Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diploma Thesis Nr. 3540

Realizing a Decision Support System for Different Deployment Automation Approaches

Bing Shao

Course of Study:	Computer Science
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dipl.-Inf. Johannes Wettinger
Commenced:	1. August 2013
Completed:	31. Januar 2014
CR-Classification:	C.2.4, D.2.1, D.2.11, D.2.13, H.4.2

Abstract

In recent years more and more IT enterprises use Cloud computing to deliver their services. To deploy services on the Cloud, there are different Deployment Automation Approaches (DAA) available, e.g. PaaS- and IaaS-based DAA. It's must be ensured, a particular service is deployable using a certain DAA, which means the deployment requirements of the service are fulfilled by the DAA. There are plenty of particular DAAs on the market. Some of them have different but similar features. Therefore a appropriate choice will be a key issue for the deployment.

This thesis proposes a Decision Support System for different Deployment Automation Approaches (DSS4DAA). Based on the research in the features of different DAAs, we design and implement the basic rule for the decision support system. Its components are reusable and deployable for the future work.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

1.1 Motivation

The business and IT leaders are facing the business challenges from the real world. In business level the rising tide of globalization changes the world markets, accordingly the business logic should be changed to adapter to the markets. The lowering costs of computation are always a key issue for business, which are driving the focus from personal to datacenter-centric computing. Based on it the changes propagated to information technology. The new delivery models for IT services are being used. On the user side the way to access IT services are also changed. Nowadays there are more mobile devices than desktop computers in use, which means more requests from mobile devices are sent. In most mobile devices instead of heavy client, browser and thin client are used to access services.

"I think the Internet was the last big change. The Internet is maturing. They don't call it the Internet anymore. They call it cloud computing." Larry Ellison, the CEO of Oracle said, "When the Internet first started, the primary device connected to it was the personal computer. Every network has enormously complex components that are hidden from consumers. The PC network was very different. The PC was very complex, and was attached to a complex network. Now we've migrated that complexity off the desktop and moved it to Internet servers. That has been recast as cloud computing." [Hes12]

As a consumption and delivery model for IT services cloud computing has its advantages, such as scalability, flexibility, reliability, high availability, to accept those challenges. It's the reason, why more and more IT enterprises deliver their businesses and services to customers through cloud computing. The cloud vendor offers computation, storage, application hosting service and provide coverage on service continuance, offering service level agreements(SLA)-backed service performance. There is a variety of cloud service offering on the market. Each of them has its characteristics. For example, Google App Engine provides an extensible runtime environment for application, but doesn't allow application to access the filesystem. Microsoft Azure provides a wide array of Windows-based services for developing and deploying Windows applications, but only supports Windows. Therefore it's important for the developers, which are required to deploy cloud applications and services, to choose the suitable cloud service for their applications and services.

Concept	Description
$S = \{s_1, ..., s_i\}$	a set of cloud services
$O = \{o_1, ..., o_j\}$	a set of options
$V = \{v_1, ..., v_k\}$	a set of option's values
$F = \{f_1, ..., f_l\}$	a set of cloud service features
$P = \{p_1, ..., p_m\}$	a set of feature's providers
$C = \{c_1, ..., c_n\}$	a set of feature's categories
$R = \{r_1, ..., r_t\}$	a set of application's requirements

Table 1.1: Concepts of the Formal Definition

1.2 Problem Statement

In reality there is a variety of services and applications. Each of them has its own requirements. For example, the self-hosted blogging tool, Wordpress[1]. PHP and MySQL are needed to deploy a Wordpress service. [wor12] In other words, a cloud service, which supports PHP and MySQL, can be used as host to deploy Wordpress on it.

In order to describe the cloud services and applications, Table 1.1 presents the concepts required for the formal definition. In the definition S is a set of offered cloud services, such as Google App Engine, Cloud Foundry, etc. The cloud services have options and features, which are presented by O and F. Options O need a set V to save all the supported values of each option. Features F should be organized by categories C, each feature should have at least one provider. All the providers of each feature are saved in P. For example, Google App Engine supports Java, Python, PHP, Go and Google Cloud SQL, which means the option "Runtime Environments" of GAE has value Java, Python, PHP, Go And GAE has a feature "Google Cloud SQL", which is provided by Google, can be organized in Category "SQL Storage". The application's requirements are presented by R. For example, the requirements of Wordpress are PHP and MySQL.

This work aims to develop a Decision Support System for different Deployment Automation Approaches(DSS4DAA) in order to help developers to make the right choice to deploy their applications or services to cloud. At the first the cloud services should be characterized to a appropriated data model, which should be comparable. Then find suitable cloud service for application to deploy. The deployability of an application on a cloud service means each requirements of the application should be fulfilled by the cloud service, otherwise is undeployable. A decision for application deployment can be defined as follow:

Definition *Decision support for the deployment of application with requirements R to the cloud can be identified with* $\left\{s_i | (\forall r \in R : r \in \hat{F} \vee r \in \hat{O}) \wedge \hat{F} \leftarrow f(s_i) \wedge \hat{O} \leftarrow g(s_i), \hat{F} \subseteq F, \hat{O} \subseteq O\right\}$

[1] http://wordpress.org/

In the definition, the function $f(s_i)$ returns the features of the cloud servie s_i, and the function $g(s_i)$ returns the options of s_i. Note the $\hat{F}$ is the features of s_i, it's part of F. As the same, $\hat{O}$ is the options of s_i, it's part of O.

It's possible, the decision support system has more than one result. A ranking subsystem is very necessary to help consumers select the most suitable cloud service.

1.3 List of Abbreviations

The following list contains abbreviations which are used in this document.

API Application Programming Interface

DAA Deployment Automation Approach

DSL Decision Support Logic

DSS Decision Support System

GAE Google App Engine

IaaS Infrastructure-as-a-Service

JSON JavaScript Object Notation

PaaS Platform-as-a-Service

SaaS Software-as-a-Service

2 Background

2.1 Cloud Computing

One of the most referred-to descriptions of cloud computing was published by the National Institute of Standards and Technology (NIST). "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [MG11] A cloud has the characteristics described in the following:

- **Self-Service**
  Using interface consumers can subscribe to cloud services and resources. The benefit of this self-service ability to the customers is, the provisioning of their needed resources is quickly, so that their business can respond to opportunities in a timely manner.
- **Broad Network Access**
  Consumers are using a variety of devices to access cloud services. The access should be secure and reliable from different networks. It's important for service developers, to use the device and system independently web technologies to keep the availability on different devices.
- **Resource pooling**
  The cloud provider should organize the resource by using a multi-tenant model.
- **Rapid elasticity**
  The consumers' demand is dynamical, so that the provisioning should be re-configured and re-applied based on demand.
- **Measured service**
  The cloud provider monitors status of resources used by consumers.

2.2 Cloud Service Models

The NIST has also defined the cloud computing in three service models, SaaS, PaaS and IaaS. [MG11]

SaaS

Software as a Service is a software delivery model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet. Compare to the traditional software delivery model, SaaS has its advantages,

- easier administration
- automatic update
- global accessibility
- compatibility by using the same version of software
- easier collaboration without version control problem

In this work we don't consider of SaaS. It can be purchased on demand, but cannot be used as target of deployment.

PaaS

Platform as a Service is a concept that describes a computing platform that is rented or delivered as an integrated solution, solution stack or service through an Internet connection. PaaS provides all the infrastructure needed to develop and run applications over the Internet. PaaS has advantages for developers. With PaaS, operating system and runtime features can be updated frequently. Distributed development teams can work together on software development projects. Costs can be reduced by the use of infrastructure services from a single vendor rather than maintaining multiple hardware facilities.

IaaS

Infrastructure as a Service is a service model that delivers computer infrastructure on an outsourced basis to support enterprise operations. IaaS provides physical server or virtual machines, storage, servers and network components. The IaaS provider owns the equipment and is responsible for running and maintaining it. The consumer pays on a per-use based billing. Characteristics of IaaS include:

- Utility computing service and billing model
- Automation of administrative tasks
- Dynamic scaling
- Desktop virtualization
- Policy-based services
- Internet connectivity

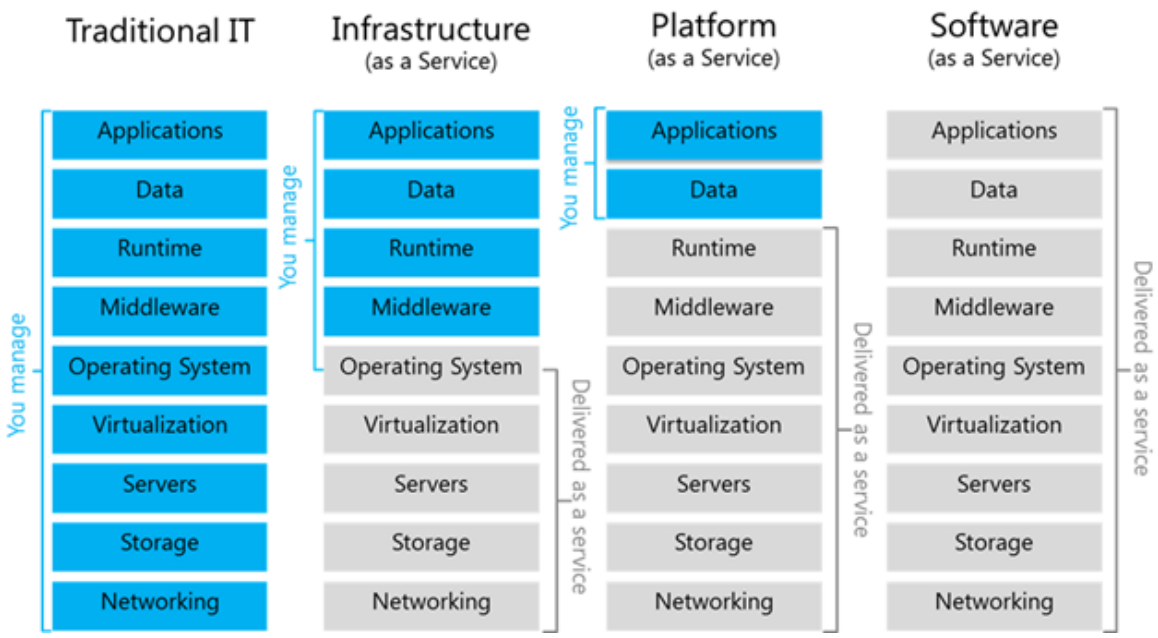

Figure 2.1: Comparison of traditional IT and three basic Cloud Service Models [Cho12]

Based on the definition, Figure 2.1 [Cho12] shows the comparison of traditional IT and three basic cloud service models. In traditional IT all works from maintain infrastructure to develop and publishing application are done by own staff. Using cloud service models the works can be reduced. As an application consumer the only thing to do is to find the need SaaS and purchase it on demand. If the needed SaaS doesn't existed, the software developer can focus on development. Using PaaS the developed application can be deployed without deal with Infrastructure. If no PaaS suit for the developed application, can use IaaS to build suitable platform for the application. In other words, system administrator manages IaaS and provides the needed PaaS to software developer. The software developer uses the PaaS publishing the needed SaaS to end user. The developer really can focus on their code, their program and their business logic. They just push it to the Cloud and let the PaaS engine take care of the functions of the middleware and the infrastructure, as well as the scalability requirements. So, developer doesn't need to waste their time on all kinds of environment setups, which actually don't add a lot of value to their work. They can pay more attention to improve their user experiments.

