Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis No. 0838-004

Persistence, Discovery, and Generation of Viable Cloud Application Topologies

Abhilash Mishra

Course of Study:	INFOTECH
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Santiago Gómez Sáez
Commenced:	June 08, 2015
Completed:	December 08, 2015
CR-Classification:	C.2.4; D.2.0; D.2.11

Abstract

Cloud computing is gaining popularity among application developers with each passing day because of the many benefits introduced by the cloud paradigm. Also the number and range of available cloud services is continuously increasing and hence the application developers are willing to migrate complete or partial applications to cloud environment. Applications can be designed to be run on the cloud, and utilize its technologies, or can be partially or totally migrated to the cloud. The application's architecture contains three layers: presentation, business logic, and data layer. Each of this layers can be deployed on a different cloud service basing on the requirements and the constraints and it is also possible to deploy a layer of the application on an on premise physical server if required and topologies need to be designed for the deployment of the application.

There are standards like TOSCA, or approaches like GENTL and MOCCA which allow for the modeling and management of application topologies. Cloud application topologies can be defined as typed labeled graphs constituted by a set of nodes, edges, and labels. Nodes represent the application components, while the edges depict the relationship among them and sub-topologies can be built by grouping multiple nodes. A cloud application viable topology is a feasible distribution of the application components according to the depicted or discovered application independent sub-topologies. Some objectives like cost, performance, etc also need to be considered in order to help analyze the fitness of the services for the desired operation.

This Master Thesis focuses on optimally distributing application components across cloud offerings efficiently. More specifically this work deals with providing a topology modeling framework capable of supporting the following three fundamental aspects: (i) leveraging existing technologies and mechanisms for analyzing the different aspects of the evolution of cloud application topologies, (ii) design and develop the concepts and mechanisms towards dynamically discovering and constructing cloud application viable distributions (viable topologies) specifications (typically XML representations) in an optimal manner, and (iii) developing the visualization means within an existing topology modeling environment.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Cloud computing has gained popularity in the recent times because of its advantages such as pay per use, rapid scalability, resource pooling and broad network access which helps the users to reduce capital and maintenance costs and also reducing business risks and maintaining expenses. This has also changed the way the computing resources were used and provided by offering them as high quality and highly available services to the users on demand as metered services. Nowadays more and more applications are partially or fully migrating to cloud environment given the rise in the number and variety of cloud services by various cloud providers. Cloud providers target to maximize their benefits by maximizing the resources usage with a minimum management effort or human interaction, while the cloud consumers can significantly reduce their capital expenditures in their IT infrastructure by outsourcing the demanded computational and storage resources to a cloud environment.

In the following sections we discuss the problem statement and motivating scenario this thesis relies on.

1.1 Problem Statement

Due to the various benefits of the cloud paradigm, there is a constant growth in the number of people willing to use it to host their applications. In order to achieve this, there can be two scenarios, either existing applications would be migrated to the cloud environment or applications are built and deployed as per the offerings available in the cloud. In either of this cases, the developer needs to know how he can deploy the applications such that he gets an optimal performance as per his requirements and constraints. This may be achieved by deploying the whole application in one particular cloud offering or deploying parts of it on various cloud offerings and also keep some part of it on premise within their physical boundaries. Topology needs to be designed for this which would tell the developers what is the best way to deploy their application.

As shown in Figure 1.1, the applications can be deployed on the various services available off premise on cloud providers like Amazon[1], Rackspace[2], Google[3] or on premise on physical servers. In order to provide the application developers with optimal topologies to deploy their application, all the available on premise as well as off premise alternatives need to be analyzed to check their suitability to deploy parts of an application as per the requirements. This masters thesis focuses on the discovery of optimal set of services for deploying each part of the application and building set of topologies with this services and providing them

[1]https://aws.amazon.com
[2]http://www.rackspace.com/cloud
[3]https://cloud.google.com

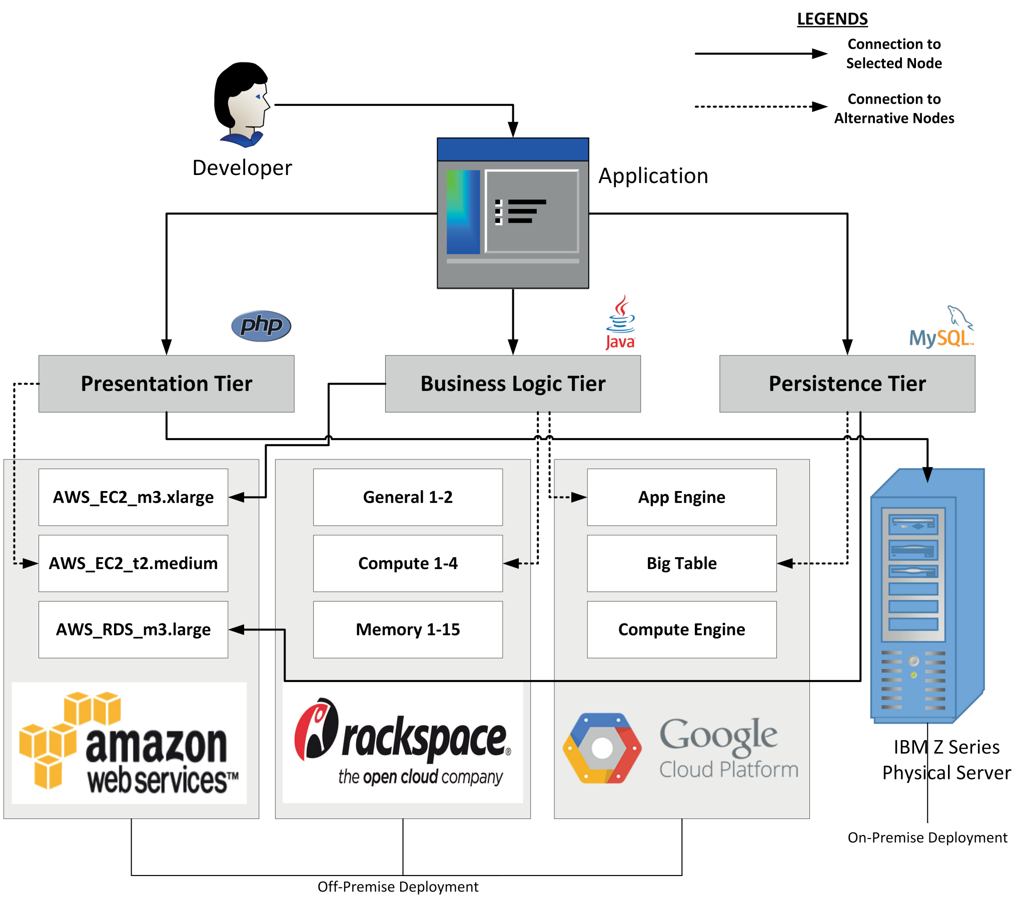


Figure 1.1: Application deployment alternatives to be analyzed and optimized

to the application developer. We have studied various existing optimization techniques for the purpose of finding optimal set of services and decided on the best one. The application developer can then choose which topology they would like to use for their application and that would be persisted so that it could be used by another user who has similar requirements. The interface of the topology modeling tool would be used to accept the users requirements, enrich topologies and build topologies from the optimal set of services. It would get the data regarding all the available cloud services from the knowledge base and uses it for the optimization process.

1.2 Motivating Scenario

Application developers as well as companies aim to minimise their initial investments as well as maintenance costs by deploying their applications in the cloud. It also makes their tasks easier as it can dynamically scale up or scale down depending on the workload on the hosting servers. The deployment of applications partially or completely in cloud environments involves the need of topologies to guide how the applications should be deployed.

Optimal distribution of applications in the cloud environment is important in order to make the most of cloud services to deploy applications as per the users requirements and constraints. The performance of an application as well as the cost of hosting it can vary significantly depending on the resources used to host various components of the application. Also it is not feasible for each and every application developer to take a look at all the available services and evaluate their feasibility as per the requirements on their own. Toward this goal, a number of approaches provide decision support for migrating existing applications to the cloud which are discussed in Chapter 3 but they do not consider as part of their process the application topology and the possibility of deploying parts of the application on physical servers. The idea for this has been introduced in [ASLW14] and we will be extending it in this thesis.

1.3 Definitions and Conventions

In the following section we list the definitions and the abbreviations used in this diploma thesis for understanding the description of the work.

List of Abbreviations

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

API	Application Programming Interface
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
CACTOS	Context-Aware Cloud Topology Optimization and Simulation
CSAR	Cloud Service Archive
CSL	Cloud Service Lifecycle
CDO	Cloud Deployment Option
GA	Genetic Algorithm
GENTL	Generalized Topology Language
HATEOAS	Hypermedia as the engine of application state
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IT	Information Technology
JSON	JavaScript Object Notation
XML	EXtensible Markup Language

MAGA	Multi-Agent Genetic Algorithm
MOCCA	MOve to Clouds for Composite Applications
NI	Number of Instances
NIST	National Institute of Science and Technology
NSGAII	Non-dominated Sorting Genetic Algortithm - II
OS	Operating System
PaaS	Platform as a Service
PDAs	Personal Digital Assistants
PI	Provider Instance
SA	Simulated Annealing
SaaS	Software as a Service
SN	Service Number
SPL	Software Product Line
ST	Service Type
SLA	Service Level Agreement
TGI	Type Graph with Inheritance
TOSCA	Topology and Orchestration Specification for Cloud Applications
TOSCA-MART	TOSCA-based Method for Adapting and Reusing application Topologies
VM	Virtual Machine

1.4 Outline

The remainder of this document is structured as follows:

- Fundamentals, Chapter 2: provides the necessary background on the different concepts, technologies, and tools used in this masters thesis.
- Related Works, Chapter 3: discusses relevant State of the Art and positions our work towards it.
- Concept and Specification, Chapter 4: explanation of the concepts used, functional and non-functional requirements and use cases are discussed in this section.
- Design, Chapter 5: gives a detailed overview of the architecture of the system, and the needed extensions to the already existing ones.

- Implementation and Validation, Chapter 6: the implemented components, as well as the necessary extensions or changes are detailed in this section from the point of view of coding and configuration and tests the prototype based on the scenario described in this document.
- Outcome and Future Work, Chapter 8: provides a conclusion of the developed work and investigate some ideas in which this master thesis can be extended.

2 Fundamentals

2.1 Cloud Computing

2.1.1 Definition and Motivation

According to the National Institute of Science and Technology (NIST) [MG11] definition, *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, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Cloud computing has adopted many things from utility computing, grid computing and cluster computing. Utility Computing[BVB08] offers IT resources as services on demand and metering service is introduced where service users are charged only the amount of resources they used. This is used by cloud computing as cloud offers everything as a service to cloud customers on pay per use basis. Cluster Computing [Ste02] combines multiple identical computing resources within the same network to perform as a single system hence leading to high availability and redundancy in the system. The members of a cluster have identical configuration and performance properties and are tightly coupled. Grid Computing [BFH03] is loosely coupled and more distributed type of cluster computing. Cloud Computing facilitates us to access data and applications remotely over the network, instead of our local machines or on-premise resources. Cloud based computing resources are located outside or inside the premises of the organization and resources are provisioned and released dynamically according to the cloud consumers' needs with minimum involvement of cloud providers and little overhead. Cloud software takes full advantage of the cloud paradigm by being service-oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability [BGPCV11].

The emergence and growth of cloud computing has brought a significant change in the Information Technology (IT) industry over the past few years. A number of large companies such as Google, Amazon and Microsoft strive to provide more powerful, reliable and cost-efficient cloud platforms, and business enterprises seek to rethink their business models to benefit from the cloud paradigm. Cloud computing provides several compelling features that motivates business owners to use it, as shown below [ZCB10].

- No up-front investment: Cloud computing uses a pay-per-use pricing model. A service provider does not need to invest in the infrastructure to start using the facilities of cloud computing. Rather, it rents resources from the cloud according to its own needs and pay for the usage [ZCB10].

- Lowering operating cost: The allocation and de allcation of resources in a cloud environment can be done rapidly, on demand. Hence, a service provider no longer needs to provision capacities according to the peak load and waste resources in case of low service demands. It helps to conserve resources by releasing them when not required and hence reduces operating cost [ZCB10].

- Reducing business risks and maintenance expenses: By outsourcing the service infrastructure to the clouds, a service provider shifts its business risks (such as hardware failures) to infrastructure providers, who often have better expertise and are better equipped for managing these risks. In addition, a service provider can cut down the hardware maintenance and the staff training costs [ZCB10].

- Capacity planning, [All08] is one of the biggest chanllenge for an organization as it involves the assessment of future demands of IT resources of an organization to provide optimized performance mainly because it is very difficult to determine the maximum amount of workload a computing resource can be able to handle. So often organizations allot more resources than required called over-provisioning causing wastage of resources and increasing the cost, or they allot less resources than the requirement known as under-provisioning which leads to unavailability or poor performance. Cloud computing removes the need of capacity planning as it dynamically allocates or provisions more cloud resources and releases the resources as well according to the dynamically increasing or decreasing demand. This characteristic of the cloud computing is known as elasticity [Inc15]. Horizontal scaling is very frequently used in cloud environment which involves scaling out (allocation) and scaling in (de allocation) of resources dynamically.

The cloud model is composed of five essential characteristics, three service models, and four deployment models[MG11] which can be seen in figure 2.1.

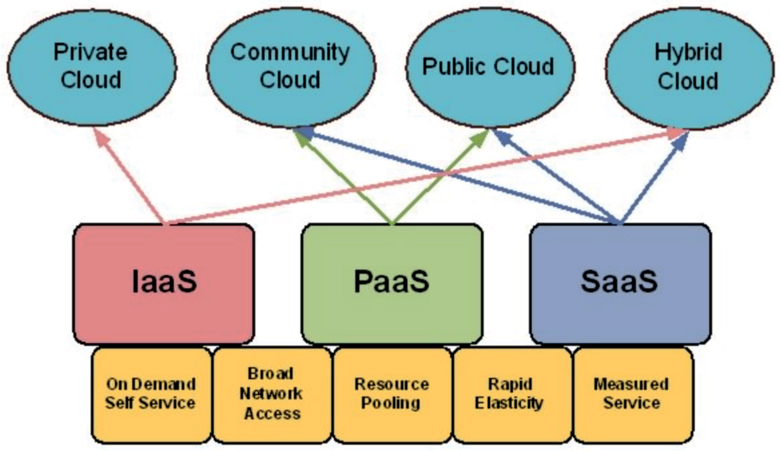


Figure 2.1: Cloud Model [Bis11]

2.1.2 Essential Characteristics

The 5 essential characteristics which define cloud computing in a better way are :

- On-demand self-service: A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider [BGPCV11].
- Broad network access: Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and Personal Digital Assistants (PDAs)) [BGPCV11].
- Resource pooling: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the subscriber generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data center). Examples of resources include storage, processing, memory, network bandwidth, and Virtual Machine (VM) [BGPCV11].
- Rapid elasticity: Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time [BGPCV11].
- Measured Service: Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service [BGPCV11].

2.1.3 Service Delivery Models

There are 3 basic service delivery models which defines how and at what levels the services are delivered by a cloud provider.

- Software as a Service (SaaS): The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a Web browser. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings [BGPCV11].

- Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or -acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations [BGPCV11].
- Infrastructure as a Service (IaaS): The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components [BGPCV11].

2.1.4 Deployment Models

The 4 cloud deployment models represent different types of cloud environments categorized by ownership, size, and access.

- Private cloud: The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise [BGPCV11].
- Community cloud: The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise [BGPCV11].
- Public cloud: The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services [BGPCV11].
- Hybrid cloud: The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds) [BGPCV11].

