a list of fitting configurations. Theses configurations. The costs of these configurations then can be calculated again with a click on the configuration.

All functions used for the visualzations are based on the Nefolog system. All communication is only done between the Nefolog system and my visualization. This is done using RESTful services. This means that all request are sent with of a HTTP request in form of a specific URL. This URL then contains the commands what to do. The Nefolog system is then responding using XML code. This code has then to be parsed in order to get to the transmitted content.

6.2 Future Work

The existing implementation of the visualization works well and supports the envisioned features. For future work on the project the performance on loading the webpage could be enhanced. In the current implementation the whole tree is build using many queries to the Nefolog system. Then the complete tree is available as Java script which allows a very fast interaction with the user. The disadvantage of this is that it takes some time before the web page is displayed. For future versions it would be a possibility to only read the first level of the tree out of the Nefolog system. Only if a user then clicks on a branch the next level of this branch is queried. This would then distribute the loading time over the interaction.

Another point which could be enhanced is the offering matcher. In the current implementation the user needs to type in the offerings in text fields. The disadvantage of this is that the user maybe does not know what to type into the fields. Here it would be possible to do a query to the Nefolog system in order to prepare drop down menues. One more step could be to adapt the user interface also to mobile devices. Many websites today offer different views dependent on what device is used. This could be because of the different display resolutions but it could also be because of the different available bandwiths which can be expected. The visualisation could provide a different user interace to the client dependent on the client identification of the used web browser.
It would be another future option to make the user interface customizable, so that the user can choose the used colors and layout of the visualization. This could be then remembered by the client.

One big challenge of the system is the up-to-dateness of the database. This is a vital factor for the accuracy of the hits found by the system. In order to have a current database it would help to create a user interface which alows to easily update the contents without having knowledge about the structure of the SQL database. This user interface could also be also implemented as web based so that the users themselves can update the database. This could be organized similary to the Wikipedia project that other users can check and rollback changes in order to avoid having users entering wrong information into the system. Having a community updating the databse it is not a big task any more.

Bibliography

[1] M.Zhu, "A decision support system for application migration to the cloud," diploma thesis no. 3472, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, 2013.

[2] M. Miller, *Cloud Computing: Web-Based Applications That Change the Way You Work.* Que, 2008.

[3] K. Jamsa, *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security.* Jones Bartlett Pub, 2012.

[4] K. Shah, "Saas revolution in payroll applications." `http://www.enterprisecioforum.com/en/blogs/kaushalshah/saas-revolution-payroll-applications`, March 2013.

[5] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the cloud environment," *Computing*, vol. 95, no. 6, pp. 493–535, 2013.

[6] M. Menzel and R. Ranjan, "Cloudgenius: decision support for web server cloud migration," in *Proceedings of the 21st international conference on World Wide Web*, pp. 979–988, ACM, 2012.

[7] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville, "The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise," *Software: Practice and Experience*, vol. 42, no. 4, pp. 447–465, 2012.

[8] Janakiraman, *Decision support system.* PHI Learning Pvt. Ltd, 2008.

[9] D. J. Power, *Decision Support Systems: Concepts and Resources for Managers.* 2002.

[10] V. Andrikopoulos, S. Strauch, and F. Leymann, "Decision support for application migration to the cloud: Challenges and vision," in *Proceedings of the 3rd International Conference on Cloud Computing and Service Science, CLOSER 2013, 8-10 May 2013, Aachen, Germany*, pp. 149–155, SciTePress, 2013.

[11] Z.Song, "A decision support system for application migration to the cloud," diploma thesis no. 3381, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, 2013.

[12] D.Thau, *The Book of JavaScript: A Practical Guide to Interactive Web Pages.* No Starch Press, 2006.

[13] JSON. `http://json.org/index.html`.

[14] R. York, *Beginning CSS: Cascading Style Sheets for Web Design.* Wiley, 2007.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Ort, Datum, Unterschift