2.3 Deployment Automation Approaches

This thesis focuses on application deployment. There are different kinds of deployment automation approaches to deploy application to the cloud. Suitable hosted-PaaS providers can be direct used to deploy application. PaaS frameworks are typically used to build custom

platforms based on IaaS offerings or physical servers. If there are no suitable PaaS for application, IaaS providers are also solutions to deploy application.

2.3.1 Hosted-PaaS Providers

PaaS hosting provides not only the needed hardware infrastructure, also the needed operating systems, middleware, database, test environments. Hosted-PaaS providers deliver a computing platform and a solution stack for software vendors, developers who want to avoid the costs and complexity of acquiring and managing their own platform. The down side of hosted-PaaS providers is, some of them have restriction, such as Google App Engine. It can be a problem for the application extensibility.

Google App Engine

Google App Engine [Eng12] is a scalable runtime environment mostly devoted to executing Web applications. These take advantage of the large computing infrastructure of Google to dynamically scale as the demand varies over time. App Engine provides both a secure execution environment and a collection of services that simplify the development of scalable and high-performance Web applications. These services include in-memory data caching, scalable data store, job queues, messaging. [Eng12] Developers can build and test applications on their own machines using the App Engine software development kit (SDK), which replicates the production runtime environment and helps test and profile applications. Once development is complete, developers can easily migrate their application to App Engine, set quotas to contain the costs generated, and make the application available to the world. The languages currently supported are Python, Java, and Go.

There are some restrictions of Google App Engine.

- Filesystem on GAE is read-only access.
- The executable code can only be called from an HTTP request.
- The maximum of request duration is 60 seconds.
- Java applications may only use a subset of classes from the standard JRE.
- For python application support only pure-Python modules.

Heroku

Heroku[1] is a popular Platform as a Service provider that's been announced in 2007, providing simplistic web application hosting for polyglot developers. Heroku supports a wide variety of

[1] https://www.heroku.com/

programming languages, and just about every web framework you can think of. What makes Heroku a an appropriate choice for hosting is that:

- Heroku is built on top of Amazon Web Services, one of the largest cloud providers in the world.
- Heroku doesn't lock you into their platform–they're built using all open source tools so you don't have to rewrite your codebase to make things work.
- Use Git to manage your deployments.
- Instantly provision, resize, and remove any needed infrastructure components (stuff like PostgreSQL, Memcached, etc.).
- Keep private data decoupled from your code via environment variables.
- Instantly scale your infrastructure up and down to support large amounts of traffic.
- Pay for usage only: no monthly fees, contracts, or anything else.
- Heroku has a large collection of addons which makes using additional infrastructure components easily.

2.3.2 PaaS Frameworks

Cloud Foundry

Cloud Foundry[2] is an open Platform as a Service, which is developed and operated by VMware. Cloud Foundry is an Open Source project available through a variety of private and public Cloud distributions. It's written in Ruby, but intended to host any language or any other component. There are many PaaS offerings in the market today, but they all lack some functionality. In consideration of this, VMware introduced their PaaS, Cloud Foundry, with an emphasis on making up for the shortcomings of other PaaS. They focused on three main categories: Clouds, Frameworks and Services.

Firstly, current Clouds in the market are tied to a single Cloud – one public Cloud or one Cloud managed by a single provider. Because Cloud Foundry is open PaaS, it supports an architecture that is extensible and collaborative, and has the ecosystem of technologies that lets developers have the choices they want. So, Cloud Foundry is going to solve the weaknesses and limitations of current PaaS offerings by supporting multiple public Clouds which run in Cloud Foundry as well as private Clouds. This enables developers to have a true kind of public, private and hybrid solution of Clouds running behind the firewall in a public Cloud partner.

Secondly, most Clouds are tied to a single framework and single language. For example, they are supporting only .Net applications, or only Java applications, or only Rails applications, etc. And they are tied to one framework, which is pretty limiting. Companies and developers

[2] http://www.cloudfoundry.com/

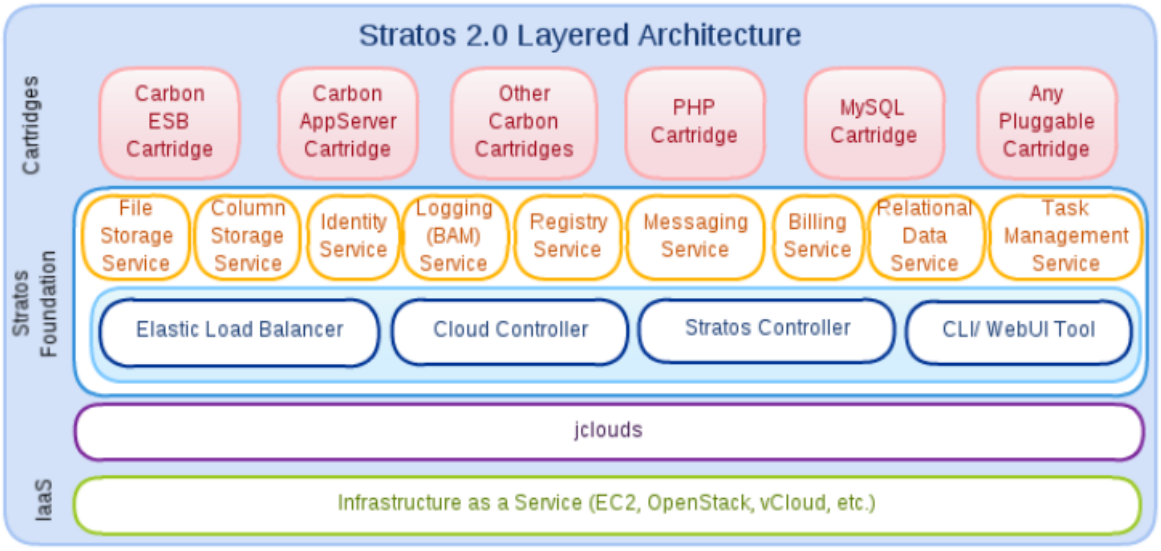

Figure 2.2: Overview of Stratos 2.0 Architecture [WSO14a]

want to mix and match different types of frameworks or different types of applications that all need to coexist. Cloud Foundry can use multiple frameworks, so it can support Java Spring applications and Ruby on Rails applications at the same time. And it is also extensible to a PaaS engine supporting future frameworks and future languages for developers. The frameworks and languages currently supported by Cloud Foundry include Spring, Java, Ruby on Rails, Sinatra, Node, Grails and Play.

Finally, most Clouds are tied to one set of application services, only allowing use of specific databases, specific service technology and management technologies. Again, people want as many choices as possible. Cloud Foundry supports multiple application services. So there are some base services shipped in Cloud Foundry like a relational database, key/value store, and an extensible API allowing more partner technologies to be added in the future. You can plug them into the API to work with Cloud Foundry. The services currently supported by Cloud Foundry include Redis, MySQL, PostgreSQL, RabbitMQ and mongoDB.

Stratos

WSO2 Stratos[3] is a middleware PaaS. It's open source, licensed under Apache 2.0. Before Stratos announced, WSO2 has been doing many releases based on Carbon platform [WSO14b], which redefines middleware by providing an integrated and componentized middleware platform that adapts to the specific needs of any enterprise IT project - on premise or in the cloud. As Figure 2.2 [WSO14a] shows the most important features for deployment, the pluggable

[3] http://wso2.com/cloud/stratos/

Cartridges like Carbon and the possibility to run Stratos on multiple infrastructures via Apache JClouds[4].

Cartridges are a way of packaging a runtime making it available to be run on Stratos. There are two types of cartridges: Carbon and non-Carbon. Carbon or Product Cartridges are wrappers around WSO2 middleware products, including ESB, API Manager, Application Server, Governance Registry, Business Process Manager, Identity Server, WSO2 Message Broker, etc.. These cartridge are provided by WSO2. A non-Carbon cartridge is a virtual machine image created for a specific IaaS, including the desired runtime and some configuration information, which is then registered with Stratos Cloud Controller and deployed to be run on a PaaS.

WSO2 currently has created PHP and MySQL cartridges with the intent to provide more in the future. Developers can create their own cartridges following the guidelines. Carbon or custom cartridges are cloud ready and have built-in multi-tenancy support. Complex products requiring multiple cartridges in order to run are not supported yet, but they will be in the next version, according to WSO2.

The other major feature available in Stratos 2.0 is the possibility to deploy the PaaS on multiple IaaS infrastructures, including the ability to dynamically switch from an IaaS to another or scaling across different IaaS as needed. This features has been added by incorporating JClouds technology, which expands the number of IaaS supported to 30 [JCl14], including all OpenStack variants, VMware, Eucalyptus, and Amazon EC2.

2.3.3 IaaS Tools

Chef

Chef[5] is a systems and cloud infrastructure automation framework that makes it easy to deploy servers and applications to any physical, virtual, or cloud location, no matter the size of the infrastructure. Figure 2.3 [ops14] shows the architecture of chef. Chef comprises three main elements: a server, one or more nodes, and at least one workstation. Each node contains a chef-client that performs the various infrastructure automation tasks that each node requires. The chef-client relies on abstract definitions, which are known as cookbooks and recipes, that are written in Ruby. Each definition describes how a specific part of your infrastructure should be built and managed. The chef-client then applies those definitions to servers and applications, as specified, resulting in a fully automated infrastructure. When a new node is purchased, the only thing the chef-client needs to know is which cookbooks and recipes to apply.