2.1.5 Layered Architecture of Cloud Computing

The architecture of a cloud computing environment can be broadly divided into 4 layers: the hardware layer, the infrastructure layer, the platform layer and the application layer, as shown in figure 2.2. Compared to traditional service hosting environments such as dedicated server farms, the architecture of cloud computing is more modular. Each layer is loosely coupled with the layers above and below, allowing each layer to evolve separately.[ZCB10]

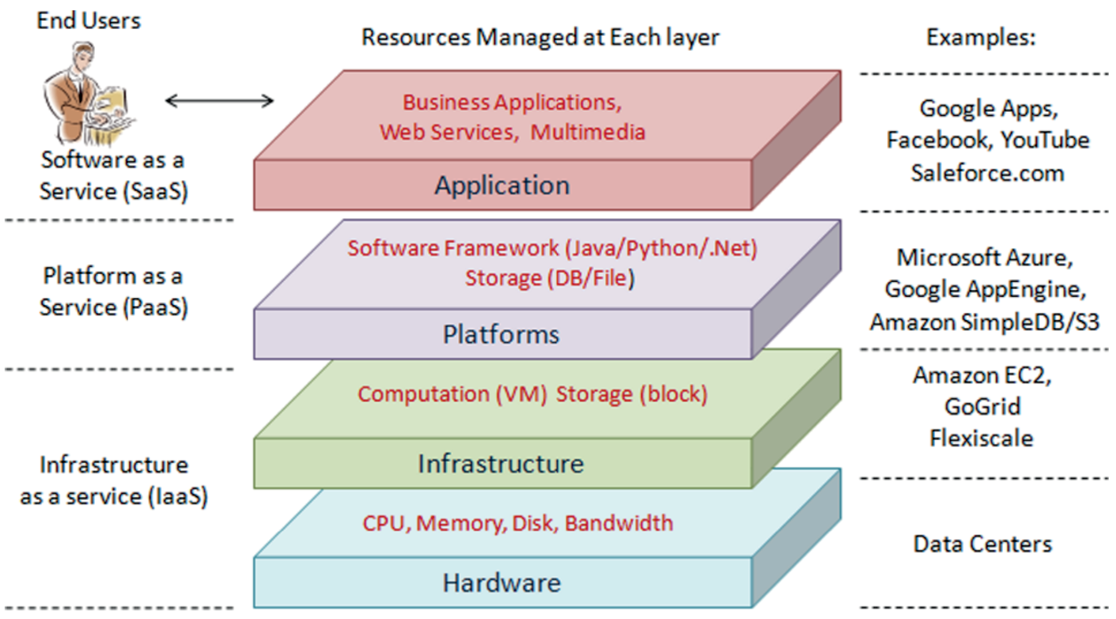

End Users
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a service (IaaS)
Resources Managed at Each layer
Business Applications, Web Services, Multimedia
Application
Software Framework (Java/Python/.Net) Storage (DB/File)
Platforms
Computation (VM) Storage (block)
Infrastructure
CPU, Memory, Disk, Bandwidth
Hardware
Examples:
Google Apps, Facebook, YouTube Saleforce.com
Microsoft Azure, Google AppEngine, Amazon SimpleDB/S3
Amazon EC2, GoGrid Flexiscale
Data Centers

Figure 2.2: Cloud computing architecture [ZCB10]

- The hardware layer: This layer is responsible for managing the physical resources of the cloud, like servers, routers, switches, power and cooling systems and is typically implemented in data centers which contains thousands of interconnected servers. The issues which may arise at hardware layer are hardware configuration, fault-tolerance, traffic management, power and cooling resource management [ZCB10] .
- The infrastructure layer: The infrastructure layer uses virtualization technologies to creates a pool of storage and computing resources by partitioning the physical resources and is also known as the virtualization layer. This layer is responsible for many key features, such as dynamic resource assignment which are only made available through virtualization technologies and hence is an essential component. [ZCB10] .
- The platform layer: It is built on top of the infrastructure layer and consists of operating systems and application frameworks. The purpose of the platform layer is to minimize the burden of deploying applications directly into VM containers. For example, Google App Engine operates at the platform layer to provide API support for implementing storage, database and business logic of typical web applications [ZCB10] .
- The application layer: It is at the top of hierarchy and consists of the actual cloud applications which can leverage the automatic-scaling feature to achieve better performance, availability and lower operating cost[ZCB10] .

2.1.6 Cloud Service Lifecycle (CSL)

The CSL consists of nine stages: Deployment, User Requirements, Matchmaking, Negotiation, Execution, Monitoring, Analyzing, Adjusting, and Ending which can be seen in the figure 2.3.

In the Deployment stage all information about a service in a service description is collected by a provider. By publishing the description a service is registered and potential consumers can find it. A service can be deployed after the consumer has sent a request for it. A service is supplied to the market once all related information is published on the Internet. During the next stage the User Requirements and priorities for a service are specified by the consumer. In this stage, the consumer details the technical and functional specifications that a service needs to fulfill. [MTS13]

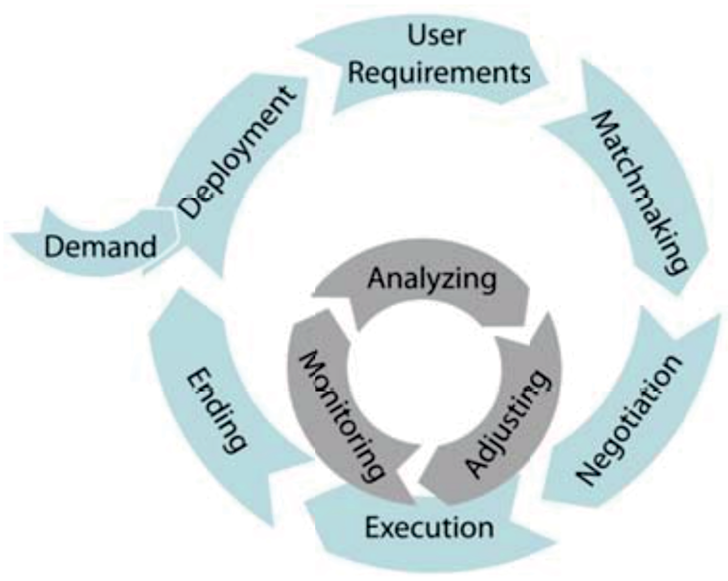


Figure 2.3: Cloud Service Lifecycle.[MTS13]

After the consumer has sent his search request a list of matching services is delivered which is called Matchmaking. From the returned list of matching services the consumer picks the most fitting one. For this service he may negotiate a Service Level Agreement (SLA) and specify the desired guarantees for the service and then the provider replies if he can fulfill them. The SLA Negotiation is similar to the Service Negotiation in the life cycles. The Execution stage is entered once the SLA is concluded where the service is activated before its execution. The Monitoring, Analyzing, and Adjusting stages are looped through during the execution. The cloud service is monitored permanently and performance data of specific service features are analyzed. The measured values are compared with their contractually agreed value qualities and if a value is not in line with its guaranteed quality a message to the Adjusting or Ending stage is triggered to start problem solving activities. In the Adjusting stage the infrastructure of a service can scale rapidly during run-time if necessary. In the Ending stage the costs for the service execution are billed, and the service is rated by the consumer.[MTS13]

2.2 Cloud Application Topology Specification

The cloud application topology specification is done by using various languages which are used to represent the structural components and their relationships as required for the application. The topology model representation helps to easily port the applications and deploy them on multiple cloud vendors. The cloud application topology and different topology specification languages such as TOSCA and GENTL are explained in the following

2.2.1 Cloud Application Topology

Application Topology is the graphical representation of the set of structural components and relationships among them which are used to build multi-tier distributed applications[Sta13]. The interaction among the various structural components, dependency among them and how they are deployed on cloud environment is defined in the Application topology model. Application components and their relationships are called elements of application topology. A Visual Notation for application topologies based on the TOSCA [Sta13] is shown in figure 2.4.

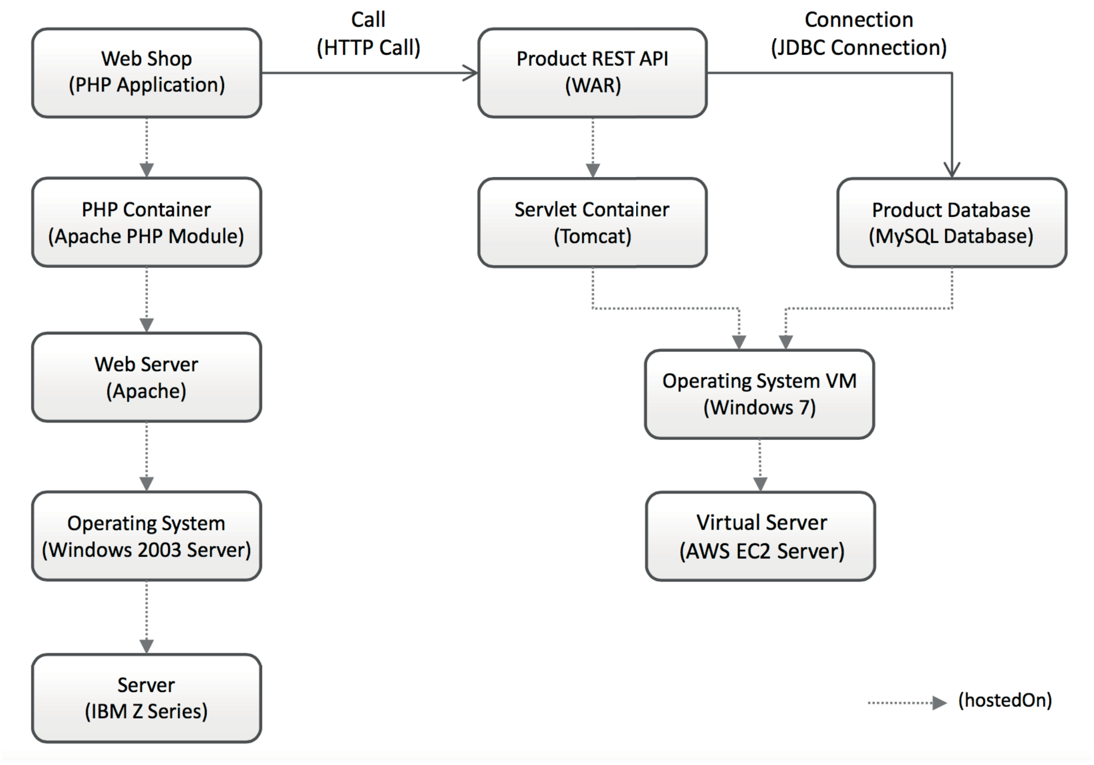


Figure 2.4: Application Topology Example. [Sta13]

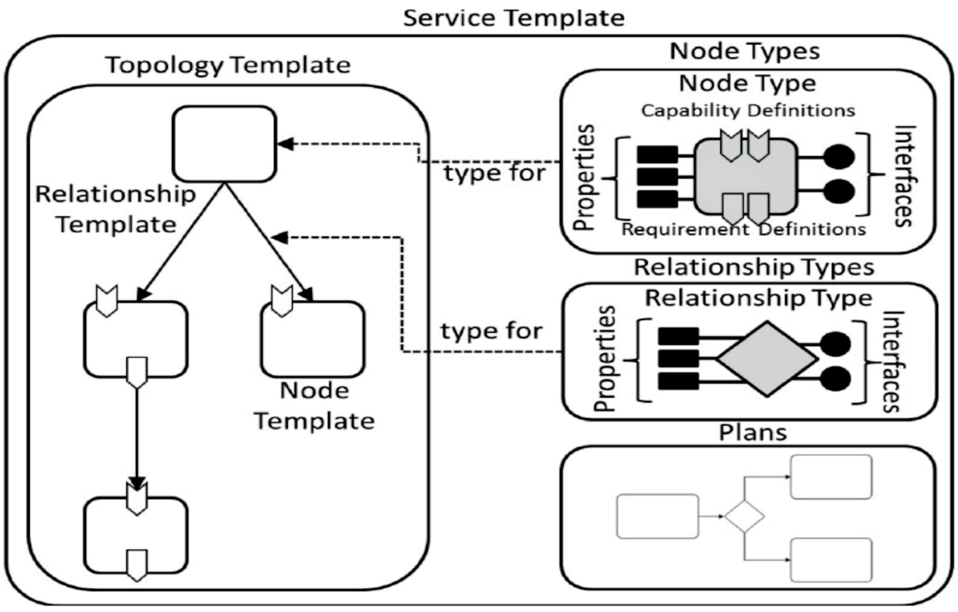


Figure 2.5: TOSCA Service Template. [Sta13]

2.2.2 Topology and Orchestration Specification for Cloud Applications (TOSCA)

TOSCA is an OASIS standardized language for the portable description of service components, their relationships and management processes [BBLS12]. It specifies the structure of multi-tier cloud application components and their relationships with the help of service topology and it also describes the management and deployment procedures to create or modify service instances using orchestration process[Sta13]. The TOSCA specification enables the portability of applications among different cloud environments. As shown in figure 2.5, Service Templates are TOSCA documents which contain node types defining the properties and interfaces of components, node templates representing specific components as a reference to a defined node type, relationship types between node types and relationship templates instantiating the relationship types, topology templates that bring together node and relationship templates, and management plans that define how to deploy, provision, update etc [Sta13].

The properties of non-functional behavior are defined in TOSCA policy type. Policies can be attached to node or relationship templates by means of an external language like WS-Policy. TOSCA also allows for the annotation of node types with requirement and capability definitions, as well as the composition of different service templates.[Sta13]

Cloud application or service is typically packaged in TOSCA using Cloud Service Archive (CSAR) file [Sta13] to run and manage it at different cloud environments. The CSAR archive contains specific hierarchical structure of folder and file as shown in figure 2.7. *TOSCA.meta* file has information about other files and directories of the CSAR zip. *Payroll.tosca* file defines the service template of an application. Node and relationship templates are defined inside the service template, whereas the deployment template is defined within node template. The

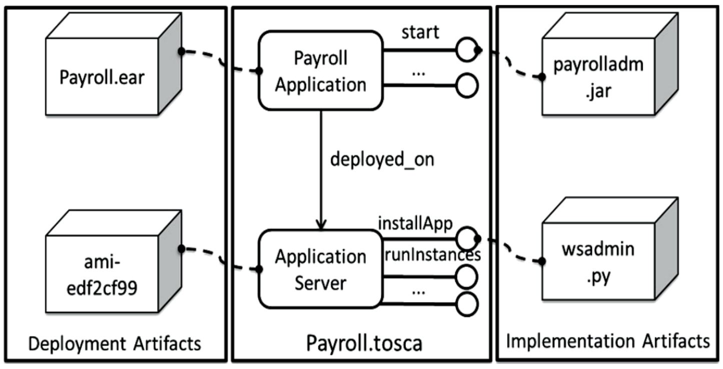
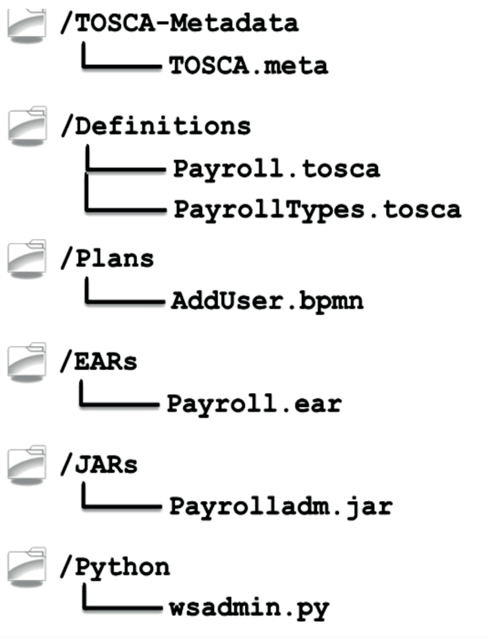


Figure 2.6: TOSCA Service Template Sample. [Sta13]



```
/TOSCA-Metadata
  └──TOSCA.meta
/Definitions
  ├──Payroll.tosca
  └──PayrollTypes.tosca
/Plans
  └──AddUser.bpmn
/EARs
  └──Payroll.ear
/JARs
  └──Payrolladm.jar
/Python
  └──wsadmin.py
```

Figure 2.7: Structure of Cloud Service Archive (CSAR) Example. [Sta13]

PayrollTypes.tosca file defines node and relationship types and is imported by *Payroll.tosca* file. The deployment and implementation executable files are stored in different directories of CSAR. *AddUser.bpmn* file describes imperative management plan. [Sta13]

2.2.3 Generalized Topology Language (GENTL)

The cloud topology languages may use different representations but they have some common fundamental features which can be reused. The application topologies generally contain components or nodes and edges or connectors that connect the components. The components and the connectors form the graphical representation of the topologies. It allows re-usability of existing models, extensibility to accommodate future developments, and composing of topology models of various granularity levels into larger, more complicated ones and facilitates the mapping from and to other languages. It relies on a generic, but typed system and GENTL models are built around a Topology element, which acts as a composer of the other elements in the model. Topology elements may have Topology Attributes that capture information about the topology model as a whole. Components have one or more attributes attached to them and have links to other GENTL Topology elements, which allows for decomposing large topology models and reusing existing ones. A Connector has attributes associated with it and captures a relationship between a source and a target component. Groups allow for the organization of components into sub-graphs of the topology model with non-exclusive memberships, enabling the creation of views on the topological model and have attributes to provide further information about the components they aggregate. [ARSL14]