The server manages every node in the organization. It ensures that the right cookbooks and recipes are available, that the right policies are being applied, that the node object used during the previous chef-client run is available to the current chef-client run, and that all of the

[4] http://jclouds.apache.org/
[5] http://www.getchef.com/

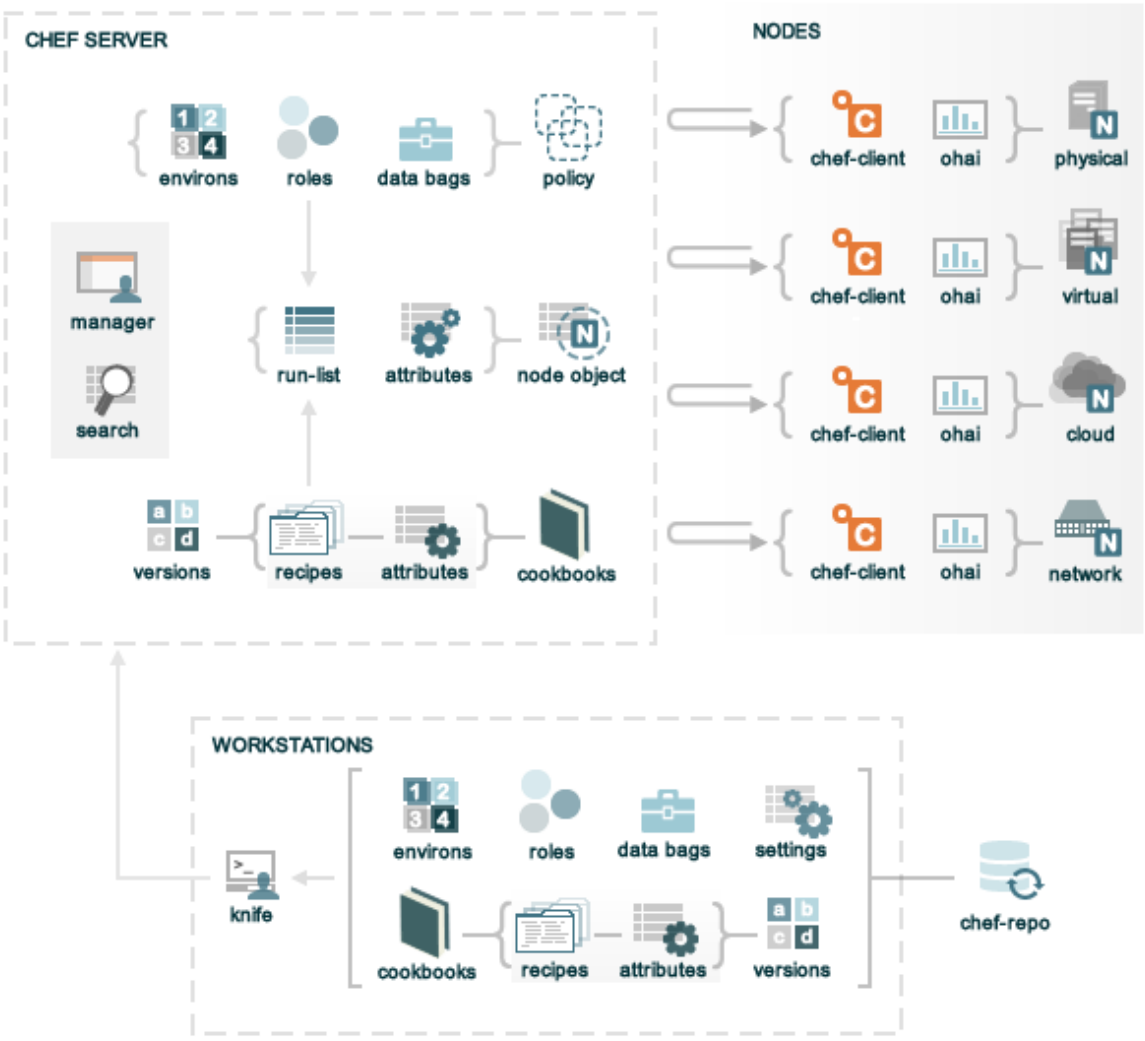

Figure 2.3: Overview of Chef Architecture [ops14]

nodes that will be maintained by the chef-client are registered and known to the server. The workstation is the location from which cookbooks and recipes are authored, policy data such as roles, environments, and data bags are defined, data is synchronized with the chef-repo, and data is uploaded to the server.

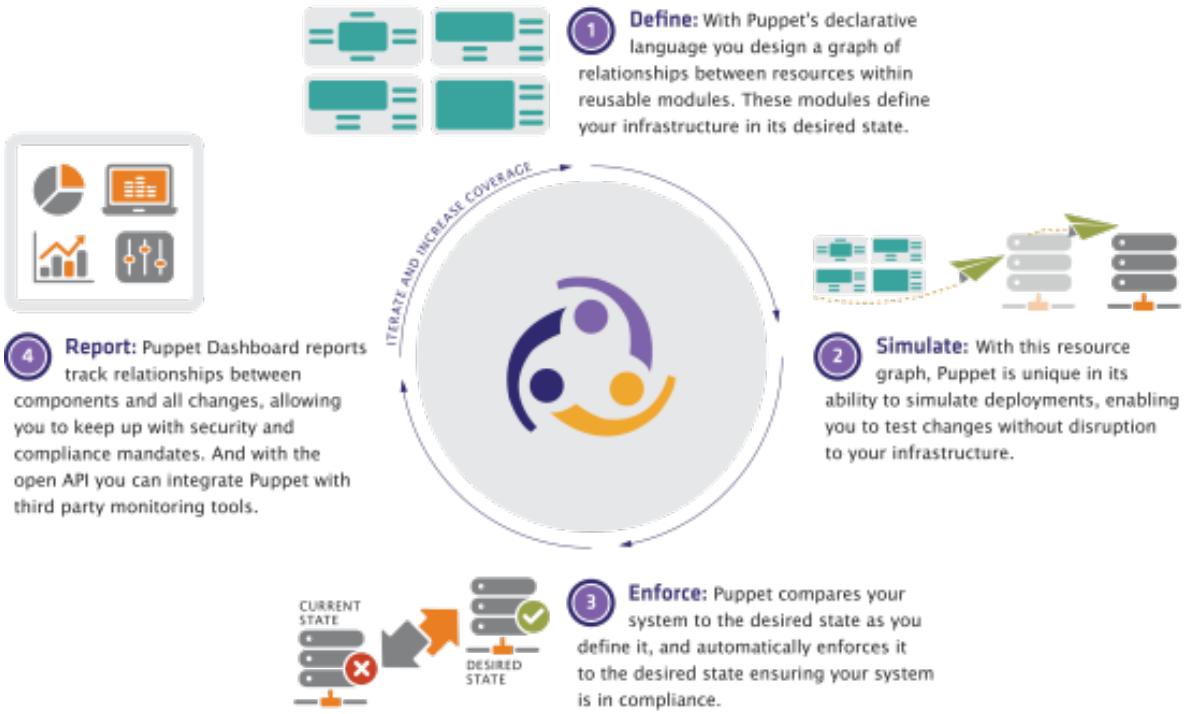

Figure 2.4: Puppet Working Process [pup14]

Puppet

Puppet[6] is IT automation software that helps system administrators manage infrastructure throughout its lifecycle, from provisioning and configuration to orchestration and reporting. Using Puppet, you can easily automate repetitive tasks, quickly deploy critical applications, and proactively manage change, scaling from 10s of servers to 1000s, on-premise or in the cloud. Figure 2.4 [pup14] shows how Puppet works.

1. **Define** the desired state of the infrastructure's configuration using Puppet's declarative configuration language.
2. **Simulate** configuration changes before enforcing them.
3. **Enforce** the deployed desired state automatically, correcting any configuration drift.
4. **Report** on the differences between actual and desired states and any changes made enforcing the desired state.

[6] http://puppetlabs.com/

Juju

Juju[7] is an "automatic service orchestration" project launched by Canonical, developers of the Ubuntu Linux-based operating system, to deploy, manage and scale software and interconnected services across one or more Ubuntu servers and cloud platforms. Juju is a next generation service orchestration framework. It has been likened to APT for the cloud. In Ubuntu APT is a package management tool, it understands collections of packages called "repositories" and is able to find the packages needed to satisfy inter-package dependencies. With Juju, different authors are able to create service formulas, called charms, independently, and make those services coordinate their communication and configuration through a simple protocol.

2.4 Decision Support System

Decision-making analysis was conducted by the Carnegie Institute of Technology in the late 1950s and early 1960s. The Massachusetts Institute of Technology (MIT) applied computer technology to decision-making theory in the 1960s. [KM78] By the 1980s, intensive research on DSS was underway, and new theories and concepts emerged from single-user models of DSS, including organizational decision support systems (ODSSs), group decision support systems (GDSSs) and executive information systems (EISs). By 1990 DSS was broadened to include data warehousing and online analytical processing.

Another taxonomy for DSS has been created by Daniel J. Power. Using the mode of assistance as the criterion, Power differentiates communication-driven DSS, data-driven DSS, document-driven DSS, knowledge-driven DSS, and model-driven DSS. [Pow02]

- **Communication-driven DSS** supports more than one person working on a shared task.
- **Data-driven DSS** emphasizes access to and manipulation of a time series of data.
- **Document-driven DSS** manages, retrieves, and manipulates unstructured information in a variety of electronic formats.
- **Knowledge-driven DSS** provides specialized problem-solving expertise stored as facts, rules, procedures, or in similar structures. [Pow02]
- **Model-driven DSS** emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model.

Three fundamental components of a DSS architecture are database (or knowledge base), model and user interface. [Pow02] User interface is used as input and output of the system. The database is required as user knowledge and expertise by the user. The model is used to make decision.

[7] https://juju.ubuntu.com/

This thesis uses knowledge-driven DSS. The existed deployment automation approaches should be stored as knowledge base. Zimmermann introduced a decision modeling framework for service-oriented architecture design. Figure 2.5 [Zim13] shows the overview of Arc decision based on [Zim09]. Requirements are gotten from `DecisionRequired`. `DecisionMade` makes decision. If mismatch refactoring are need, which is defined as undecide-redecide actions. Based on it a decision support logic should be used as the model of the decision support system.