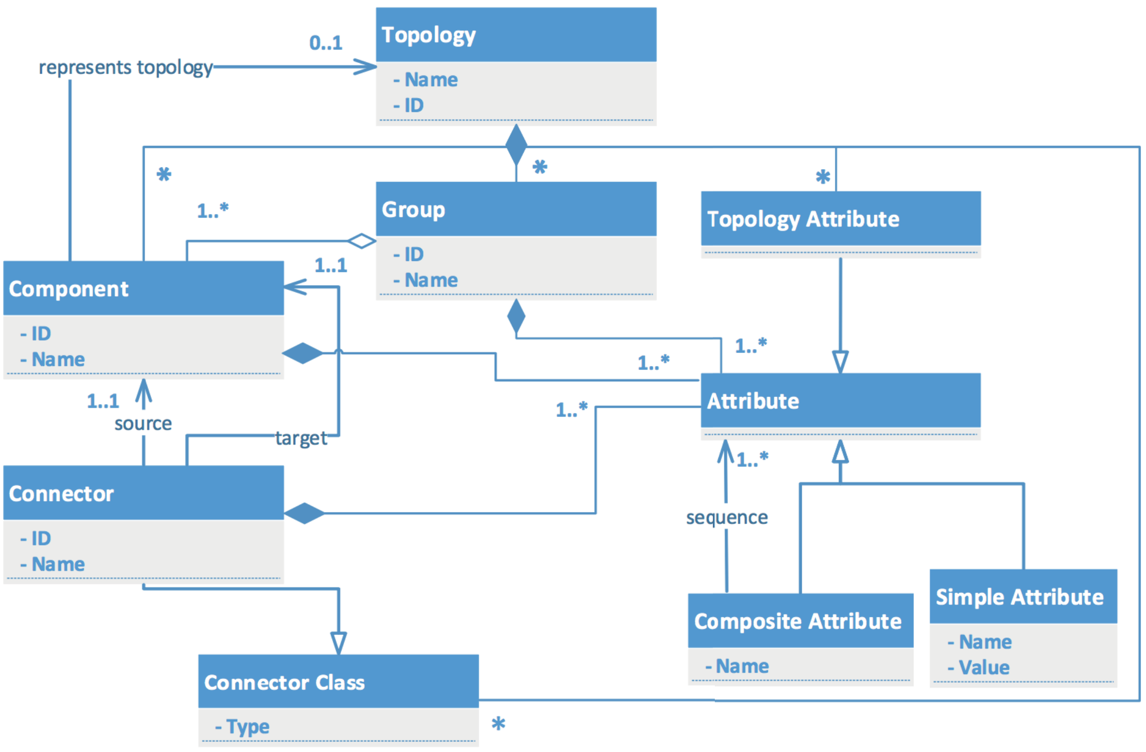


Figure 2.8: GENTL Meta model. [ARSL14]

2.3 DevOps

According to [Htt12], "DevOps is a mix of patterns intended to improve collaboration between development and operations. DevOps addresses shared goals and incentives as well as shared processes and tools. Because of the natural conflicts among different groups, shared goals and incentives may not always be achievable. However, they should at least be aligned with one another." DevOps is an emerging paradigm to eliminate the split and barrier between developers and operations personnel that traditionally exists in many enterprises today. The main promise of DevOps is to enable continuous delivery of software in order to enable fast and frequent releases. This enables quick responses to changing requirements of customers and thus may be a critical competitive advantage. To overcome the split between developers and operations personnel, that is predominant in many organizations today, organizational changes, cultural changes, and technical frameworks are required [WBL14].

DevOps enables the benefits of Agile development to be felt at the organizational level. DevOps does this by allowing for fast and responsive, yet stable, operations that can be kept in sync with the pace of innovation coming out of the development process. The important theme of DevOps is that the entire development-to-operations lifecycle must be viewed as one end-to-end process. Individual methodologies can be followed for individual segments of that processes (such as Agile on one end and Visible Ops on the other), so long as those processes can be plugged together to form a unified process. [dev10]

DevOps approaches are typically combined with Cloud computing to enable on-demand provisioning of resources such as virtual servers and storage in a self-service manner . Different interfaces are offered by Cloud providers such as graphical user interfaces, command line interfaces, and APIs to provision and manage these resources. Especially APIs and command line interfaces are an efficient means to integrate DevOps automation approaches with Cloud resource management programmatically. This can be done for public, private, and hybrid cloud scenarios. Moreover, some cloud providers offer higher-level services such as middleware services (e.g., runtime-as-a- service, database-as-a-service, etc.) and operations automation services, abstracting from the underlying infrastructure. These services can be used alternatively or complementary to the lower-level infrastructure services. [WAL]

2.4 Optimization Algorithms

In this section, we discuss about the various optimization algorithms we considered to use in our application.

2.4.1 Genetic Algorithm GA

According to [Mit96] Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics[Mit96]. They represent an intelligent exploitation of a random search used to solve optimization problems by exploiting historical information to direct the search into the region of better performance within the search space by simulating the survival of the fittest among individuals over consecutive generation for solving a problem[Mit96].

A population of individuals is maintained within search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the binary alphabet 0,1. To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. A fitness score is assigned to each solution representing the abilities of an individual to *compete*. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use selective *breeding* of the solutions to produce *offspring* better than parents by combining information from the chromosomes.[Mit96]

The GA maintains a population of n chromosomes (solutions) with associated fitness values from which parents are selected to mate, on the basis of their fitness, producing offspring. The solutions with higher fitness are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, new generations of solutions are produced containing, on average, more good genes than a typical solution in a previous generation. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation, hence over successive generations better solutions will thrive while the least fit solutions die out. Each successive generation will contain more good *partial solutions* than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.[Mit96]

After an initial population is randomly generated, the algorithm evolves through the following steps[Mit96]:

1. Selection Operator - It works on the principal of the survival of the fittest.
    - Give preference to better individuals to produce fitter offspring.
    - The goodness of each individual depends on its fitness, which is determined by an objective function or by a subjective judgment.
2. Crossover Operator - This is used to mate different individuals
    - Two individuals are chosen from the population using the selection operator
    - A crossover site along the bit strings is randomly chosen and the values of the two strings are exchanged up to this point
    - The two new offspring created from this mating are put into the next generation of the population, which are likely to be better.
3. Mutation Operator - It is used to introduce random modifications
    - A portion of the new individuals will have some of their bits flipped with a low probability.
    - It is done to maintain diversity within the population and inhibit premature convergence.

Effects of Genetic Operators

- Using selection alone will tend to fill the population with copies of the best individual from the population [Mit96].

- Using selection and crossover operators will tend to cause the algorithms to converge on a good but sub-optimal solution [Mit96].
- Using mutation alone induces a random walk through the search space [Mit96].
- Using selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm [Mit96].

Pseudo Code for Genetic Algorithm[KP13]

```
begin GA procedure;
generate populations and fitness function;
evaluate population;
while (termination criteria not meet) do
    while (best solution not meet) do
        crossover mutation evaluate;
    end
end
post-process results and output;
end GA procedure;
```

Algorithm 1: Genetic algorithm

2.4.2 Multi-Agent Genetic Algorithm (MAGA)

According to [ZSL+11], Multi-Agent Genetic Algorithm is a improved hybrid algorithm combining genetic algorithm and multi-agent techniques and demonstrates greatly enhanced convergence time and optimization results compared to that of traditional GA specially when handling very large-scale, high- dimensional, complex, and dynamic optimization problems. It treats an individual within GA as an agent which is capable of local perception, competition, cooperation, self-learning, and reaching the purpose of global optimization through the interaction between both agent and environment, and agent and agent. The genetic operators used here are neighborhood competition operator which realizes the operation of competition among all agents; the neighborhood orthogonal crossover operator achieves collaboration among agents; the mutation and self-learning operators which accomplish the behavior that agents exploit their own knowledge. [ZSL+11]

To establish a load balancing model, the main parameters required from a single user include: User (ReqPerHrPerUser, ReqSize, ReqCPU, ReqMemory, Count). To solve the issue of exploding dimension, group strategy is exploited to set up a resource scheduling model. It is based on the user requested parameters, and the parameters inside the group all have the maximum value. The establishment of the load balancing model mainly refers to the design of fitness function. On the basis of group strategy, all of the virtual resources on a physical resource's host correspond to the user group strategy request. VM is able to allocate several groups like: Group (ReqPerHrPerUser, ReqSize, ReqCPU, ReqMemory, Count). Binary encoding is used here.[ZSL+11]

2.4.3 Simulated Annealing (SA)

According to [Bro11], Simulated Annealing is inspired by the process of annealing in metallurgy in which a material is heated and slowly cooled into solid crystal state with minimum energy and larger crystal size to reduce defects in metallic structures. The heat increases the energy of the atoms allowing them to move freely, and the slow cooling schedule allows a new low-energy level to be discovered.

Each set of a solution represents a different internal energy of the system and heating the system results in a relaxation of the acceptance criteria of the samples taken from the search space. As the system is cooled, the acceptance criteria of samples are narrowed to focus on improving movements. Once the system has cooled, the configuration will represent a sample at or close to a global optimum. The SA algorithm allows the search to sometimes accept worst solutions with a probability that would decrease with the temperature of the system. Initial temperature and cooling rate should be set such that the slower the temperature is decreased, the greater the chance an optimal solution is found. [KP13]

Pseudo code for Simulated Annealing [KP13]

```
begin SA procedure;
generate populations initial solutions;
set temperature and cooling rate;
while (termination criteria not meet) do
    generate new solutions;
    access new solutions;
    if (accept new solution) then
        update storage;
        adjust temperature;
    end
end
post-process results and output;
end SA procedure;
```

Algorithm 2: Simulated Annealing

2.4.4 Ant Colony Optimization

As per [KP13], ACO is inspired by a mechanism called *stigmergy* or indirect communication and coordination which is used by ants to find optimum path between colony and food source. When food is located, real ants initially roams randomly from their colony to food depositing pheromones on their paths while going as well as returning. The ants following shortest path return earlier and amount of pheromone on that path is more and after sometime it has more traffic. The pheromone evaporates at a certain rate so the longer paths are eliminated after certain time. The ants use the history in terms of pheromone trail to search shortest path from colonies to their food.[KP13]

ACO algorithm is employed to imitate the behavior of collective foraging of real ants [GGQ$^+$13]. It has been successfully applied to solve numerous optimization problems and is also now used to solve continuous optimization problems. According to [GGQ$^+$13], there are three basic points on which ACO algorithm and its variations are based:

1. Pheromone update - When updating pheromone trails, one has to decide on which of the constructed solutions to lay pheromones. There are usually two strategies to update the pheromone trails. A first strategy is to select the iteration-best or best-so-far solutions to update the pheromone matrices, with respect to each objective. A second strategy is to collect and store the non-dominated solutions in an external set which are allowed to update the pheromones. [GGQ$^+$13]

2. Definition of pheromone and heuristic information - At each step of the construction of a solution, a candidate is chosen relative to a transition probability which depends on a pheromone factor and a heuristic factor. The pheromone/heuristic information can be defined using one or multiple matrices. When only one matrix is utilized, the pheromone information associated with each objective is combined to reduce the multiple objectives into a single one. If multiple matrices are used, usually each matrix corresponds to one objective. With respect to the pheromone information, each matrix may contain different values depending on the implementation strategy applied. The same applies to the heuristic information. [GGQ$^+$13]

3. Pheromone and heuristic aggregation - Whenever multiple matrices are used, one must use some form of aggregation procedure to aggregate the pheromone/heuristic matrices. There are three common strategies for this: the weighted sum, where matrices are aggregated by a weighted sum; the weighted product, where matrices are aggregated by a weighted product; and random, where at each step a random objective is selected to be optimized. Weights used for aggregating multiple matrices, can be set dynamically, where each ant may be assigned a different weight from the other ants at each iteration or fixed, where we can assign to all ants the same weight and each objective has the same importance during the entire algorithm run. [GGQ$^+$13]

Pseudo Code for Ant Colony Optimization [KP13]

```
begin procedure ACO;
generate pheromone trails and other parameters;
while (termination criteria not meet) do
    construct solutions;
    update pheromone trails;
end
post-process results and output;
end procedure ACO;
```

Algorithm 3: Ant Colony Optimization

2.5 Migration of Applications to Cloud

2.5.1 Optimal Distribution of Applications in Cloud

According to [ASLW14], there are many ways which can be used for the distributed deployment of the application across cloud providers like TOSCA, CloudML or Cloud Blueprints which allow for a portable and inter operable topological description of the application stack. But these approaches lack decision support capabilities towards optimally selecting the best of the identified application topologies in a given situation. The main aim was to provide a technology-agnostic formal framework that provides the means to:

– Model, verify and automatically generate alternative scenarios for the distribution of an application stack across cloud offerings. Applications in this context may entail a complete information system, or only part of it [ASLW14].

– Evaluate each one of these distribution scenarios with respect to various dimensions using different criteria, and allow the selection of an optimal scenario given the application needs [ASLW14].

An application topology is a labeled graph G = (NL,EL,s,t) where N is a set of nodes, E is a set of edges, L a set of labels, and s,t the source and target functions. A type graph with inheritance TGI is a triple (TG,I,A) consisting of a type graph TG=(N,E,s,t),an inheritance graph I sharing the same set of nodes N, and a set NA c N, called abstract nodes.A typed topology T is viable w.r.t. Type Graph with Inheritance (TGI), iff all elements of T are labeled (typed) over the elements of T GI , i.e. there exists a graph morphism m : TGI -> T which uses the inheritance clan relation. There are two ways to look at the morphism m that translates TGI to T: top-down and bottom-up [ASLW14].

The type graph with inheritance for a viable application topology is called μ-topology and has two parts: the application-specific α-topology and the non application-specific, reusable γ-topology [ASLW14]. A sample μ topology for the Web shop application is shown in the figure 2.9.

2.5.2 Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud

According to [FFH13], CDOSim can evaluate CDOs, e.g., regarding response times and costs. CDOSim simulates the response times, SLA violations, and costs of a Cloud Deployment Option (CDO) by exploring the CDO search-space on the basis of automatically extracted architectural models and approximates the corresponding pareto optimum.