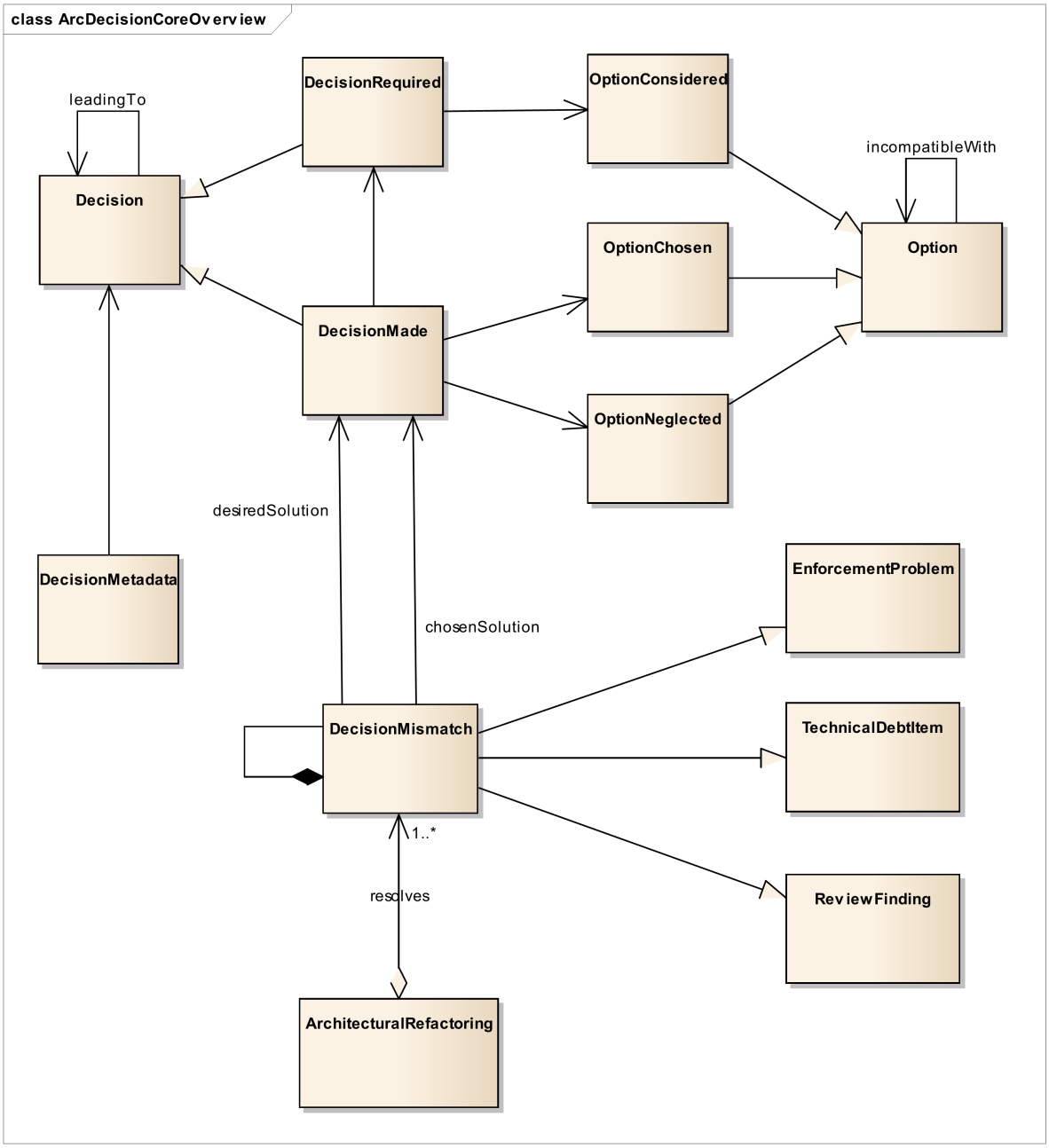

Figure 2.5: Arc Metamodel of decision [Zim13]

3 Specification

This thesis focuses on the decision support system of application deployment. This DSS should have the following characteristics

- **Extensibility**
  This work defines the basic rule of a DSS for application deployment. The functionality of DSS should be extensible in the further work. The features of deployment automation approaches can be extended by the cloud service providers. As knowledge base it need to be extend in the future.
- **Distributability**
  The components of the DSS should be distributable. As a service this DSS should be deployable. Distributability of components makes deployment of DSS flexible.
- **Reusability**
  The components are not only for this DSS. It should be reused in other decision support systems. The interface of each components should be specified.
- **Easy to maintain**
  Because of the update of cloud service providers, the maintain of knowledge base may be a recurring task. It's very necessary to make it easier and independence.
- **Integration-aware**
  It should be possible to integrate the developed DSS with other DSSs, like the DSS for cost calculation [Son13].

To deploy an application to cloud, two kind of information are needed, requirements of application and available features of deployment automation approaches. These information are gained from different actors. Like figure 3.1 shows, user and DSS. User chooses the application requirements, includes the weights of requirements, which are used to calculate ranking. The application requirements are used to make decision. Each application has its own requirements. For instance, some typical requirements can be:

- **Runtime environment**
  Web-based services and applications need runtime environment to run. It can be a develop language environment, like Java, PHP, Python, Ruby, or a framework, such as Rails, Django, Spring. Note that applications can be develped using multi-language or frameworks. In this case the application needs a multi-runtime environment.

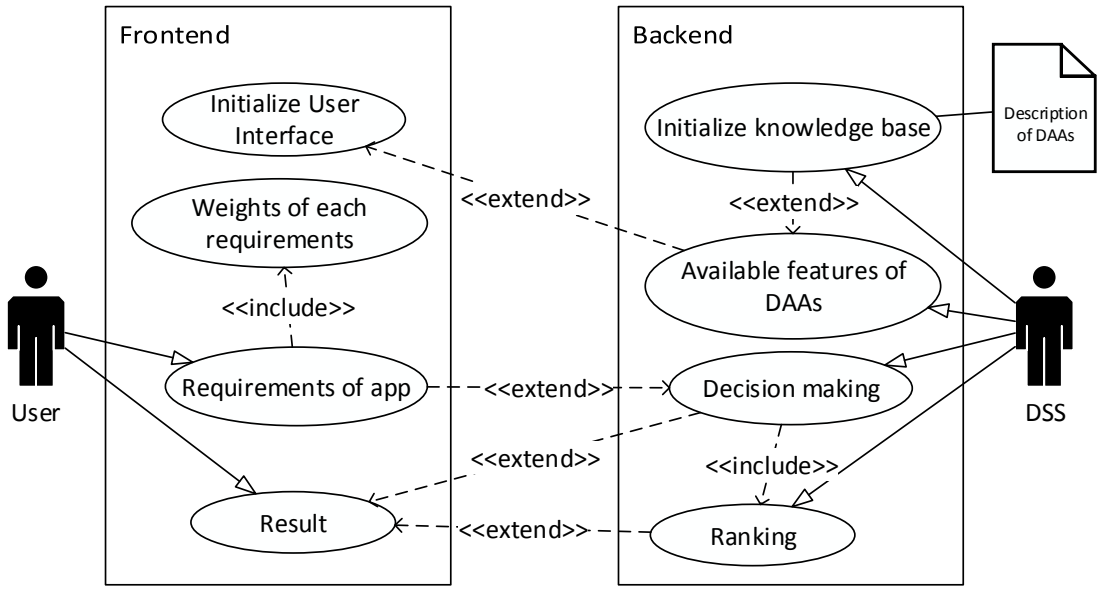

Figure 3.1: Use Case Diagram

- **Supported database**
  Database is a typical requirement of application. It could be a SQL database, e.g. MySQL, PostgreSQL, or a NoSQL database, e.g. MongoDB, Redis.
- **Supported Web Servers**
  Web servers are used to run web-based services. The typical web servers are Apache, Jboss, Nginx, Microsoft IIS.
- **Filesystem access**
  If application need to read data from filesystem, such as reading configuration file, then needs a read-only filesystem access. If application need to write data to filesystem, like modify the configuration file or update a photo to cloud, then the filesystem access should be read-write.
- **Data processing services**
  Some application need server-side data processing. For example, the self-hosted invoicing and project management service Pancake[1] needs a server-side image processing GD[2]. [pan14]

The features of deployment automation approaches are described. The description of DAAs uses documents. The reasons of using documents instead of database to describe DAAs are

[1] https://pancakeapp.com/
[2] http://libgd.bitbucket.org/

independence, easy to maintain. The document formats can be CSV, XML, JSON, YAML. Knowledge base is initialized based on the description. In backend decision support logic has two components, decision making and ranking. They use the application requirements as parameters, to make decision and handle ranking.

As a component of the DSS, user interface is used to get the application requirements from user as input and put the result back to user. Each application has its own requirements. In order to make decision, the requirements should be comparable with the features of DAAs, which means the modeling of application requirements and features of DAAs should be identify. They are two solutions to make the comparison possible, modeling from the application requirements and modeling from DAAs.

4 Design & Implementation

This chapter presents design and implementation of the system. In Section 4.1 the architectural solution considered to build the system, which fulfills the requirements specified in Chapter 3. Section 4.2 shows the deployment requirements. As the components of the DSS, Knowledge Base and Decision Support Logic are described in Section 4.3 and Section 4.4.

4.1 System Design

As mentioned before, knowledge base, model and user interface are the fundamental components of a decision support system. The data model of knowledge base is designed for not only itself, but also used by decision support logic. The DSL is designed as model to make decision. as in- and output of the system the user interface is not detailed introduced, expect the API for the UI and the messages.

Figure 4.1 shows the overview of activity diagram for the decision support system. The first step is initialization. Before system starting, the information of cloud service providers as knowledge base are saved into external documents, such as XML, JSON, YAML. All the documents, which save knowledge base, should be loaded into system. The user interface is used to get the requirements of application and show the result. The requirements of application are used to make decision, which means it should be comparable with the features of deployment automation approaches. It's a solution, provides available features of all DAAs. User just need select the needed requirements for their application from the selection list. It's possible, the requirements are not features of DAAs, which means there are no DAAs support those requirements yet. In this case, the deployment of application is not possible. It's no need to use this decision support system. Otherwise it's suitable to get requirements of application. It's the reason, why user interface is initialized after initialization of knowledge base.

After initialization, system is ready to use. The requirements as input are sent to decision support logic. The requirements are chosen from the features list of DAAs, which means every requirement is support at least by one DAA. If a DAA support all the requirements, it's a suitable DAA for the application. Otherwise it needs a recommend deployment solution, a combination of DAAs. For example, a use-developed application needs write access of filesystem, Java runtime and MySQL support. In knowledge base there are PaaS-based DAAs support Java and MySQL, but no write access of filesystem. Only IaaS-based DAAs support write access of filesystem. A solution could be combination of a PaaS-based DAA and a IaaS-based DAA, and the Paas-based DAA should deployable on the IaaS-based DAA.