CDOXplorer optimizes the allocation of software components to VMs, but also searches for reconfiguration rules that are aligned with the cloud's elasticity and the specific performance and pricing models of the available cloud environments. It uses techniques of the search-based software engineering field and simulation runs of CDOSim to restrict the search-space and to steer the exploration towards promising CDOs.Four input models(architectural model,

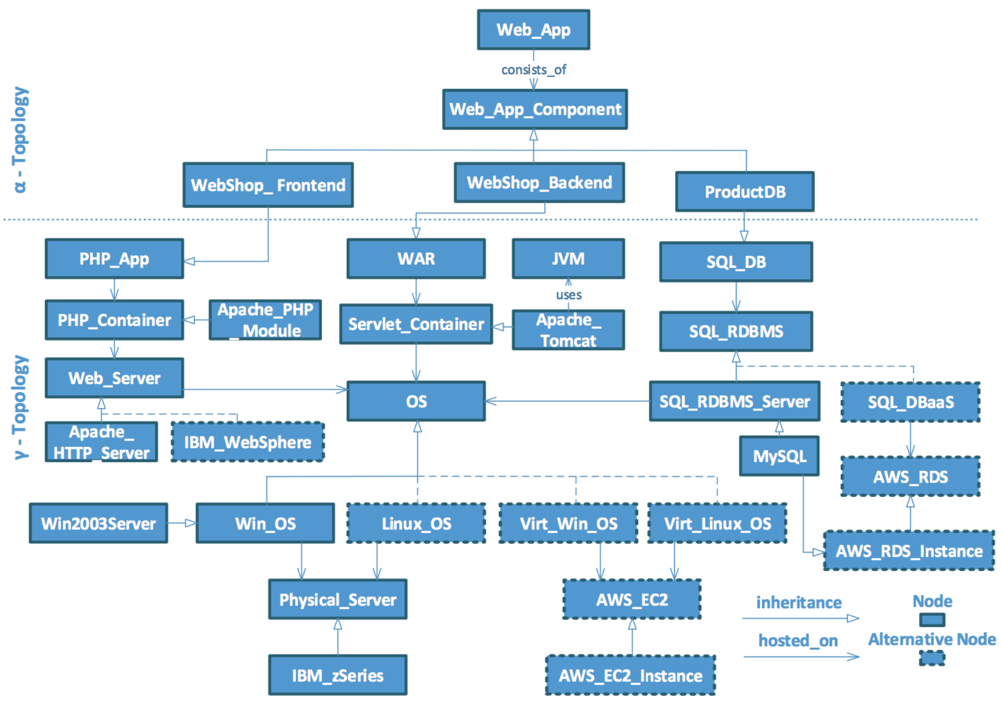


Figure 2.9: Extended μ-Topology of the Webshop Application. [ASLW14]

status-quo deployment model, workload profile, and cloud profile) have to be provided to CDOXplorer so it can find well-suited cloud deployment models and reconfiguration rules.[FFH13]

There exists no single global optimum, so genetic algorithms aim to iteratively approximate the pareto optimal set (pareto optimum) which is a subset of all individuals that includes all pareto optimal individuals, i.e., individuals for which the improvement of one objective would lead to a deterioration of another objective. The reproduction of each generation includes the following four basic steps: Select parents, Recombine parents (crossover), Mutate offspring and Evaluate offspring's fitness. Appropriate pairs of diverse parents are selected based on NSGA-II algorithm. [FFH13]

Multi-objective optimization problem that is tackled by CDOXplorer can be described as follows. The goal is to find a CDO from the set of all feasible options that complies with the structure of CDOs, the value ranges defined, and the constraints that are described. The crossover operator was designed to produce only feasible individuals; hence the mixing was restricted to dedicated positions. As a mutation also has to maintain the inner structure of a chromosome, it is divided in five sub operators for cloud environments, node configurations, initial start configuration, service composition and a reconfiguration rule. [FFH13]

2.6 Case Based Reasoning

It is the process of solving new problems based on the solutions of similar problems encountered in the past. It involves four steps: Retrieve, Reuse, Revise and Retain. Retrieve refers to looking up cases from past relevant to problem at hand which tells us about the past problem, its solutions and how the solution was derived. In the Reuse step, we try to map the older problem to the problem at hand and try to modify the solution as per our needs. Next, we Revise the modified solution in real world scenarios and after it has been completely adapted to the problem at hand, we need to Retain it as a new case. [Wik15]

2.7 Feature Models and Feature Diagram

Software Product Line (SPL) engineering is a paradigm to develop software applications using platforms and mass customization.A feature model is a hierarchically arranged set of features [SHT06]. As per [Bat05], relationships between a parent feature and its child features are categorized as:

- And — all subfeatures must be selected,
- Alternative — only one subfeature can be selected,
- Or — one or more can be selected,
- Mandatory — features that required, and
- Optional — features that are optional.

The graphical representation of the various notations can be seen in the Figure 2.10

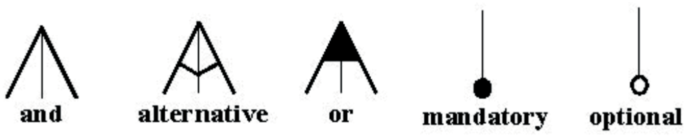


Figure 2.10: Feature Diagram Notations. [Bat05]

According to [Bat05],Feature models are used to specify members of a software product line and a member is defined by a unique combination of features. A feature diagram is a graphical representation of a feature model organized as a tree. It is a tree where primitive features are leaves and compound features are interior nodes. A tree grammar requires every token to appear in exactly one pattern, and the name of every production to appear in exactly one pattern. The root production is an exception; it is not referenced in any pattern. More general grammars can be used, but iterative trees capture the minimum properties needed for our discussions.[Bat05]

2.8 RESTful Services

The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system [Fie00]. The details of component implementation and protocol syntax are ignored in REST in order to focus on the roles of components, the constraints upon their interaction with other components, and the interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors and data that define the basis of the web architecture and thus the essence of its behavior as a network-based application. The RESTful architectural constraints as mentioned in [Fie00] are :

- Client-server - It is the most basic constraint and enforces separation of concerns which helps establish a decoupled architecture supporting independent evolution of logic on the client and the server. As per [Fie00], *A client is a triggering process and a server is a reactive process. Clients make requests that trigger reactions from servers thus, a client initiates activity at times of its choosing; it often then delays until its request has been serviced. On the other hand, a server waits for requests to be made and then reacts to them. A server is usually a non-terminating process and often provides service to more than one client.*

- Stateless - The communications in a RESTful system must be stateless which means no session state is saved in the server. It means each request from a client should include all the necessary contextual information which the server requires to understand the request. This in return relaxes the web server by freeing it from memorizing the state of its client application. The benefit lies in the possible scalability of the web's architectural style.[Fie00]

- Cache - Caches help the server, client, or one of the intermediary middleware components to store the response for reuse in later requests which enables increased overall availability of responses, reliability of an application thus controlling a web server's load and reducing the cost. The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions [Fie00]. Caching also counter balances some of the negatives incorporated in REST design due to stateless constraint.

- Uniform Interface - To ensure effective and robust communication system, the uniformity of the interfaces for interaction between the web components such as servers, clients and network-based intermediaries is of major concern. [Fie00] identified four constraints in order to achieve the uniformity which are:

  1. Identification of resources - According to [Fie00], *The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object, and so on.* The resources can be identified by defining a global addressing space for resource and service discovery. E.g., for a particular home page has to be unique in order to be specific to that website's root resource.

2. Manipulation of resources through representations - We should be able to manipulate the representations so that the same exact resource can be represented to different clients in different ways. This provides the freedom for the varied representation of the same resource. [Fie00]

3. Self-descriptive messages - We need to ensure that the messages are complete in itself thus including the meta-data to convey details regarding the resource state, the representation format and size to enable multiple formats for the resource representation as mentioned above. [Fie00]

4. Hypermedia as the engine of application state (HATEOAS) - A resource's state representation includes links to related resources and so the resource's current state has to be within the message and not on the server end. This leads to interactions which are stateful as the information is contained in hyperlinks but is not stored at the server side but at the client side. A client interacts with a network application entirely through hypermedia provided dynamically by application servers. [Fie00]

- Layered System - *A layered system is organized hierarchically, each layer providing services to the layer above it and using services of the layer below it but no one layer can see past the next* [Fie00]. Hence it helps reduce coupling across multiple layers by hiding the inner layers from all except the adjacent outer layer, thus improving re-usability and the ability to evolve. This helps to transparently deploy components like proxies and gateways which are required for enforcing security, load balancing or response caching. By restricting knowledge of the system to a single layer, we place a bound on the overall system complexity and promote substrate independence.

- Code-On-Demand - This constraint tends to establish a technology coupling between web servers and their clients, since the client must be able to understand and execute the code that it downloads on-demand from the server. This optional constraint is primarily intended to allow logic within clients (such as Web browsers) to be updated independently from server-side logic [Inc].

The architectural properties offered by REST that help establish the design goals that lie behind the application of REST constraints are:

- High performance - REST can support the Performance goal by using caches to keep available data close to where it is being processed and can further help by minimizing overhead associated with setting up complex interactions by keeping each interaction simple and self-contained (as a request-response pair) [Inc]. The most common kind of resource is a file, but a resource may also be for example a dynamically-generated query result or a document.

- Scalability - This property supports the growing demand in the form of need to support a large number of instances or concurrent interactions. Being stateless, all the required state information for the interaction are contained within the request itself. The request hence has no server affinity which enables to spray the request across cluster of servers hence providing a scaling system. As per [Inc], four basic approaches for dealing with scalability demands are identified and can be combined in various ways:

  - scaling up - increasing the capacity of services, consumers, and network devices
  - scaling out - distributing load across services and programs
  - smoothing out - evening out the number of interactions over peak and non-peak periods to optimize the infrastructure (thereby reducing the impact of the peaks to avoid the infrastructure sitting idle at other times)
  - decoupling the consumption of finite resources - such as memory from concurrent consumers

- Simplicity - For the application to be simple and understandable, the proper application of separation of concerns is required. Four generic verbs Create, Read, Update and Delete (CRUD) allows for the ease of implementation.
- Portability - The ease at which services and solutions can be moved from one deployed location to another is represented by the goal of portability [Inc].The application must be capable to be run on heterogeneous environments as the service oriented architecture supports the provision to use services being technology and platform independent.
- Data Independency - REST promises to allow the possibility to use any possible format for a resource. Different formats may be used to represent the data of a single resource therefore the request may contain client's formatting capabilities allowing for content negotiation.

2.8.1 RESTful API Design

An API or Application Programming Interface lets the client program communicate with the service. In order to use the web services there are APIs for the client program and an API exposes a set of data and functions to facilitate interactions between computer programs and allow them to exchange information. A REST API is a web API that confirms with REST architectural style. To design a REST API, there are certain practices implicit to the HTTP standard but due to the flexibility of designing the API it becomes easier to develop comprehensible APIs according to what a service has to offer. The designing of the API is of significance as for a client to be able to use a service, the functionalities it offers and how these functionalities can be used is visible via the API alone. The resources are the most fundamental units to a REST API. Hence, to access the resource a URI must be designed to be able to reach out to a resource. The other aspect to it is to perform any action on the resource which can be done via standard methods offered by HTTP. [Mas11]

API Design Rules

A set of rules have been defined for designing REST APIs in order to maintain the consistency and to leverage a standard and clean API for the client usage. These rules let the client use the API with clear understanding of what service is offering and saves the designer from confusions and provokes them for a careful consideration while designing the API. Many of the rules have become standard for the design while some rules can be accepted or slightly modified to make the URIs more readable and easier to understand. The design rules for URI format are [Mas11]:

- Forward slash separator (/) must be used to indicate a hierarchical relationship.
- A trailing forward slash (/) should not be included in naming the resources.
- Lowercase letters should be preferred in URI paths. File extensions should not be included in URIs rather REST API clients should be encouraged to utilize HTTP's provided format selection mechanism. The REST API clients should allow the user to explicitly mention the format they expect for the response.
- URI path conveys the REST API's resource model, hence each forward slash separated path segment must correspond to a unique resource within the model's hierarchy.
- While considering different resource archetypes, a singular noun should be used for document names 15, a plural noun should be used for collection names 16 and a plural noun should be used for store names 17.
- CRUD function names should not be used in URIs.
- The query component of a URI may be used to filter collections or stores. This would help to distinguish between the resources.
- GET and POST must not be used to tunnel other request methods where tunneling refers to incorrectly using the HTTP methods to limit the client with less HTTP vocabulary.
- GET must be used to retrieve a representation of a resource, PUT must be used to both insert and update a stored resource with a a request message having a body to represent the desired changes. POST must be used to create a new resource in a collection and DELETE must be used to remove a resource from its parent.
- The HTTP Location response header must designate the URI of the newly created resource.
- Custom HTTP headers must not be used to change the behaviour of HTTP methods.

These rules are of importance for designing any REST API therefore have been considered for this thesis work. The use of these rules has been discussed in a elaborate manner in Chapter 5.

2.9 Nefolog Cost Calculator

Nefolog contains the collection of decision support services and the knowledge base and its basic architecture can be seen in the figure 2.11. The services have 2 representations- JavaScript Object Notation (JSON) and EXtensible Markup Language (XML) and are defined by different URIs which can also present the requirements of users. They are also used to handle the interactions between users and the Cloud provider knowledge base. Knowledge

Base is a relational database and data are organized by tables which are linked by foreign keys. Web services are implemented in the Restlet framework and operated in conjunction with the knowledge base. There are two main decision support services offered: candidate search and cost calculator. The candidate search service is accomplished by the comparisons between user demands which are presented by the query part of URIs and data in knowledge base. The cost calculator service is used to calculate the costs of candidate offerings with the help of cost formulas. analyze can query on the system to get service types, providers, etc.[XA+13]

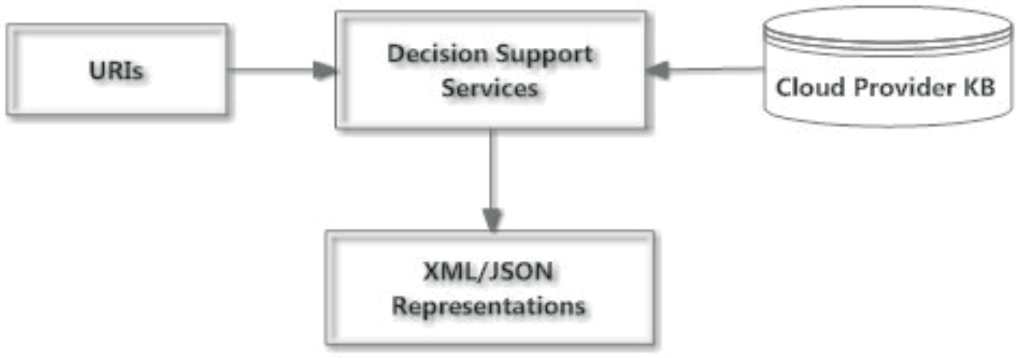


Figure 2.11: Architecture of Nefolog. [XA+13]

2.10 OpenTOSCA

OpenTOSCA is a runtime supporting imperative processing of TOSCA applications. Imperative means that the deployment and management logic is provided by plans. The key tasks of OpenTOSCA, addressed by the architecture depicted in Fig. 2, are to operate management operations, run plans, and manage state. Requests to the Container API are passed to the Control component, which orchestrates the different components, tracks their progress, and interprets the TOSCA application. The Core component offers common services to other components, e. g., managing data or validating XML.[BBH+13]

According to [BBH+13], Management operations of nodes and relationships are either provided by running (Web) services or by Implementation Artifacts contained in the CSAR. In the latter case, the Implementation Artifact Engine is responsible to run these artifacts in order to make them available for plans. Implementation Artifacts, e. g., a SOAP Web service implemented as Java Web archive (WAR), are processed by a corresponding plugin of the engine which knows where and how to run this kind of artifact. The plugins deploy the respective artifacts and return the endpoints of the deployed management operations to be stored in the Endpoints database. [BBH+13]

The management plans contained in CSARs are processed by the Plan Engine, which also employs plugins to support different workflow languages, e.g., Business Process Model and Notation (BPMN) or Business Process Execution Language (BPEL), and their runtime environments. Plans only define abstractly which kind of service they require but not their

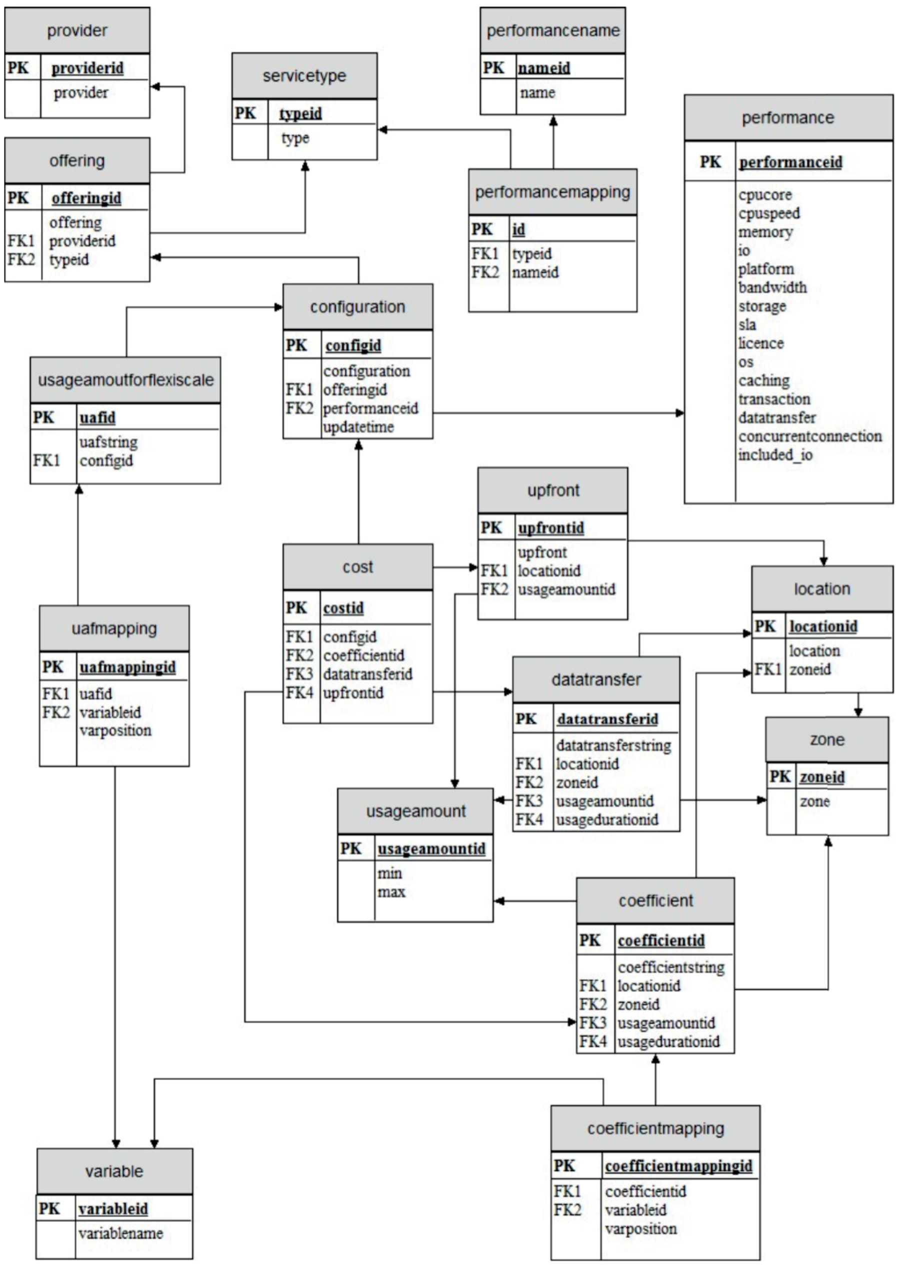


Figure 2.12: Data Model of Knowledge Base.[XA+13]

concrete endpoints. Therefore, the corresponding plan plugin binds each service invoked by the plan to the endpoint of the management operation before it deploys the plan to the respective work flow runtime. The service's endpoint was added to the endpoint database before by the Implementation Artifact Engine. This way of binding workflows ensures portability of management plans between different environments and runtimes. By using the Plan Portability API, management plans can access the topology and instance information, e.g., the property values of nodes and relationships.[BBH+13]