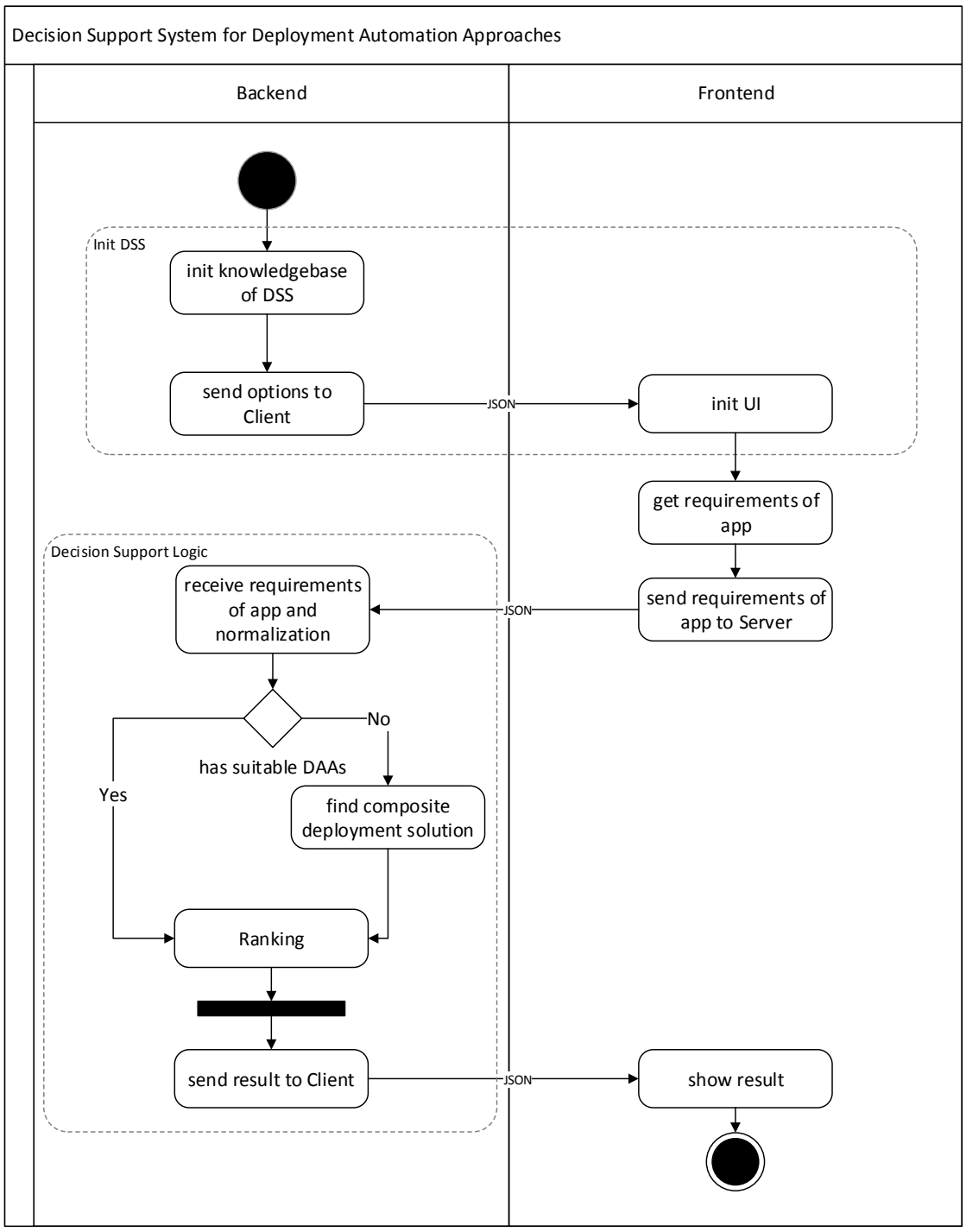

Figure 4.1: Activity Diagram of Decision Support System

In decision support logic, a ranking subsystem is necessary. Some requirements, like PHP, Java, MySQL, are supported by almost each PaaS-based DAAs. If a application has the those requirements, there are many suitable DAAs. The ranking is used to help the user to make the choice from the list. Ranking needs parameter of each requirements, Weights. It should be set by user. The message between frontend and backend are using JSON.

4.2 Deployment Requirements

The common requirements of application are described like Wordpress [wor12], a list of requirements. There is another case like Joolma[1]. The requirements of Joolma 3.x are : [joo14]

- **Software**: PHP
- **Supported Databases**: MySQL, MSSQL, PostgreSQL
- **Supported Web Servers**: Apache, Nginx, Microsoft IIS

Note, there are more than one supported databases and web servers. It doesn't means all the databases or web servers are required by Joolma. One of them is enough. To describe the requirements correctly, logical expression is used. The requirement of Joomla should be formatted as:

$$r_{a1} \wedge (r_{b1} \vee r_{b2} \vee r_{b3}) \wedge (r_{c1} \vee r_{c2} \vee r_{c3})$$

$$\begin{aligned} with \quad & r_{a1} : \text{PHP requirement} \\ & r_{b1} : \text{MySQL requirement} \\ & r_{b2} : \text{MSSQL requirement} \\ & r_{b3} : \text{PostgreSQL requirement} \\ & r_{c1} : \text{Apache requirement} \\ & r_{c2} : \text{Nginx requirement} \\ & r_{c3} : \text{Microsoft IIS requirement} \end{aligned}$$

As the same, the requirements of Wordpress should be:

$$r_a \wedge r_b$$

$$\begin{aligned} with \quad & r_a : \text{PHP requirement} \\ & r_b : \text{MySQL requirement} \end{aligned}$$

[1] http://www.joomla.org/

Logical expression should be used to describe requirements of application, which must be fulfilled. And some requirements are optional, which means it's unrelated with the deploy availability. But it's useful for the ranking. Therefore the deployment requirements can be described as

$$R = \begin{cases} \mathsf{KnockedOutRequirements} & \text{must be fulfilled, described using logical expression} \\ \mathsf{OptionalRequirements} & \text{optioinal, described using list} \end{cases}$$

Normally JSON is used as data format in Frontend. The requiements are gotten from user interface. So we choose JSON as data format for the requirements of application. Listing 4.1 implements the requirements of Joolma using JSON.

4.3 Knowledge Base

Knowledge base is the fundamental part of implementing decision support logic. It provides information of DAAs. Each DAA has its own features. Each features should be categorized in different categories. Figure 4.2 shows Entity-Relation diagram for description of deployment automation approach. Entity DAA contains the basic information of a Cloud service provider, name, type, description. Type of DAA should be Hosted-PaaS, PaaS Framework and IaaS. A DAA could have many categories, like SQL Storage, NoSQL Storage, Mail Services, Data Processing Services, etc. So DAA has a one-to-many relationship with Category entity. Each category could have many features. In some case a feature could have more than one provider. For example, as a NoSQL storage service, MongoDB[2] can be provided by MongoHQ[3] or MongoLab[4]. Therefore Category entity has a one-to-many relationship with Provider entity.

Some features of DAA are simply, need not described by using the Category-Feature-Provider Entity-Relatioin. For example, runtime environment. The common runtimes could be Java, PHP, Python, Ruby, Node.js. They are standard. The Provider entity is not needed. Those features can be described by using entity Option. As the basic information each Option entity has name and values. For example, Google App Engine supports Java, PHP, Python, Go. For GAE the Option "Runtime Environment" has value "Java, PHP, Python, Go".

Note, that Option entity is not only for DAA entity. Feature entity and Provider entity could also have Option entity. Each feature may has its own properties. Feature can be detailed described using Option entity. For Provider entity, the different providers of same feature may not have same instance. For example, as intance of MongoDB, MongoHQ supports *Automatic Database Optimization*, but MongoLab doesn't specified if *Automatic Database Optimization* supported.

[2] http://www.mongodb.com/
[3] http://www.mongohq.com/
[4] http://mongolab.com/

Listing 4.1 Requirements of Joomla using JSON

```
{
  ‘‘Operation’’: ‘‘AND’’,
  ‘‘data’’: [
    { ‘‘Runtime Environments’’: ‘‘PHP’’ },
    { ‘‘Operation’’: ‘‘OR’’,
      ‘‘data’’: [
        { ‘‘categories’’: ‘‘SQL Storage’’,
          ‘‘features’’: {
            ‘‘Name’’: ‘‘Microsoft SQL’’,
            ‘‘Options’’: [ ... ]
          }
        },
        { ‘‘categories’’: ‘‘SQL Storage’’,
          ‘‘features’’: {
            ‘‘Name’’: ‘‘MySQL’’,
            ‘‘Options’’: [ ... ]
          }
        },
        { ‘‘categories’’: ‘‘SQL Storage’’,
          ‘‘features’’: {
            ‘‘Name’’: ‘‘PostgreSQL’’,
            ‘‘Options’’: [ ... ]
          }
        }
      ]
    },
    { ‘‘Operation’’: ‘‘OR’’,
      ‘‘data’’: [
        { ‘‘categories’’: ‘‘Web Server’’,
          ‘‘features’’: {
            ‘‘Name’’: ‘‘Apache’’,
            ‘‘Options’’: [ ... ]
          }
        },
        { ‘‘categories’’: ‘‘Web Server’’,
          ‘‘features’’: {
            ‘‘Name’’: ‘‘Nginx’’,
            ‘‘Options’’: [ ... ]
          }
        },
        { ‘‘categories’’: ‘‘Web Server’’,
          ‘‘features’’: {
            ‘‘Name’’: ‘‘Microsoft IIS’’,
            ‘‘Options’’: [ ... ]
          }
        }
      ]
    }
  ]
}
```