The OpenTOSCA architecture can be seen in the Figure 2.13. The plugin architecture of the Implementation Artifact Engine and Plan Engine ensure extensibility. Portability is ensured by the two engines working together when binding management plans. Strict separation of architectural components through well-defined OSGi interfaces enables the replacement of implementations of components. This also allows each component to be scaled independently. [BBH^{+}13]

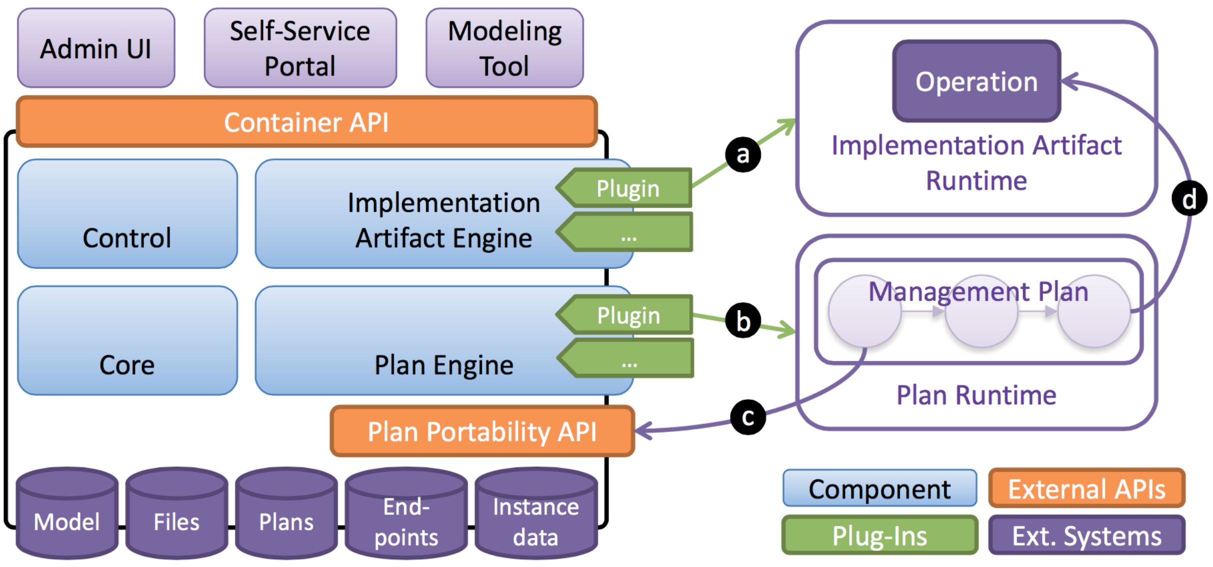


Figure 2.13: OpenTOSCA Architecture Overview and Processing Sequence.[BBH^{+}13]

3 Related Works

There are several existing approaches for analyzing and processing the evolution of Cloud application topologies in an efficient manner. These approaches, such as Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud and TOSCA Mart, are described in the following subsections.

3.1 Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud

This work deals with deploying applications in the cloud and gives the user various cloud deployment options.CDOSim can evaluate CDOs, e.g., regarding response times and costs. CDOSim simulates the response times, SLA violations, and costs of a CDO. It explores the CDO search-space on the basis of automatically extracted architectural models and approximates the corresponding pareto optimum. CDOXplorer optimizes the allocation of software components to VMs, but also searches for reconfiguration rules that are aligned with the cloud's elasticity and the specific performance and pricing models of the available cloud environments. It uses techniques of the search-based software engineering field and simulation runs of CDOSim to restrict the search-space and to steer the exploration towards promising CDOs.Four input models (architectural model, status-quo deployment model, workload profile, and cloud profile) have to be provided to CDOXplorer so it can find well-suited cloud deployment models and reconfiguration rules.[FFH13]

This work provides well suited deployment models for applications on cloud. We will be using similar idea as the one here to provide application topologies with parts of application deployed on physical servers. In this work, they build the topologies and then compare among them to find out the pareto optimal set. But we would be discovering the optimal set of services for each tier of the application and then build the optimal deployment alternatives for the whole system. Also we divide the application topology into an application specific part and a non application specific, reusable part.

3.2 TOSCA-based Method for Adapting and Reusing application Topologies (TOSCA-MART)

TOSCA-MART allows to concretely implement application components with certain requirements by adapting and reusing fragments of existing application topologies. It allows developers to define custom TOSCA application components by declaring the offerings and requirements they need to be properly operated. These features are then matched against those provided by each of the topologies available in a repository of existing cloud applications, so as to determine the topology fragments and automatically select the best. It can be used to discover complete topologies as well as middleware and infrastructure fragments to host new applications so instead of modeling complete topologies, application developers can define only the offerings and requirements and TOSCA-MART will then automatically implement them, thus significantly decreasing the effort and cost needed for developing cloud applications. It does not restrict to applications developed with a specific methodology, nor it requires the availability of application's source code, and it is hence applicable also to non open-source, third-party services. [SBB+15]

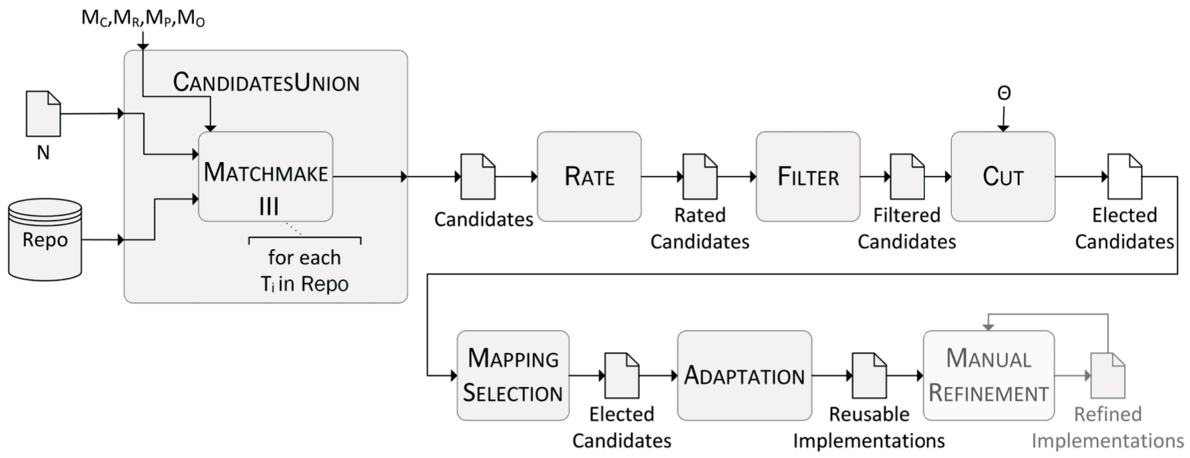


Figure 3.1: The TOSCA-MART matchmaking and adaptation method. [SBB+15]

TOSCA-MART method is illustrated in the figure 3.1. The goal is to derive an implementation for a target Node Type N by excerpting it from a repository Repo of cloud applications. Once N and Repo are available, each application topology in Repo is compared with N by employing the Matchmake procedure and obtain the set Candidates. Due to the potentially huge number of already avail- able topologies and to the possibility of having multiple candidates for each of these topologies, the set Candidates may become huge. The number of available candidates is reduced by applying 3 subsequent steps. First, Rate computes a score for each candidate and the set Candidates is transformed in the set RatedCandidates. After this, Filter reduces the number of RatedCandidates by removing duplicates and gives the set FiletredCandidates. Finally, Cut reduces the number of candidates according to a threshold and the set FilteredCandidates is reduced to the set ElectedCandidates, which contains only the best candidates. Each of the ElectedCandidates has to be adapted to properly implement the target. MappingSelection step helps to select the most proper mapping among the available ones in order to avoid the user to select mappings. Once the mappings are selected, each of the ElectedCandidates is adapted by resolving the unsatisfied dependencies of the selected components, and by enclosing the candidate fragments into standalone application specifications which implement the target NodeType N. All these specifications compose the set ReusableImplementations, which is the output of the TOSCA-MART method. Finally, an optional ManualRefinement step may be done to allow the cloud application developer to manually modify the outputted NodeType implementations, if they are not designed as desired. [SBB$^+$15]

As per [SBB$^+$15], TOSCA-MART goes a step further, by allowing to reuse not only entire application topologies, but also fragments of such topologies. It focuses mainly on topology completion and reusability i.e. it can be used to discover complete topologies as well as middleware and infrastructure fragments to host new applications. TOSCA-MART automatically implements the topologies as per the offerings and requirements defined by the developer but it does not focus on providing an optimal set of topologies. Also it does not take into account the option of deploying parts of applications on physical servers when building the topologies.

3.3 CloudGenius

CloudGenius is a framework, which helps in migration of web applications to the cloud. Migrating Web applications to cloud services and integrating cloud services into existing computing infrastructures is non-trivial, the problem being selecting the best and compatible mix of software images (e.g., Web server image) and infrastructure services to ensure that Quality of Service (QoS). CloudGenius automates the decision-making process based on a model and factors specifically for Web server migration to the Cloud. It leverages a well known multi-criteria decision making technique, called Analytic Hierarchy Process, to automate the selection process based on a model, factors, and QoS parameters related to an application. [MR12]

Cloud infrastructure service selection, cloud VM image selection, cloud VM image customization, migration strategy definition, and migration strategy application are the 5 main steps that outline a migration of an organization’s Web application to an equivalent on a cloud infrastructure service. CloudGenius provides a decision support system that is capable of enhancing the quality of cloud infrastructure service selections and cloud VM image selections. This approach that translates both selection steps into multi-criteria decision-making problems to determine the most valuable combination of a cloud VM image and a cloud infrastructure service. defines a Cloud migration process. CloudGenius offers a model and methods to determine the best combined choice of a Cloud VM image and a Cloud infrastructure service. The framework leverages an evaluation and decision-making framework, called (MC2)2 to support requirements and adopt a profound multi-criteria evaluation approach. [MR12]

CloudGenius allows users to define an abstract Web server and set requirements for a Web server implementation that condition what attributes are acceptable for a Cloud equivalent from which, a user has to choose relevant factors from a criteria list and define their priorities by setting weights for criteria in pair-wise comparisons [MR12]. From the information given by the user the CloudGenius approach employs a model and the user preferences to apply the (MC2)2 framework and suggest a best cloud image and cloud service combination. But, it does not provide the user with a set of pareto optimal deployment alternatives to choose from. Also, it selects a cloud image and service configuration for the whole application rather than providing a chance to deploy various parts of applications in various cloud offerings.

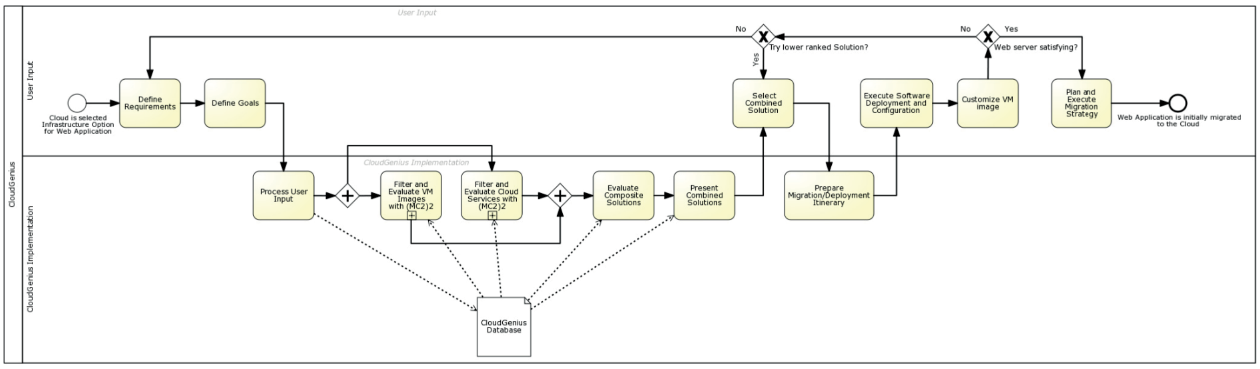

CloudGenius
User Input
CloudGenius Implementation
User Input
Cloud is selected Infrastructure Option for Web Application
Define Requirements
Define Goals
No
Try lower ranked Solution?
Yes
Select Combined Solution
Execute Software Deployment and Configuration
Customize VM image
No
Yes
Web server satisfying?
Plan and Execute Migration Strategy
Web Application is initially migrated to the Cloud
CloudGenius Implementation
Process User Input
Filter and Evaluate VM Images with (MC2)2
Filter and Evaluate Cloud Services with (MC2)2
Evaluate Composite Solutions
Present Combined Solutions
Prepare Migration/Deployment Itinerary
CloudGenius Database

Figure 3.2: Migration Process of the CloudGenius Framework. [MR12]

3.4 Context-Aware Cloud Topology Optimization and Simulation (CACTOS)

CACTOS approach to cloud infrastructure automation and optimization addresses heterogeneity through a combination of in-depth analysis of application behavior with insights from commercial cloud providers. The aim of the approach is threefold: to model applications and data center resources, to simulate applications and resources for planning and operation, and to optimize application deployment and resource use in an autonomic manner. The vision of CACTOS is to produce new data center optimization and simulation mechanisms that can handle the scale, heterogeneity, and complexity of modern cloud application workloads while providing advanced infrastructure capabilities such as resource elasticity and controllable application quality of service (QoS). The long-term goal of this work is to develop integrated monitoring, simulation, and management tools that accurately capture the dynamics of complex workloads, abstract the heterogeneity of resource sets, and optimize virtual machine and resource configurations to increase the resource and energy efficiency of cloud data centers. It emphasizes the three core concepts: Context-awareness, Topology optimization and Simulation.[OGW$^+$14]

There are 2 major challenges which exist for realizing the CACTOS vision :

- Cloud System Scale and Complexity : The scale of cloud applications ranges from basic services running in individual virtual machines to complex and distributed applications spanning multiple services hosted in multiple geographically distributed data centers [OGW$^+$14].