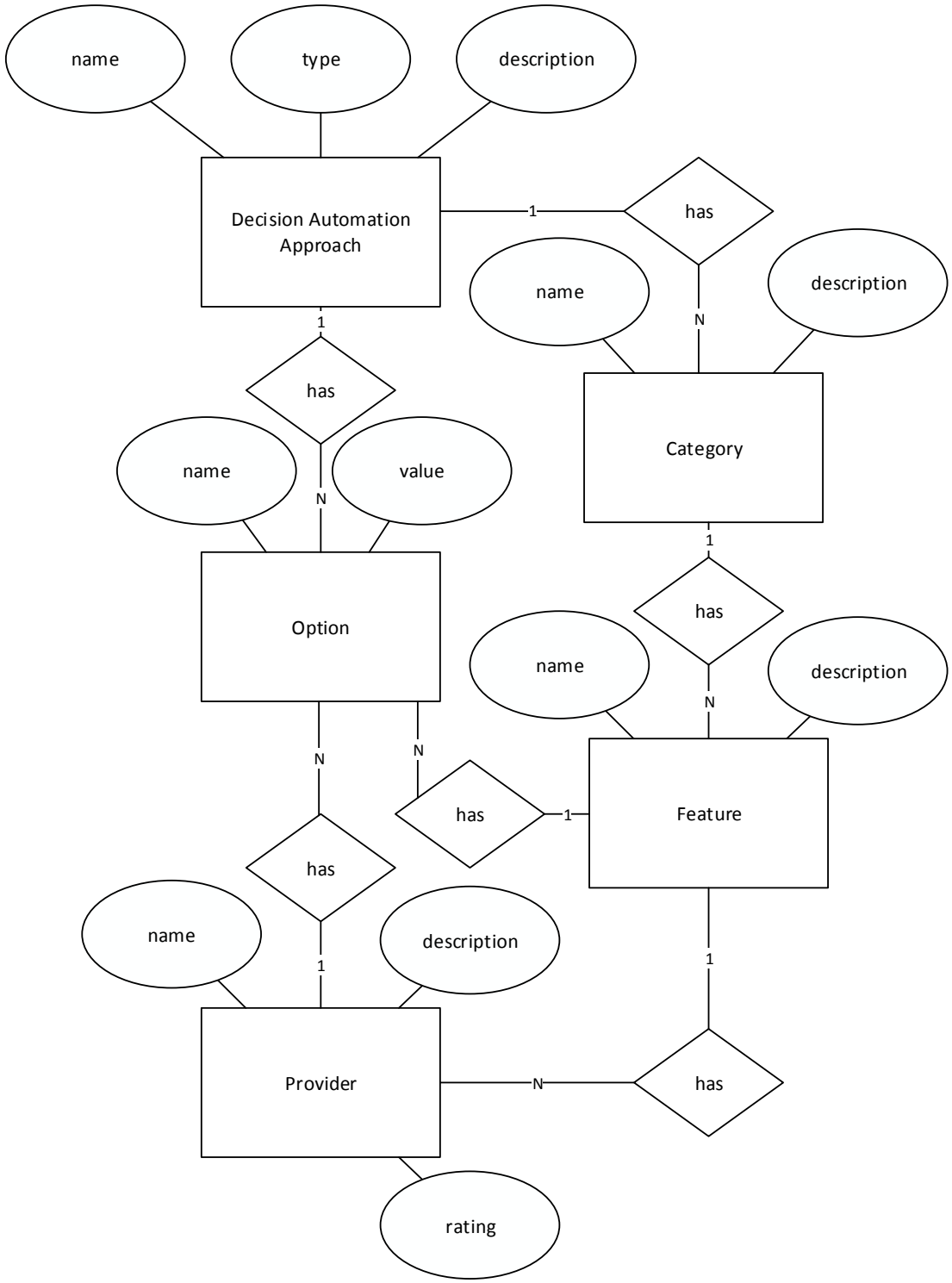

Figure 4.2: ER Diagram for description of Deployment Automation Approach

Data member	Key	Value
`daas`	Name of DAA	DAA object with the name
`categories`	`DAA` object	All the supported categories by this DAA
`features`	`Category` object	All features belongs to this Category
`providers`	`Feature` object	All providers of this Feature

Table 4.1: Key-Value Definition of Map object in class `KnowledgeBase`

Entity Option and Category-Feature-Provider are two ways to describe the features of DAAs. It's necessary to consider, which way is suit to describe the feature. For the same feature some DAAs describe detailed information about it, some DAAs not. In this work the comparison is important to make decision. To keep the feature of DAAs comparable each feature of DAAs should be described in same way, Option entity or Category-Feature-Provider, even if some DAAs have no enough information of the feature.

As mentioned before, the first step is initializing knowledge base. The knowledge base is loaded to decision support system. Figure 4.3 shows the class diagram of knowledge base in DSS. This DSS should have only one knowledge base. In package `knowledgebase`, the class `KnowledgeBase` is a *static* class. In class `KnowledgeBase` data member `kbPath` saves the path of the knowledge base documents. The procedure INIT() is used to initialize knowledge base. If the knowledge base documents are modified, procedure UPDATE() can be used to apply the change. After initialization all the loaded information from the external documents are saved in this class. `daas` saves all the DAAs. `categories` saves all available categories, as the same `features` for features, `providers` for providers. The data structure of `daas`, `categories`, `features` and `providers` is *Map*. The reason using *Map* instead of *List* or *Set* is, in implementation phase using key-value pair of *Map* is easy to find value without traversing. Table 4.1 shows the key-value definition of the map object in class `KnowledgeBase`.

Class `DAA` is the data structure of DAAs. `name`, `type` are basic information of the DAA. Unlike `categories` in class `KnowledgeBase`, `categories` in class `DAA` saves only the supported categories by this DAA. Its key-value definition is also different. The key of this `categories` is the name of category, value of this `categories` is the `Category` object with the name. `supportedIaaS` is a special, only for class `DAA`. Technically PaaS frameworks should be deployed on all IaaS. Some PaaS framework specified the supported IaaS. It could be considered, the guarantee of deployment to the supported IaaS. It's vantage to keep the deployment infallible.

Class `Category` is easy to understand. Like `DAA`, `name` and `description` are basic iniformation. `features` in class `Category` saves the features of this category. The key of this `features` is the name of feature, value of this `features` is the `Feature` object with the name. Class `Feature` and `Provider` are similar to `Category` except `rating` in `Provider`. `rating` is used to save the user rating of the `Provider`. It can help user to make choice.

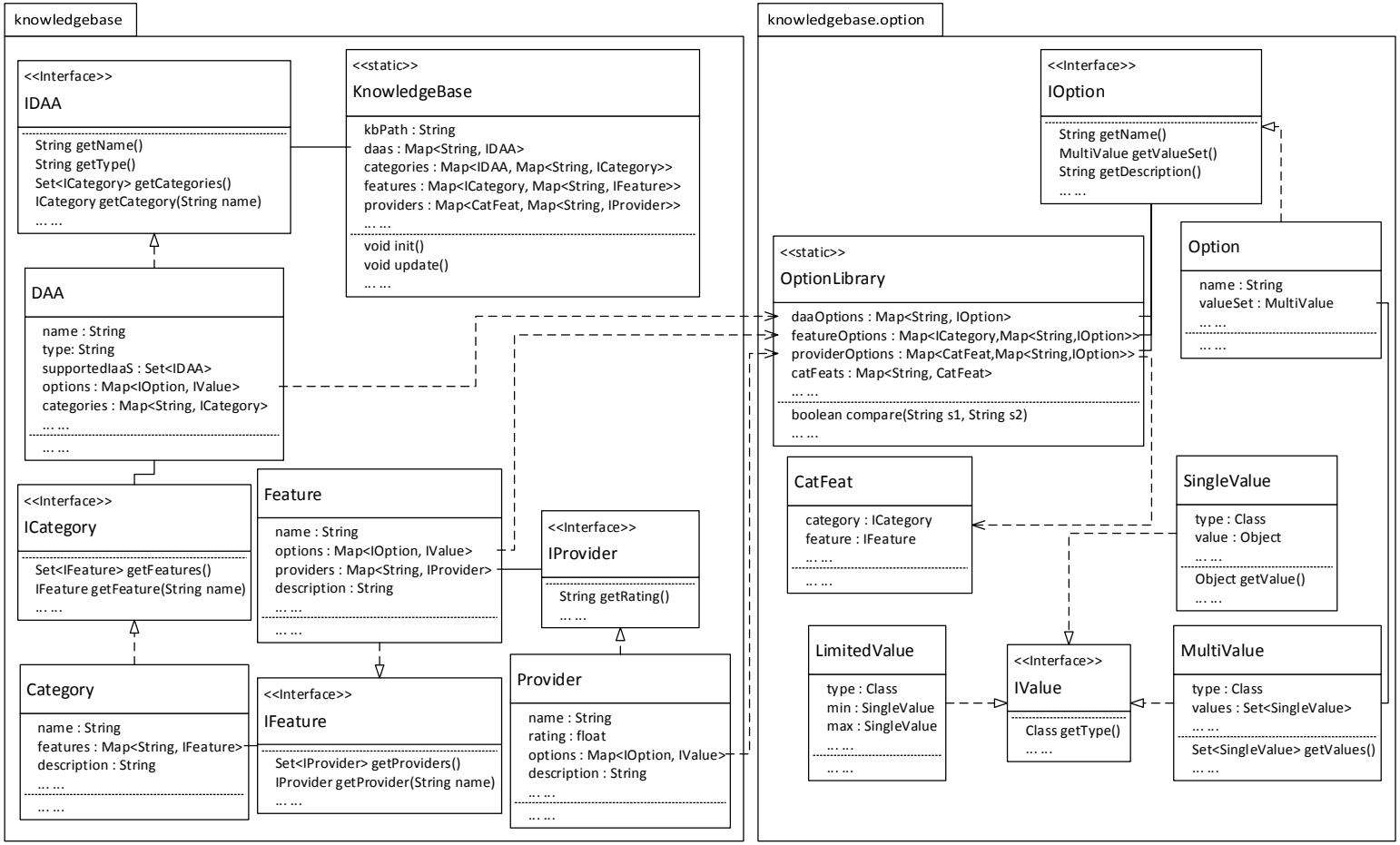

Figure 4.3: Class Diagram of Knowledge Base

Data member	Key	Value
`daaOptions`	Name of DAA	Option object with the name
`featureOptions`	`Category` object	All available option for the features of this category
`providerOptions`	`CatFeat` object	All available option for the providers of this CatFeat

Table 4.2: Key-Value Definition of Map object in class `OptionLibrary`

The package `knowledgebase.option` is used to model option of DAAs. As mentioned before, there are three categories of option, option for DAAs, option for features and option for providers. As a singleton class `OptionLibrary` saves all options. There are `daaOptions`, `featureOptions` and `providerOptions`. Their data structure is also *Map*. Table 4.2 shows key-value definition of map object in class `OptionLibrary`. Note, that the key of `featureOptions` is `Category` instead of `Feature`. There are no needs for each feature a set of option to generation. For example, Category "NoSQL Storage" has feature "MongoDB", "CouchDB" and "Redis". The three features are all NoSQL service. Most of their option could be duplicated. To avoid this case, a better solution is merging the option of all features of the category. As the same reason, the key of `providerOptions` is `CatFeat` object instead `Provider`. Class `CatFeat` is used to identify provider for different features. Because it possible, a provider has more features, which belongs to different category.