- Cloud Workload and Infrastructure Heterogeneity : Heterogeneity permeates both cloud workloads and infrastructures at multiple levels. Modeling and prediction of workloads requires understanding of application behavior and heterogeneous workload characteristics. Resource requirements may vary and depends heavily on the resource usage profile. Modeling and characterizing changes in the behavior of heterogeneous workloads on heterogeneous hardware poses major challenges. [OGW$^+$14]

The key challenges in cloud infrastructure topology optimization include the identification of key performance indicators and management actions that can be monitored and used to control data center resources. In CACTOS, data centers are modeled in a sensor-actuator model where sensor (monitoring) data are captured in infrastructure topology and load models and actuator actions are represented in optimization plans that list recommended changes to the infrastructure using instructions from a predefined optimization plan language. Using this model, data center operations are then described in a closed Observe-Plan-Act loop, where the state of the data center resources and applications are continuously monitored, and plans (changes to resource configurations and application mappings) are made and enacted to optimize data center operations towards selected objective functions. [OGW$^+$14]

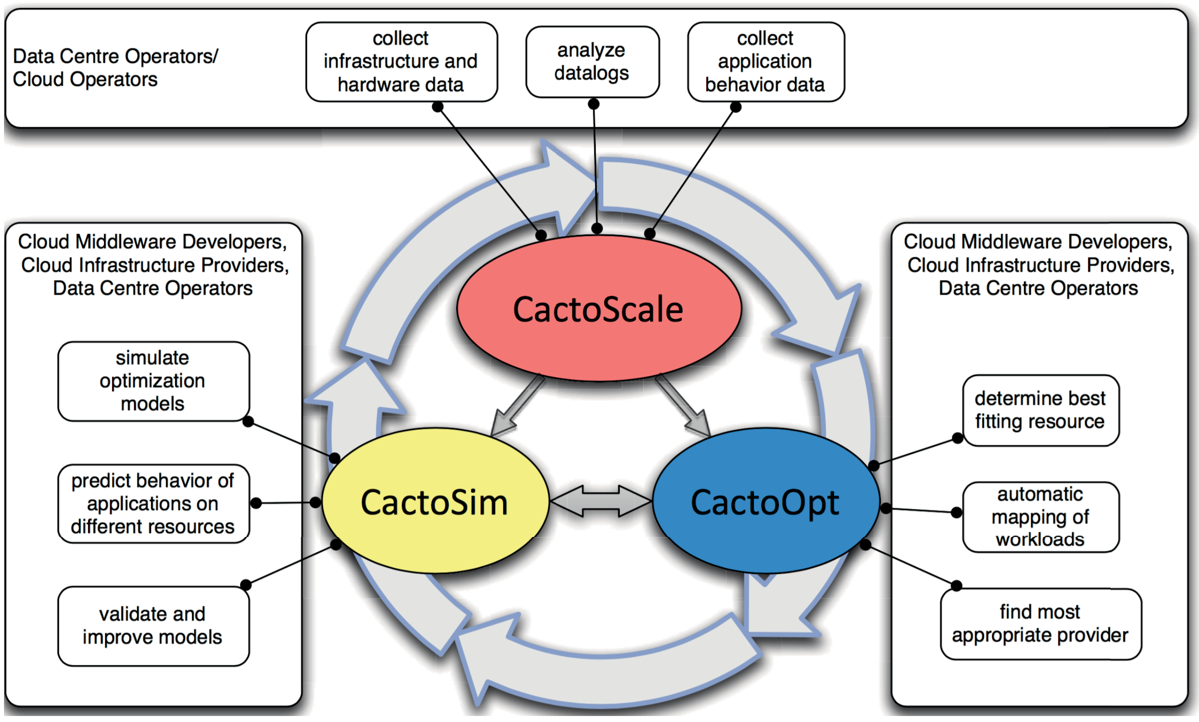

Data Centre Operators/ Cloud Operators
collect infrastructure and hardware data
analyze datalogs
collect application behavior data
Cloud Middleware Developers, Cloud Infrastructure Providers, Data Centre Operators
simulate optimization models
predict behavior of applications on different resources
validate and improve models
CactoScale
CactoSim
CactoOpt
Cloud Middleware Developers, Cloud Infrastructure Providers, Data Centre Operators
determine best fitting resource
automatic mapping of workloads
find most appropriate provider

Figure 3.3: The continuous cycle of the CACTOS Observe-Plan-Act loop. [OGW$^+$14]

The main focus of CACTOS is to develop tools that accurately capture the dynamics of complex workloads, abstract the heterogeneity of resource sets, and optimize virtual machine and resource configurations to increase the resource and energy efficiency of cloud data centers. It also helps to model applications and optimize application deployment but it is more focused on middleware and infrastructure related problems rather than topologies for deploying complete applications. It aims to optimize the mapping of services to resources rather than providing set of optimal topologies for deploying applications as per the requirements of the developer.

3.4.1 MOve to Clouds for Composite Applications (MOCCA)

[LFM$^+$11] says, MOCCA aims to split an application and decide which component of the application should be put in which cloud basing on the functional and non-functional properties. The rearrangement of the application's deployment topology is a major challenge in moving applications to cloud. It solves the Move-to-Cloud problem which is how to rearrange the components of a multi-tier, multi-component application into disjoint groups of components. Each such group can be provisioned separately to different clouds while preserving the desired properties of the whole application. It transforms the Move-to-Cloud problem into a graph-partitioning problem and provides a methodology and a corresponding tool chain that allows application developers and architects to model their applications. MOCCA assumes that the architecture model, the deployment model and the deployment artifacts for the application are provided. A cloud distribution is a set of architectural components of the application that are to be moved to the same cloud and is derived based on the architecture model and deployment model. The actual provisioning of the cloud distribution in the target clouds is performed based on the automatic creation of provision clusters.[LFM$^+$11]

A customizable application is represented by an instance of the entity type Application Template, which consists of one or more instances of the Component entity type. A component has a name and type and may contain other components, is related to other components and is realized by exactly one Implementation and the type attribute of implementation indicates the main manner or technological basis used to realize the implementation. An implementation consists of zero or more Artifacts and an artifact has zero or more Variability Points with a Name and a Locator attribute. The Locator attribute is used to point directly into the artifact to distinguish the piece and variability point support users in customizing an application template by providing a list of potential values to choose from for variability point which is assigned a value of exactly one of the associated alternatives. The various types of alternatives are explicit which has a predefined value, free which allows the input of arbitrary values and visible, which points to a Visible property of a component. Visible Properties and Variability Points can be defined for the components of an application to support the specification of the parametrization aspects of a deployment, which will support an automatic provisioning of applications. To support an automatic installation of an application the Implementation and Artifacts of a component have to be defined. A generic orthogonal variability allows all variability of an application to be expressed in one model and is necessary as variability in one component might depend on the binding of other variability points of other components.[LFM$^+$11]

An architecture model of the application to be moved to the cloud has to be provided which is enriched with deployment information and rearranged into groups of components that belong into the same cloud creating a cloud distribution, auto provisioning is done. Finally, the cloud distribution and the combined architecture/deployment model annotated with the required implementation units are combined into a provision cluster which represents all the information needed to provision the rearranged application into its target clouds. To automatically compute an optimized cloud distribution hill climbing, simulated annealing, an evolutionary algorithm and a hybrid approach. Clouds are modeled as tuples of properties relevant for deciding the cloud distribution problem. The Architecture Modeler helps to model the architecture models; the Deployment Modeler specifies deployment topologies and models, middleware deployments, deployment relevant parameters and installation relevant artifacts, and the Cloud Distributor helps to split the application. The Provision Preparation component determines the corresponding provision cluster and the Customization Flow Generator generates a customization workflow that derives the properties required for provisioning and deployment of the rearranged application. [LFM+11]

MOCCA has a proper tool set for realising the proposed method for splitting applications in order to deploy various part of it on various clouds. But, MOCCA assumes that the architecture model, the deployment model and the deployment artifacts for the application are provided. So it rearranges the deployment topology of an application but does not provide the option for building topologies for applications from requirements. It seems to be a very effective method for migrating applications to cloud but does not seem to have support for deploying new applications.

4 Concept and Specification

In this chapter we first present a comparison of various algorithms we have studied for discovery of cloud services and then provide an overview of the system. In the second part of this chapter we specify the functional and non-functional requirement the system must fulfill, and provide a list of the use cases.

4.1 State-of-the-Art on Optimization Algorithms

In this section, we will list down the advantages and disadvantages of the following approaches and then compare and decide which one is the most suitable for our use.

4.1.1 ACO

The Advantages of this approach are :

- It has advantage of distributed computing. [KP13]
- It is robust and also easy to accommodate with other algorithms.[SGGB]
- When the graph may change dynamically, the ant colony algorithms can be run continuously and adapt to changes in real time.[SGGB]

Some of the shortcomings of this approach are :

- Though ant colony algorithms can solve some optimization problems successfully, we cannot prove its convergence.[YJB08]
- It is prone to falling in the local optimal solution because the ACO updates the pheromone according to the current best path.[ZZZ06]

4.1.2 Simulated Annealing

The Advantages of this approach are :

- It is relatively easy to code, even for complex problems.[KP13]
- It statistically guarantees finding an optimal solution.[KP13]
- SA can deal with nonlinear models, unordered data with many constraints.[KP13]
- It is versatile because it does not depend on any restrictive properties of the model. [KP13]

Some of the shortcomings of this approach are :

- It is very time consuming, especially if the cost function needs more computation. [KP13]
- SA is not that much useful when the energy landscape is smooth, or there are few local minima.[KP13]
- SA is a meta-heuristic approach, so it needs a lot of choices to turn it into an actual algorithm. [KP13]
- There is a trade-off between the quality of the solutions and the time needed to compute them.[KP13]
- More customization work needed for varieties of constraints and have to fine-tune the parameters of the algorithm.[KP13]
- The precision of the numbers used in implementation have a major effect on the quality of the result. [KP13]

4.1.3 Genetic Algorithm

Some of the main advantages of GA are :

- It always gives solution and solution gets better with time. [SGGB]
- It supports multi-objective optimization. [SGGB]
- It is more useful and efficient when search space is large, complex and poorly known or no mathematical analysis is available. [BS12]
- The GA is well suited to and has been extensively applied to solve complex design optimization problems because it can handle both discrete and continuous variables, and nonlinear objective functions without requiring gradient information. [HCDWV05]

A few limitations are :

- When fitness function is not properly defined, GA may converge towards local optima. [BS12]
- GA is not appropriate choice for constraint based optimization problem. [BS12]

Here we have discussed about the advantages and disadvantages of the various optimization algorithms discussed by us in the fundamentals chapter. This is done in order to be able to select the algorithm that best suits the requirements of our application. We see that GA is better for solving complex optimization problem involving large search spaces and without the availability of much mathematical analysis. Also it supports multi objective optimization and provides us a pareto optimal set of solutions. SA gives an optimal solution but is largely impacted by external factors and also is of not much use when the energy landscape is smooth. ACO can solve optimization problems but its convergence cannot be proven and also it is prone to falling into local optima more often then not. GAs also at times tend to converge towards local optima but this can be avoided by defining proper fitness functions. Although the solutions of GA gets better with time but it guarantees a solution even in lesser number of iterations and it works for nonlinear objective functions without requiring gradient information. All of these features of GA which suits the requirements of the system we want to design is the motivation behind using GA.

4.2 Optimal Discovery of Cloud Services

In this section, we would give an overview about the steps we are going to follow to complete the objective of discovering cloud services and topologies for an application. Algorithm 4 is a basic algorithm stating the same:

```
Receive the $\alpha$ or preliminary mu topologies from the user;
Analyze the topology to interpret requirements and hard constraints;
Apply hard constraints using queries to get compatible services;
Use Genetic Optimization on compatible services to get Pareto optimal sets of deployment
alternatives;
Use feature models to get middleware alternatives;
Combine the deployment alternatives and middleware alternatives and build a set M of
viable $\mu$ topologies;
Show the $\mu$ topologies to the developer;
Persist the topology selected by the developer;
```

Algorithm 4: Optimal Discovery of Cloud Services

4.2.1 Discovery, Selection, and Composition of Cloud Services - Genetic Algorithm

Concept

To implement the genetic algorithm, we first need to design a basic structure for the chromosome which we are going to use and the description of the various genes taken into account while building the chromosome. The basic structure of the chromosome is shown in the figure 4.2. The genes that are taken into consideration while building the chromosomes of population are:

- Service Type (ST) : It defines the cloud service provider or physical server that we are taking into account. Ex - AWS EC2,AWS RDS, IBM Z Series
- Provider Instance (PI) : It defines the instance type of the service provider considered. e.g. - m3.xlarge in AWS EC2
- Operating System (OS) : It tells us about the operating system used on the selected service provider.
- Number of Instances (NI) : It tells us about the number of instances of the particular service being used.

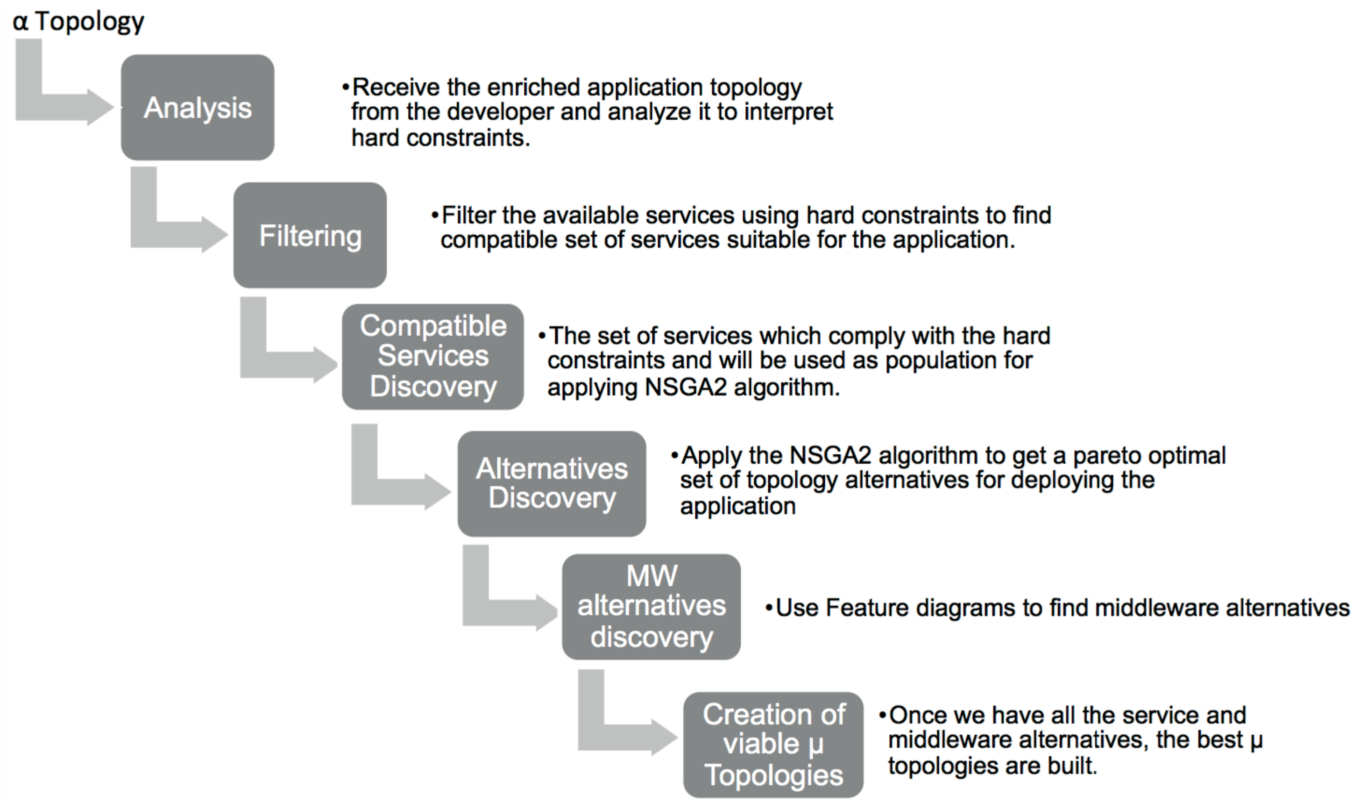

α Topology
Analysis
•Receive the enriched application topology from the developer and analyze it to interpret hard constraints.
Filtering
•Filter the available services using hard constraints to find compatible set of services suitable for the application.
Compatible Services Discovery
•The set of services which comply with the hard constraints and will be used as population for applying NSGA2 algorithm.
Alternatives Discovery
•Apply the NSGA2 algorithm to get a pareto optimal set of topology alternatives for deploying the application
MW alternatives discovery
•Use Feature diagrams to find middleware alternatives
Creation of viable μ Topologies
•Once we have all the service and middleware alternatives, the best μ topologies are built.

Figure 4.1: Steps for Discovery of cloud services.

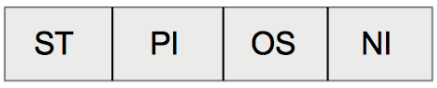


Figure 4.2: Generic Structure of the Chromosome.

The above section shows how the chromosome for a particular service would look like but an application would ideally have more than one service. So now in the figure 4.3 we will see how a deployment alternative of an application with more than one service could be represented as a genotype. Here in order to identify the various services, we have to introduce an additional gene :

- Service Number (SN) : It indicates the various services with a unique number.

The Genetic algorithm that we are using for the thesis is Non-dominated Sorting Genetic Algortithm - II (NSGAII). The figure 4.4 gives a brief overview of the working of NSGAII algorithm which we shall explain in detail in the remaining of this section.

In terms of genetic algorithms, fitness refers to the quality of the members of a population basing on the objectives taken into consideration. In NSGAII, random population are created and sorted based on non domination. Each solution is given a fitness, 1 being the best. A solution is said to be non dominated, if it is better in atleast one objective and worse in no objectives. So if we compare two solutions x and y basing on 2 objectives namely cost and

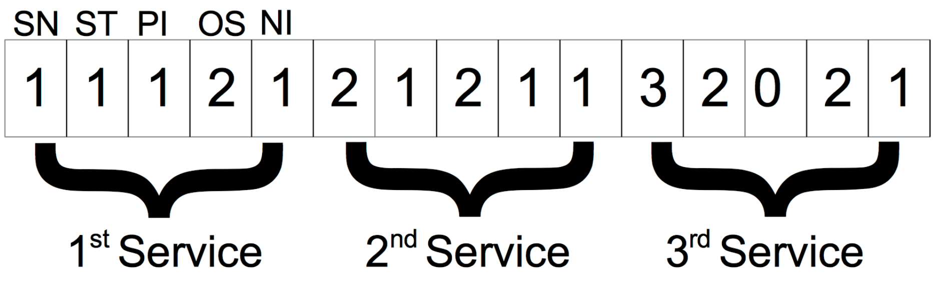


Figure 4.3: Deployment Alternative of Application coded as a Genotype.

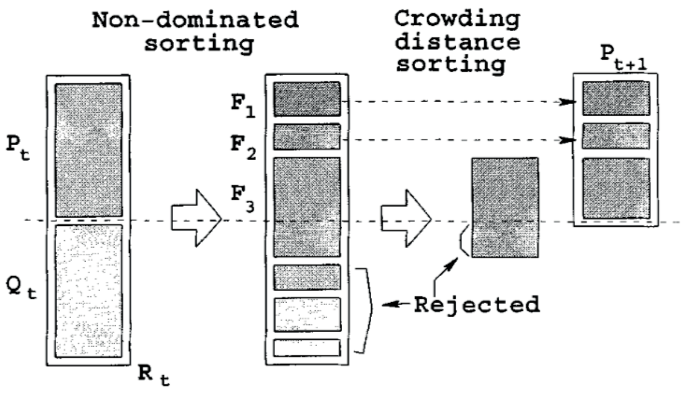


Figure 4.4: NSGAII Procedure.[DPAM02]

availability, x dominates y if it is at least cheaper or more available then y but not costlier and not lesser available. [DPAM02]

Considering a pool of solutions, we need to calculate 2 entities:

- domination count (n_p) - It refers to the number of solutions that dominate the solution p.
- A set (s_p) of solution that the solution p dominates.

All the solutions which have minimum domination count are put into the first Non Dominated set or the First front(F_1). For each solution with n_p=0, we visit each member q of its set s_p and reduce its domination count by one. In doing so, if for any member the domination count becomes zero, we put it in a separate list and all such members belong to the second non dominated front. Now, the above procedure is continued with each member of second front and the third front is identified. This process continues until all fronts are identified.[DPAM02]

Once we have the population divided into various non domination fronts, we move on to the main loop of the algorithm where we implement the genetic operations to select the pareto optimal set of solutions. Initially, a random parent population P_0 is created and each solution is assigned a fitness corresponding to its non domination front. Then the usual binary tournament selection, recombination, and mutation operators are used to create a offspring population Q_0 of size N. In NSGAII, elitism is introduced by comparing current population with previously found best non dominated solutions, the procedure is different after the initial generation[DPAM02].

Let us consider the kth generation or iteration of the algorithm. Here, after generating Q_k from P_k, we combine them to get a population $R_k = P_k \cup Q_k$ of size 2n. Then the population R_k is sorted according to non domination. All current and previous generations are including in R_k hence ensuring elitism. Now, solutions belonging to the best non dominated set F_1 are best solutions in the combined population and must be emphasized more than any other solution in the combined population. If the size of F_1 is smaller than n, we definitely choose all members of the set for the new population. The remaining members of the population are chosen from subsequent non dominated fronts in the order of their ranking. This procedure is continued until no more sets can be accommodated. In the process, a point of time will mostly come when we cannot accommodate the complete population of a particular front. In this case we have to sort the population of that particular front and select the best few so as to select exact n number of elements. In such cases, we need to calculate a value called as crowding distance[DPAM02].

The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude. Thereafter, for each objective function, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with other objective functions. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance. The figure 4.5 shows the algorithm for crowding distance computation[DPAM02].

```
crowding-distance-assignment($\mathcal{I}$)
$l = |\mathcal{I}|$                                              number of solutions in $\mathcal{I}$
for each $i$, set $\mathcal{I}[i]_{\text{distance}} = 0$                    initialize distance
for each objective $m$
  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$                                   sort using each objective value
  $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$                so that boundary points are always selected
  for $i = 2$ to $(l-1)$                                   for all other points
    $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$
```

Figure 4.5: Algorithm for computation of Crowding Distance.[DPAM02]

Example- MediaWiki Application

Let us consider an example for deploying the media wiki application. The topology of the application is shown in figure 4.6. This figure shows a representation of how parts of the application can be deployed on various cloud offerings and some parts on physical machine. Now we shall explain how the various topologies can be formed basing on the available offerings. The objectives that we use for the implementation of this multi objective algorithms are Cost and Availability. For this purpose, we select some examples of the offerings with generated values for cost and availability and we assign the various offerings codes from A-G(for ease of reference when explaining the example) as seen in the table in figure 4.7

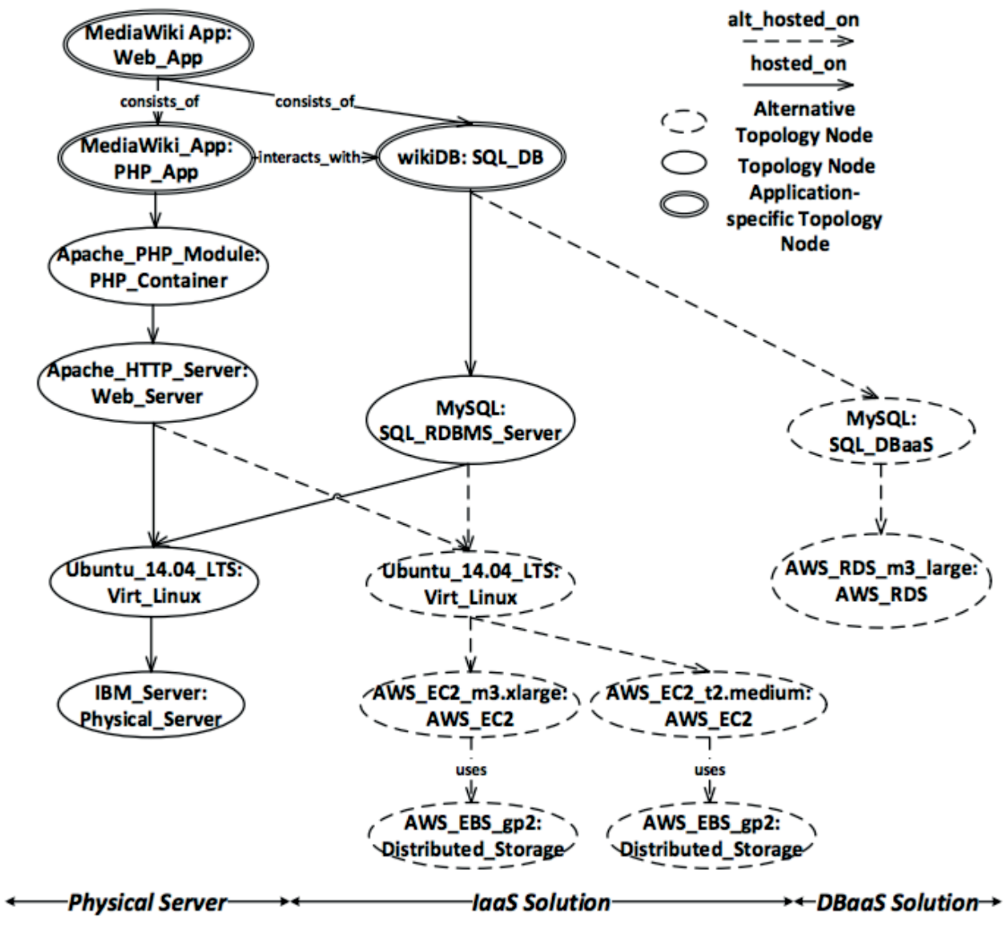
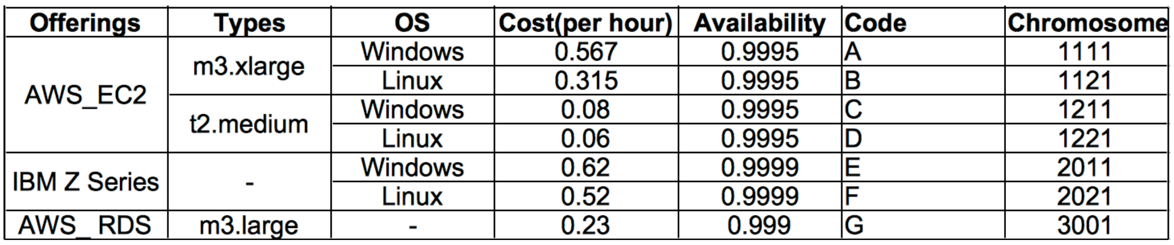

alt_hosted_on
hosted_on
Alternative Topology Node
Topology Node
Application-specific Topology Node
MediaWiki App: Web_App
consists_of
consists_of
MediaWiki_App: PHP_App
interacts_with
wikiDB: SQL_DB
Apache_PHP_Module: PHP_Container
Apache_HTTP_Server: Web_Server
MySQL: SQL_RDBMS_Server
MySQL: SQL_DBaaS
Ubuntu_14.04_LTS: Virt_Linux
Ubuntu_14.04_LTS: Virt_Linux
AWS_RDS_m3_large: AWS_RDS
IBM_Server: Physical_Server
AWS_EC2_m3.xlarge: AWS_EC2
AWS_EC2_t2.medium: AWS_EC2
uses
uses
AWS_EBS_gp2: Distributed_Storage
AWS_EBS_gp2: Distributed_Storage
Physical Server
IaaS Solution
DBaaS Solution

Figure 4.6: The Media Wiki Application Topology.[SALS15]



Offerings	Types	OS	Cost(per hour)	Availability	Code	Chromosome
AWS_EC2	m3.xlarge	Windows	0.567	0.9995	A	1111
		Linux	0.315	0.9995	B	1121
	t2.medium	Windows	0.08	0.9995	C	1211
		Linux	0.06	0.9995	D	1221
IBM Z Series	-	Windows	0.62	0.9999	E	2011
		Linux	0.52	0.9999	F	2021
AWS_ RDS	m3.large	-	0.23	0.999	G	3001

Figure 4.7: Example of some Offerings.[San15]

Now first we calculate the domination count(n_p) and the set(s_p) of solutions each solution dominates.

For A: There is no solution that A dominates as for none of the solutions, A has better cost

and availability. The cost of A is lesser than only E but the availability is not better. Hence s_A $= \{\}$
The solutions B,C,D have similar availability as compared to A but offer better cost and hence dominate A and the solution F has better cost as well as availability as compared to A. Hence $n_A = 4\ \{B, C, D, F\}$
Similarly, For B: $s_B = \{A\}$, $n_B = 2\ \{C, D\}$
For C: $s_C = \{A, B, G\}$, $n_C = 1\ \{D\}$
For D: $s_D = \{A, B, C, G\}$, $n_D = 0\ \{\}$
For E: $s_E = \{\}$, $n_E = 1\ \{F\}$
For F: $s_F = \{A, E\}$, $n_F = 0\ \{\}$
For G: $s_G = \{\}$, $n_G = 2\ \{C, D\}$

So here the non dominated fronts can be defined as :
First Front $(F_1) = \{D, F\}$
Second Front $(F_2) = \{C, E\}$
Third Front $(F_3) = \{B, G\}$
Fourth Front $(F_4) = \{A\}$

Now we demonstrate an example where we take into account an application needing just one service for the ease of explanation. So we start the process with randomly selected parent population P_0. In figure 4.8, we can see the randomly selected parent populations be considered for this example.

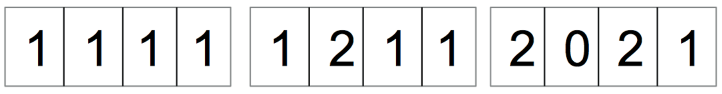


Figure 4.8: The Randomly Selected Parents.

Here we have 3 randomly selected parents P_0(A,C,F) so we have odd number of elements and hence the best one among them should be replicated to mate with two other parents to ensure better offspring. In this case F is the fittest parent and has to be replicated. If the randomly selected parent population had even number of elements, we can mate them in 2 pairs to produce offspring. So after crossover and mutation, we get the set of offspring Q_0. The offspring are the services (B,D,E). Here we get 2 instances of E as offspring so use it only once for the next step.

So now we have the population $R_0 = \{A, B, C, D, E, F\}$ from which we have to select the best 3 to be the parents for the next step. So now we look in the non dominated fronts and select 2 elements D,F from the first front and the set becomes $\{D, F\}$. Now in the second front, we have two elements C,E but we have to select only one of them to complete the set of 3 elements so here we use the crowding distance operator to calculate the crowding distance. Here we have 2 objectives namely cost and availability and for calculating crowding distance, we need to arrange the values in ascending order and apply the formula :

$I[i]_{\text{distance}} = (I[i]_{\text{distance}} + (I[i+1].m - I[i-1].m) / (f_m^{\max} - f_m^{\min}))$

Now we arrange the services in ascending order of cost as a function where minimum cost is the best. So we arrange it in the order as maximum cost first and then go on decreasing till the minimum $\{E, A, F, B, G, C, D\}$. As per the definition now we calculate the crowding distance of the offerings C and E as

For C, $I[3]_{\text{distance}} = (0 + (0.06 - 0.23) / (0.06 - 0.62) = \frac{17}{56} = 0.3035$

Similarly for E, $I[5]_{\text{distance}} = 0 + \infty = \infty$ (Solution with boundary values are always selected)

Now considering the objective availability, the services will be ordered as $\{G, A, B, C, D, E, F\}$. Now we use the same formula to calculate the total crowding distances

For C, $I[1]_{\text{distance}} = 0.3035 + (0.0004) / (0.0009) = 0.7479$

For E, $I[5]_{\text{distance}} = \infty$.

So here the crowding distance value is more for E as compared to C, so E gets selected to fill the set and hence the final set to be the parent population for the next step, P_1 will be D, F, E and the same procedure repeats.

Also, there is a special case we have to consider when taking into consideration offerings which hides the OS layer from the user and have 0 for the field OS as in RDS or have no PI type and have value 0 for it as in IBM Z series. We will have a condition that whenever we find a value 0 for the OS or PI, the crossover has to be done for more than one genes. In case we have the value 0 for PI then we have to crossover both the ST and PI. In case we have a value 0 for the OS, then we need to crossover both PI and OS. And suppose we need to make a crossover between the above mentioned 2 special cases, then we have to do a crossover of all the 3 genes; ST,PI and OS.

This processes are repeated for a large number of times before we reach at a final result which would be the set of optimal services which would be something that shown in figure 4.9

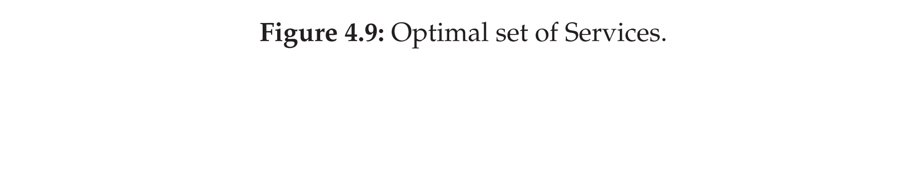


Figure 4.9: Optimal set of Services.