Class `Option` describes option, `name` as identifier and `valueSet` saves all acceptable values of the option. There are two classes to describe the value of option, `SingleValue` and `MultiValue`. `SingleValue` has only one value. Its `type` is not fixed. It can be any data type supported by the implementation language, like String, Boolean, Integer, Float, Double, Date, etc. `MultiValue` has a set of `SingleValue`. Note, that the type of `MultiValue` should as same as each `SingleValue` of it. Before add a `SingleValue` to a `MultiValue`, it must be sure, that the `type` of the `SingleValue` and the `MultiValue` are same. Otherwise the `SingleValue` can't be added to the `MultiValue`. Some cases should be noticed. For example, a `Multivalue` object has a `SingleValue` with `type` *String* and `value` 1. The type of this `MultiValue` is also *String*. Then another `SingleValue` with `type` *Integer* and `value` 2 is needed to add to the `MultiValue`. In this case, it cannot direct add the `SingleValue` 2 to the `MultiValue`. It should convert the `type` of `MultiValue` from *String* to *Integer* firstly, then add.

Class `LimitedValue` describes another set of value, value range. For example a option accepts a *Float* value range between 0.1 and 0.9, or a *Date* value range from today. The lower value saves in `min` and upper value saves in `max`. `min` and `max` are both `SingleValue`. Same as `MultiValue`, `type` of `min`, `max` and the `LimitedValue` must be identified.

Most data types are easy to compare, except *String*. The function COMPARE(STRING s1, STRING s2) in class `OptionLibrary` provides a solution to compare strings, and return a reasonable value. Table 4.3 shows some example of comparison. Note, that some cases are not reasonable because it's not easy to implement. For example, the reasonable result of comparison "Java 7" and "Java 7.0" should be *True*. But function COMPARE(STRING s1, STRING s2) returns a *False*.

$s1$	$s2$	$s1 == s2$
“Java 7”	“ Java 7 ”	True
“Java 7”	“Java7”	True
“Java 7”	“jAva 7”	True
“Java 7”	“ava 7”	False
“Java 7”	“Java 7.0”	False

Table 4.3: Example of Comparison of two Strings and the Result of function COMPARE()

The package `knowledgebase.option` provides a solution to model option and compare its value. `options` in class `DAA` is used for DAA option, in class `Feature` is for feature option, in class `Provider` is for provider option. All the `options` are used *Map* as its data type, `Option` object as its identifier and selected value saved as value of *Map*.

To implement knowledge base we can use XML, JSON, YAML. Compare to XML, JSON and YAML is easy to read and edit. To save same information, JSON and YAML need less storage than XML. YAML and JSON are similar. YAML can therefore be viewed as a natural superset of JSON, offering improved human readability and a more complete information model. This is also the case in practice; every JSON file is also a valid YAML file. This makes it easy to migrate from JSON to YAML if/when the additional features are required. [yam14] And web technologies, like AJAX, tend to use JSON. YAML is currently being used more for offline data processes. Knowledge base in this work is offline data process. Therefore we choose YAML to implement knowledge base.

Based on design of the knowledge base, List 4.2 shows part of knowledge base of Google App engine implemented by using YAML.

4.4 Decision Support Logic

Decision support logic has two components, find the suitable DAAs for the application and Ranking subsystem. Algorithmus 4.1 shows a procedure used to find suitable DAA for application. As the input data there are two variables. R contains the requirements of application. As mentioned in 4.2, the requirements R contains not only the requirements, the logical relation of requirements should also be described in R. L_d has all DAAs saved in knowledge base. Each DAA should be checked, if the DAA fulfills the requirements. As a result `suitable` contains all suitable DAAs for the application.

The Ranking subsystem is used to valuation the suitable DAAs depended on the requirements. It needs the information of requirements and features. Each features of DAAs has been evaluated by a point. The same feature in different DAAs may have not same point. The other side, requirements have different weights indicates the importance of each requirements for application. The deployment of application to DAA can be evaluated as :

Listing 4.2 Knowledge Base of Google App Engine using YAML

```
name: Google App Engine
type: Hosted PaaS
description: Google App Engine
options:
  Runtime Environments:
    - Java
    - Python
    - Go
    - PHP
  Filesystem Access: read only
categories:
  - SQL Storage:
      description: All the supported SQL database storage services from this Deployment
          Automation Approach
      features:
        - Google Cloud SQL:
            options:
              Sql support: true
              Supported sql dialects:
                - mysql
                - oracle
            providers:
              - Google:
                  description: A fully-managed web service that allows you to create,
                      configure, and use relational databases that live in Google's cloud.
                  rating:
                  Sql support: true
                  Supported sql dialects:
                    - mysql
                    - oracle
  - NoSQL Storage:
      description: All the supported NoSQL database storage services from this Deployment
          Automation Approach
      features:
        - Google Datastore:
            options:
            providers:
              - Google:
                  description: A schemaless object datastore providing robust, scalable
                      storage for your web application, a rich data modeling API, and a
                      SQL-like query language called GQL.
        - Google Blobstore:
            options:
            providers:
              - Google:
                  description: Allows your application to serve large data objects, such as
                      video or image files, that are too large for storage in the Datastore
                      service.
        - Google Cloud Storage:
            options:
            providers:
              - Google:
                  description: Lets your application read files from and write files to
                      buckets in Google Cloud Storage, with with internal error handling and
                      retry logic.
                        ...
```

Algorithm 4.1 Suitable DAA

```
procedure SUITABLEDAA(R,L_d)          // R denotes requirements of application
                                      // L_d denotes list of DAAs
    suitable ← ∅
    for all daa ∈ L_d do
        matched ← true
        for all r ∈ R do
            matched ← matched ∧ MATCHING(r, daa)
            if ¬(matched) then
                break;
            end if
        end for
        if matched then
            ADD(daa,suitable)
        end if
    end for
end procedure
```

Algorithm 4.2 DAA Ranking

```
procedure RANKING(R,L_d)              // R denotes requirements of application
                                      // L_d denotes list of DAAs
    score ← ∅
    for all daa ∈ L_d do
        sum ← 0
        for all r ∈ R do
            sum ← sum + r.GETWEIGHTS() × daa.GETSEVERITY(r)
        end for
        score.PUT(daa,sum)
    end for
    score.SORT()
end procedure
```

$$score = \sum_{i=1}^{n} w_i \times p_i$$

$with$ w_i : Weights of requirement i
p_i : Point of requirement i, which specified in DAA

As algorithm 4.2 shows, the function GETWEIGHTS() is used to get the weights of requirements. GETSEVERITY() can get the point of features. After calculate the evaluation of DAAs, the list should be sorted in descending order.

5 Evaluation

This chapter provides evaluation of the system. It must be ensured, that the system requirements specified in Chapter 3 are fulfilled in the design and implementation phases. Some cloud providers and test cases are used to validate the results.

There is a variety of cloud providers on the market. Some of them are chosen as knowledge base. Table 5.1 shows four options of knowledge base.

- **Out-of-the-box Supported Hosting Provider**
  Sometimes the requirements of application are not fulfilled by a single DAA. In this case, the possible solution is a combination of a PaaS and IaaS. Technically the PaaS framework can by deployed on all IaaS platform. This option shows the tested IaaS from official documentation.
- **Runtime Environment**
  Application needs runtime environment supported by IAAs, such as Java, PHP, Python, Ruby, etc.
- **Filesystem Access**
  Application has different requirements of filesystem access, read only, read/write.
- **Control over Infrastructure**
  If DAA can control the infrastructure. Normally IaaS has high ability of infrastructure control, Hosted-PaaS has no control of infrastructure.

Table 5.2 shows the features of knowledge base. The features of PaaS providers are mostly official specified. The features of IaaS providers are extended as plug-ins. Note, that the knowledge base has only deployment-related information of DAAs. The documentation of some DAAs is not in detail.

	Google App Engine	**Heroku**	**CloudFoundry**	**Stratos**	**Chef**	**Puppet**
Out-of-the-box Supported Hosting Providers			Amazon Web Services EC2, OpenStack, VMware vSphere, vCloud Director, Chef	SUSE Cloud, other OpenStack-based offerings, VMware, Eucalyptus, and Amazon Elastic Computing Cloud (EC2), vCloud, jclouds		
Runtime Environments	Java 6, Java 7, Python 2.6, Python 2.7, Go, PHP	Ruby, Java, Python, Clojure, Scala, Node.js, Play Framework, Spring, Rails, Django, Facebook App	Java, Spring, Ruby, Node.js	php	PHP, Java, Node.js, Ruby, JRuby, Django, R, Perl, ...	PERL, Java, PHP, Node.js, Python, Ruby, ...
Filesystem Access	read-only	read-only	read/write	read/write	read/write	read/write
Control over Infrastructure	no control	no control	low	low	high	high

Table 5.1: Option of Knowledge Base

	Google App Engine	Heroku	CloudFoundry	Stratos	Chef	Puppet
SQL Storage Services	Google Cloud SQL	MySQL (ClearDB), Heroku Postgres, Amazon RDS	MySQL (ClearDB), PostgreSQL (ElephantSQL)	MySQL	MySQL, PostgreSQL, Oracle, SQL Server, ...	MySQL, PostgreSQL, SQLite
NoSQL Storage Services	Google Datastore, Google Blobstore, Google Cloud Storage	MongoDB (MongoHQ), CouchDB (Cloudant), Redis (Redis To Go, Redis Cloud, RedisGreen, openredis), Neo4j, Hadoop (Treasure Data)	Redis (Redis Cloud), MongoDB (MongoLab)		MongoDB, Hadoop, CouchDB, Redis, ...	MongoDB, Redis,
Web Server		Nginx			Apache, Nginx, lighttpd, JBoss, IIS, Zend, ...	Apache, Nginx, lighttpd, JBoss, IIS, Zend, ...
Queueing Services		IronMQ, RabbitMQ, ...	RabbitMQ (CloudAMQP)		RabbitMQ, ZeroMQ, ActiveMQ	RabbitMQ, ActiveMQ,
Cache Services	Google Memcache	Memcachier(addons), IronCache			memcached	memcached
Data Processing Services	Google Images, Google MapReduce				imagemagick, pdf2image	imagemagick
Mail Services	Google Mail, SendGrid Email (SendGrid)	SendGrid Email (SendGrid), Mailgun, CloudMailIn, ...	SendGrid		sendmail	Postfix, sendmail
Dev & Monitoring Services	Google Logs	App logs, System logs, API logs, PG Backups, ...	BlazeMeter, Load Impact, New Relic, Component Logging, Database Migrations, Log Aggregation		logwatch, rsyslog,	rsyslog, monitor, automysqlbackup

Table 5.2: Features of Knowledge Base

Requirements	PHP	MySQL	Result
Google App Engine	True	False	False
Heroku	False	True	False
CloudFoundry	False	True	False
Stratos	True	True	True
Chef	True	True	True
Puppet	True	True	True

Table 5.3: Decision Support for Deployment of Wordpress

Requirements	PHP (r_{a1})	MySQL (r_{b1})	MSSQL (r_{b2})	PostgreSQL (r_{b3})	Apache (r_{c1})	Nginx (r_{c2})	IIS (r_{c3})	Result
Google App Engine	True	False	False	False	False	False	False	False
Heroku	False	True	False	True	False	True	False	False
CloudFoundry	False	True	False	True	False	False	False	False
Stratos	True	True	False	False	False	False	False	False
Chef	True	True	True	True	True	True	True	True
Puppet	True	True	False	True	True	True	True	True

Table 5.4: Decision Support for Deployment of Joolma

Test Case: Wordpress

The requirements of Wordpress are PHP and MySQL. [joo14] Based on decision support logic Table 5.3 shows the decision support for deployment of Wordpress. The result is Stratos, Chef and Puppet. As the most used blog service, Wordpress can be also deployed on Google App Engine, Heroku and CloudFoundry. The requirements of Wordpress are not directly fulfilled by the three DAAs, which means more operation are needed to deploy Wordpress on GAE, Heroku and CloudFoundry. Therefore the system suggests using other DAAs to deploy Workpress.

Test Case: Joolma

As mentioned before, the requirements of Joolma are : [joo14]

- **Software**: PHP
- **Supported Databases**: MySQL, MSSQL, PostgreSQL
- **Supported Web Servers**: Apache, Nginx, Microsoft IIS

Table 5.4 shows the decision support for deployment of Joolma. The logical expression of requirements is

Requirements	Ruby (r_{a1})	JRuby (r_{a2})	Rails (r_{b1})	MySQL (r_{c1})	PostgreSQL (r_{c2})	SQL Server (r_{c3})	SQLite (r_{c4})	Result
Google App Engine	False	False	False	False	False	False	False	False
Heroku	True	False	True	True	False	False	False	True
CloudFoundry	True	False	False	True	False	False	False	False
Stratos	False	False	False	True	False	False	False	False
Chef	True	True	True	True	True	True	False	True
Puppet	True	False	False	True	True	False	True	False

Table 5.5: Decision Support for Deployment of Redmine

$$r_{a1} \wedge (r_{b1} \vee r_{b2} \vee r_{b3}) \wedge (r_{c1} \vee r_{c2} \vee r_{c3})$$

Based on decision support logic the results is Chef and Puppet. The same as Wordpress, Chef and Puppet have fulfilled the requirements. Official, the other DAAs have not fulfilled the requirements. Because of the reliability and dependency of deployment, other DAAs are not suggested.

Test Case: Redmine

Redmine[1] is a flexible project management web application. Written using the Ruby on Rails framework, it is cross-platform and cross-database. The reuirements are : [red14]

- Runtime: Ruby Version: Ruby, JRuby. Rails
- Database: MySQL, PostgreSQL, Microsoft SQL Server, SQLite

Table 5.5 shows the decision support for deployment of Redmine. The logical expression of requirements is

$$(r_{a1} \vee r_{a2}) \wedge r_{b1} \wedge (r_{c1} \vee r_{c2} \vee r_{c3} \vee r_{c4})$$

Based on decision support logic the matched results are Chef and Heroku. The same as Wordpress, Chef and Puppet have fulfilled the requirements. Other DAAs have not enough official support for the requirements. Other DAAs are not suggested by the system.

Requirements	Node.js	Result
Google App Engine	False	False
Heroku	True	True
CloudFoundry	True	True
Stratos	False	False
Chef	True	True
Puppet	True	True

Table 5.6: Decision Support for Deployment of Etherpad

Test Case: Etherpad

Etherpad[2] is a really-real time collaborative editor maintained by the Etherpad Community. It's written in Javascript. For deployment we notice on the server side it's only a Node.js runtime needed. To find the suitable DAAs, the requirement should be matched with the knowledge base. As Table 5.6 shows, the results are Heroku, CloudFoundry, chef and Puppet.

[1] http://www.redmine.org/
[2] https://github.com/ether/etherpad-lite

6 Summary and Future Work

Cloud computing is an opportunity for enterprises. It decreases IT costs, increases profit. It's also a challenge. Unlike traditional IT architecture Cloud computing is a different consumption and delivery model. Application should adapt the new model to deliver service. A variety of application are needed to deploy to Cloud. There are also plenty of Cloud services on the market. It's necessary to help the consumer make the suitable choice to deploy their application. Therefore this work aims at realizing a Decision Support System for different Deployment Automation Approaches (DSS4DAA).

Chapter 2 present the necessary background about the technologies used in this work. The concrete Cloud services are categorized in three categories, Hosted-PaaS, PaaS Framework and IaaS. After researching on the feature similarities and differences of Cloud services, a description of DAA is specified. Based on the specification a DSS is designed. The DSS contains knowledge base, decision support logic and user interface. In order to update the information of DAAs without modification of implementation of DSS, the collected data and information of DAAs are saved in external document, its implementation uses YAML. Using external documents save information of DAAs. The components of DSS are implemented as module, which can be reused. The system are implemented using restful API to provide service. At the end serveral DAAs and application are chosen to evaluate the system.

Future Work

This work has proposed the basic rules of a decision support system for deployment automation approaches. It can be more specific. The following features are identified for future work:

1. User Interface
   The knowledge base and decision support logic are implemented in this work. The other component of system, user interface is not implemented. The API for UI is defined as restful service endpoint. It can be used for data exchanging between GUI and other components.

2. Extension of DAAs in the knowledge base
   The implemented knowledge base contains six DAAs by now. It should be extended with more DAAs for a further wide field of application. And features of each DAAs should be specified in detailed. The update of knowledge base can be manual, or semi-automatic solution.

3. Comparison of options of features and providers
   In system design there are options for features and providers defined. It should be comparable to make user clear, the difference of features and providers between DAAs. The system may suggest more than one DAAs to user. It can help user to make choice from the result.

4. Point setting of ranking subsystem
   In ranking subsystem we define point for options, features and providers. By now they are all set as the default value. It should be set reasonable. The criteria of setting should be described.

5. Finer description of application requirements
   The idea of system design is based on the research of DAAs. The all available options, features and providers are saved in knowledge base. The other side, the requirements of application should be also refined, like categorized as Game Apps, Communication Apps, Picture Handling Apps, Audio Apps, Geolocation Apps, ... Each category of application should have similar requirements, which can be saved as part of knowledge base. And requirments of each category of application could be pre-matched with DAAs before system is used. It makes the decision support logic as a two level architecture, pre-match for each category of application, match for the application. It can make the system more effective.

Bibliography

[Cho12] D. Chou. Three basic Cloud Service Models. http://blogs.msdn.com/b/dachou/, 2012. [Online; accessed December-2013]. (Cited on pages 6 and 15)

[Eng12] G. A. Engine. Overview of App Engine Features. https://developers.google.com/appengine/features/, 2012. [Online; accessed December-2013]. (Cited on page 16)

[Hes12] A. Hesseldahl. Oracle CEO Larry Ellison: Dog Fight in the Cloud. http://allthingsd.com/20120530/oracle-ceo-larry-ellison-live-at-d10/, 2012. [Online; accessed December-2013]. (Cited on page 9)

[JCl14] JClouds. APACHE JCLOUDS SUPPORTED PROVIDERS. http://jclouds.apache.org/documentation/reference/supported-providers/, 2014. [Online; accessed January-2014]. (Cited on page 19)

[joo14] joolma.org. Joolma REQUIREMENTS. http://www.joomla.org/technical-requirements.html, 2014. [Online; accessed January-2014]. (Cited on pages 31 and 44)

[KM78] P. Keen, M. Morton. *Decisión support systems: an organizational perspective*. Addison-Wesley series on decision support. Addison-Wesley Pub. Co., 1978. URL http://books.google.de/books?id=iQtPAAAAMAAJ. (Cited on page 22)

[MG11] P. Mell, T. Grance. The NIST definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011. (Cited on page 13)

[ops14] opscode.com. An Overview of Chef. http://docs.opscode.com/chef_overview.html, 2014. [Online; accessed 06-January-2014]. (Cited on pages 6, 19 and 20)

[pan14] pancake.com. Pancake REQUIREMENTS. http://help.pancakeapp.com/customer/portal/articles/602921-server-requirements, 2014. [Online; accessed January-2014]. (Cited on page 26)

[Pow02] D. Power. *Decision Support Systems: Concepts and Resources for Managers*. Quorum Books, 2002. URL http://books.google.de/books?id=9NA6QMcte3cC. (Cited on page 22)

[pup14] puppetlabs.com. What is Puppet. http://puppetlabs.com/puppet/what-is-puppet, 2014. [Online; accessed January-2014]. (Cited on pages 6 and 21)

[red14] redmine.org. Redmine REQUIREMENTS. http://www.redmine.org/projects/redmine/wiki/RedmineInstall, 2014. [Online; accessed January-2014]. (Cited on page 45)

[Son13] Z. Song. *A decision support system for application migration to the Cloud*. Master's thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2013. URL http://elib.uni-stuttgart.de/opus/volltexte/2013/8262. (Cited on page 25)

[wor12] wordpress.org. WordPress Requirements. http://wordpress.org/about/requirements/, 2012. [Online; accessed December-2013]. (Cited on pages 10 and 31)

[WSO14a] WSO2. Stratos 2.0 Architecture. http://docs.wso2.org/display/Stratos200/Architecture, 2014. [Online; accessed January-2014]. (Cited on pages 6 and 18)

[WSO14b] WSO2. WSO2 Carbon. http://wso2.com/products/carbon/, 2014. [Online; accessed January-2014]. (Cited on page 18)

[yam14] yaml.org. YAML 1.2 Specification. http://www.yaml.org/spec/1.2/spec.html, 2014. [Online; accessed January-2014]. (Cited on page 38)

[Zim09] O. Zimmermann. *An architectural decision modeling framework for service-oriented architecture design*. Ph.D. thesis, Stuttgart, Univ., Diss., 2009, 2009. (Cited on page 23)

[Zim13] O. Zimmermann. Cloud Computing – Aus der Sicht des Anwendungsarchitekten, 2013. (Cited on pages 6, 23 and 24)

All links were last followed on Januar 31, 2014.

Decleration

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.

---

place, date, signature