4.2.2 Feature Diagrams for Discovery of Middleware Alternatives

We will use the feature diagrams for the purpose of discovery of middleware. We will use an extensible framework where current research on VM automated analysis might be developed and easily integrated into a final product. The framework is built following the SPL paradigm supporting different variability meta models, reasoners or solvers, analysis questions and reasoner selectors, easing the production of customized VM analysis tools. [TBRC+08]

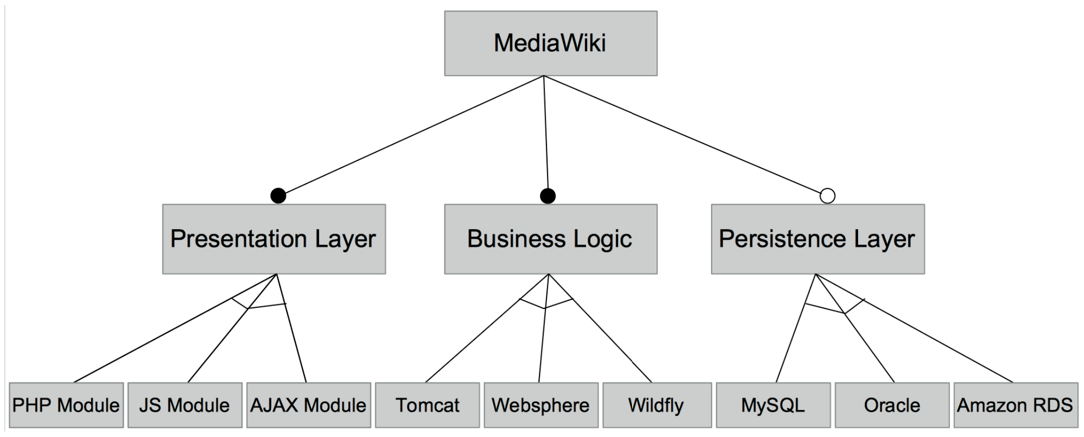


Figure 4.10: Sample Feature Diagram.

Feature models have hierarchical tree like structures and we can specify how we will select from the many available children of a node by using various relationships like And,Or,Alternative,etc. The middleware required for the Mediawiki application can be broadly categorized into those required for presentation layer, business logic layer and persistence layer out of which the first two are mandatory and the third can be marked as optional in the feature diagram. It can be seen in the figure 4.10 which has been drawn as per the standards discussed in the section 2.7. As per the diagram, it is mandatory to select one middleware each for the presentation and business logic layers and if needed one can also be selected for the persistence layer.

4.3 System Requirements

This section aims to provide the set of requirements of the application which were extracted by taking into consideration the problem statement and previous works. We tried to understand the previous works which aimed at solving problems similar to what we have at hand. After making a proper analysis of other works and then relating them to our problem, we arrived at the following requirements which have been discussed below.

4.3.1 Functional Requirements

- Application Topology and Requirements Analysis : The application profile provided by the developer are analyzed in order to find out the hard constraints needed to filter the services. The system must interpret what exactly the developer requires to deploy the various tiers of his application and then try to find out the parameters which could be used to filter services.

- Filtering of Services: The hard constraints discovered in the previous step are used to filter out the services which suit the application's requirements from the pool of services available.
- Application Distribution Cloud Service Alternatives Discovery: After the system gets the filtered set of services as per the requirements, NSGAII algorithm discussed above is used to discover the pareto optimal set of service alternatives among all the services that suit the requirements.
- Application Middleware Discovery: After the service alternatives have been discovered,the middleware alternatives are discovered using feature diagrams. Here, the system searches through a tree of various alternatives for deploying each tier of the solution and finds out the suitable deployment options.
- Construction of μ topologies: Once the service alternatives and the middleware alternatives are found, the system must build the set of concrete μ topologies which can be given to the developer.
- Persist the μ topologies: Once the pareto optimal set of μ topologies are ready, we need to return it to the developer and persist the one in a database which is selected by the developer.
- Discover Compatible topologies: When the set of requirements for the application are known, there is a possibility to search for an already existing topology that fulfils the requirements. In this case, developer can reuse the topology if it suits the application to be deployed.

4.3.2 Non Functional Requirements

- Performance : The topology discovery perform very efficiently by fulfilling the required steps of filtering services and discovering optimal set of services.
- Availability : The framework should always be able to discover topologies for the requirements provided by the developers for the applications or retrieve the topologies from the available set of topologies if suitable for the application.
- Security : The implementation of the framework should not violate the security of used tools like winery,nefolog,etc.
- Compatibility : The implemented service discovery framework should be compatible with the topology modelling tool such as Winery.

- Consistency: The topology discovery and optimization framework should work consistently on the various requirements and objectives provided by the application developer. The user interface and all the operations should always behave in a similar fashion. The framework should also be consistent while
- Usability: The user interface should be easy to use and informative for the developers to provide the data and query for whatever they need. There should be proper documentation of the framework which should be complete and self explanatory.

4.4 Use Cases and Roles

The system contains the application developer as an actor. The application developer roles are described in the following subsection. The identified use cases of the use case diagram in figure 4.11 are also described in the subsequent subsection.

4.4.1 Application Developer

The application developer is the actor responsible for the roles described below.

- Specifying application requirements: The developer provides the α or a preliminary μ topology of his application or in some cases developer may provide just the requirements of the application.
- Specifying the objectives and the allowable limits for them: The developer should provide what are the main objectives which is to be taken into consideration when selecting the pareto optimal set of alternatives. In this thesis, we take into consideration two objectives namely cost and availability.

4.4.2 Description of Use Cases

Name	**Enrich Application Topology**
Goal	The developer wants to enrich the application topology as per the requirements.
Actor	Application Developer
Pre-Condition	The application developer has access to the winery system and has the application requirements ready.
Post-Condition	The developer gets an enriched topology.
Post-Condition in Special Case	Here goes the post-condition in special case

Normal Case	1. Developer selects *Enrich Topology* and specifies the partial topology of the application. 2. The topology modelling framework enriches the topology and returns it back to the user.
Special Cases	1a. The provided partial topology does not match the allowed format. a) System shows *Enter Valid data* message.

Table 4.1: Description of Use Case *Enrich Application Topology*.

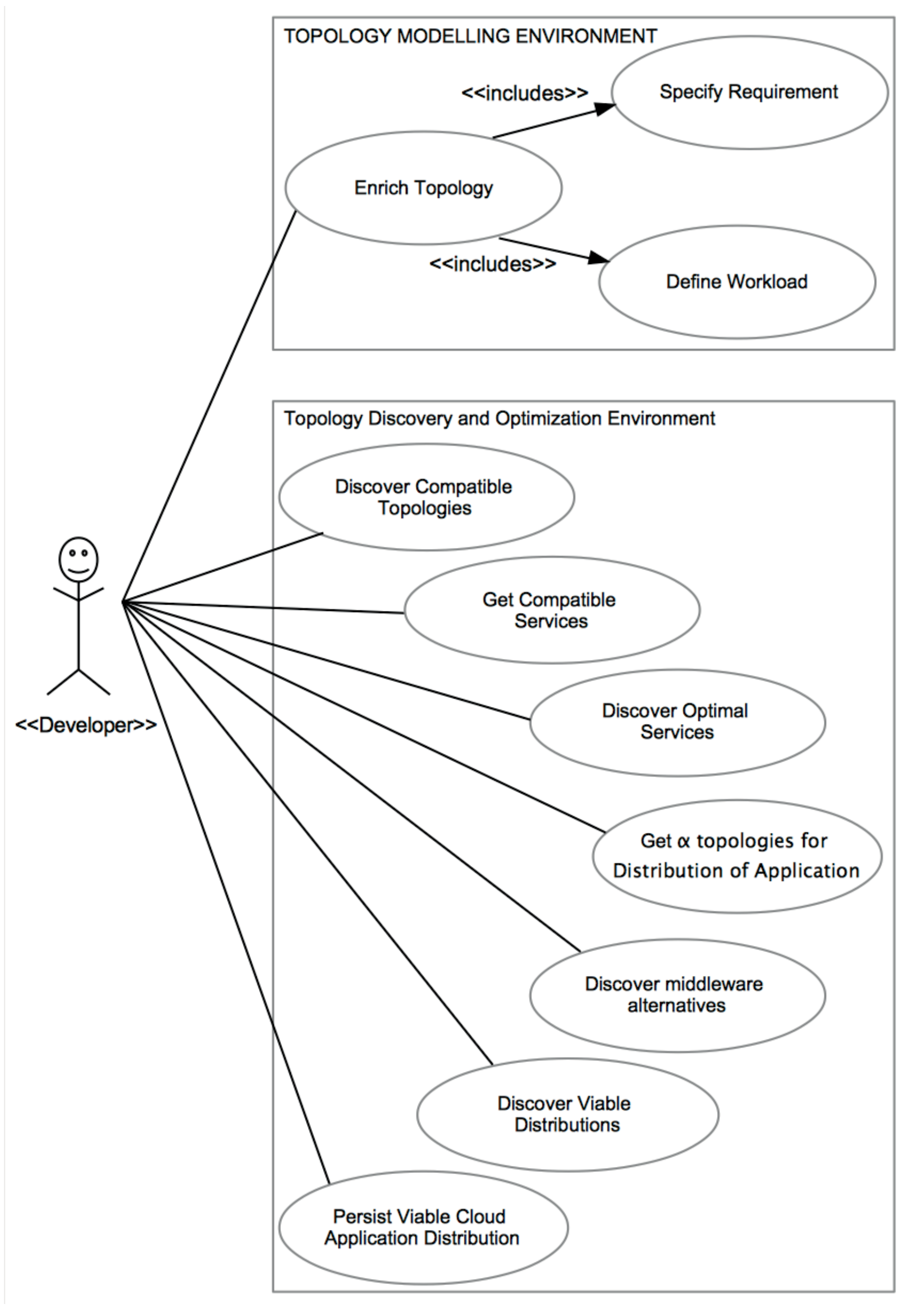

TOPOLOGY MODELLING ENVIRONMENT
<<includes>>
Specify Requirement
Enrich Topology
<<includes>>
Define Workload
Topology Discovery and Optimization Environment
Discover Compatible Topologies
Get Compatible Services
Discover Optimal Services
Get α topologies for Distribution of Application
Discover middleware alternatives
Discover Viable Distributions
Persist Viable Cloud Application Distribution
<<Developer>>

Figure 4.11: Use Case Diagram for Developer.

Name	**Discover Compatible Topologies**
Goal	The developer searches for the topologies in the database which may be suitable for the requirements of the application.
Actor	Application Developer
Pre-Condition	The developer has access rights to the database and has the application requirements ready.
Post-Condition	The developer may finds topology matching the application requirements.
Post-Condition in Special Case	Here goes the post-condition in special case
Normal Case	1. Developer selects *Search for Topology* and specifies the requirements. 2. The framework looks for the relevant topologies in the database and returns a list of topologies which may match the requirements.
Special Cases	1a. The requirements provided are not in correct format or incomplete. a) System shows *Enter Valid Requirements* message.

Table 4.2: Description of Use Case *Discover Compatible Topologies*.

Name	**Get Compatible Services**
Goal	The developer sends a request to the framework to fetch all the services compatible with the requirements of the application.
Actor	Application Developer
Pre-Condition	The developer must have access to the system and should have provided the application requirements.
Post-Condition	The system returns a set of services which fulfil the requirements for the application.
Post-Condition in Special Case	The system does not return any services.
Normal Case	1. Developer provides the requirements and the objectives to consider for optimization to the system. 2. Developer selects the *Get Compatible Services* option. 3. The system analyzes the requirements and finds out the hard constraints. 4. The system then queries the Nefolog basing on the hard constraints and gets a set of filtered services and returns it to the user.
Special Cases	3a. Some of the requirement provided are vague. a) System shows *Enter Valid Requirements* message.

Table 4.3: Description of Use Case *Get Compatible Services*.

Name	**Discover Optimal Services**
Goal	The developer sends a request to the framework to discover all the pareto optimal services for each tier of the application.
Actor	Application Developer
Pre-Condition	The developer must have access to the system and should have provided the application requirements and the objectives for optimisation to the system.
Post-Condition	The system returns a set of services which are best suited as per the requirements for each tier of the application.
Post-Condition in Special Case	The system does not return any services or returns the services only for some parts of the application.
Normal Case	1. Developer provides the requirements and the objectives to consider for optimization to the system. 2. Developer selects the *Discover Optimal Services* option. 3. The system analyzes the requirements and finds out the hard constraints. 4. The system then queries the Nefolog basing on the hard constraints and gets a set of filtered services for each tier. 5. Then the system uses the optimizer to find out the pareto optimal set of services for each tier and returns it to the developer.
Special Cases	3a. Some of the requirement provided or the objectives to be considered for optimization are vague. a) System shows *Enter Valid Requirements and Objectives* message.

Table 4.4: Description of Use Case *Discover Optimal Services*.

Name	**Get alpha Topologies**
Goal	The developer sends a request to the framework to discover all the alpha topologies for the application.
Actor	Application Developer
Pre-Condition	The developer must have access to the system and should have provided the application requirements and the objectives for optimisation to the system.
Post-Condition	The system returns a set of pareto optimal alternative alpha topologies for the application.
Post-Condition in Special Case	The system does not return a set of pareto optimal alternative alpha topologies for the application.
Normal Case	1. Developer provides the requirements and the objectives to consider for optimization to the system. 2. Developer selects the *Discover Optimal Services* option. 3. The system analyzes the requirements and finds out the hard constraints. 4. The system then queries the Nefolog basing on the hard constraints and gets a set of filtered services for each tier. 5. Then the system uses the optimizer to find out the pareto optimal set of services for each tier. 6. Once the services are discovered for each tier, the system combines them and builds alternative alpha topologies and returns them to the developer.
Special Cases	3a. Some of the requirement provided or the objectives to be considered for optimization are vague. a) System shows *Enter Valid Requirements and Objectives* message.

Table 4.5: Description of Use Case *Get alpha Topologies*.

Name	**Discover Middleware Alternatives**
Goal	The developer wants the system to find out all the middleware alternatives feasible for the application.
Actor	Application Developer
Pre-Condition	The developer must have access to the system and should have provided the application requirements to the system.
Post-Condition	The system returns a set of best suitable middleware alternatives for the application.
Post-Condition in Special Case	The system does not return a set of best suitable middleware alternatives for the application.
Normal Case	1. Developer provides the requirements for the application to the system. 2. Developer selects the *Discover Middleware Alternatives* option. 3. The system analyzes the requirements and finds out the hard constraints. 4. The system uses feature diagram to search through the middleware alternatives and find out the best suitable ones and return them to the developer.
Special Cases	3a. Some of the requirement provided are not appropriate. a) System shows *Enter Valid Requirements* message.

Table 4.6: Description of Use Case *Discover Middleware Alternatives*.

Name	**Discover Viable Distributions**
Goal	The developer sends a request to the framework to discover all the viable distributions or mu topologies for deploying the application.
Actor	Application Developer
Pre-Condition	The developer must have access to the system and should have provided the application requirements and the objectives for optimisation to the system.
Post-Condition	The system returns a set of near optimal alternative mu topologies for the application.
Post-Condition in Special Case	The system does not return a set of near optimal alternative alpha topologies for the application.
Normal Case	1. Developer provides the requirements and the objectives to consider for optimization to the system. 2. Developer selects the *Get mu Topologies* option. 3. The system analyzes the requirements and finds out the hard constraints. 4. The system then queries the Nefolog basing on the hard constraints and gets a set of filtered services for each tier. 5. Then the system uses the optimizer to find out the pareto optimal set of services for each tier. 6. When the services are discovered for each tier, the system combines them and builds alternative alpha topologies. 7. The system then searches for the middleware alternatives. 8. Once all the alpha topologies and middleware alternatives are discovered, the mu builder builds the mu topologies and gives them back to the developer.
Special Cases	3a. Some of the requirement provided or the objectives to be considered for optimization are vague. a) System shows *Enter Valid Requirements and Objectives* message.

Table 4.7: Description of Use Case *Discover Viable Distributions*.

Name	**Persist Selected Topology**
Goal	The developer wants to persist one of the topologies in the database.
Actor	Application Developer
Pre-Condition	The developer should have access to the system and successfully discovered topologies for deploying the application at hand.
Post-Condition	The selected topologies get persisted in the database.
Post-Condition in Special Case	The topologies may not be persisted in the database.
Normal Case	1. The developer has a set of topologies out of which he marks the topology to persist and selects *Persist Topology*. 2. The system persists the topology in the database.
Special Cases	

Table 4.8: Description of Use Case *Persist Selected Topology*.

4.5 System Overview

In this section, we provide an overview of the system we are going to design for the purpose of discovery, generation and persistence of cloud application topologies. This topologies will help the developers to deploy their applications such that parts of the application can be deployed on cloud based services while some others on physical machines. We design a framework which accepts the requirements and objectives of the application from the user as input. The system analyzes the requirements and hard constraints are interpreted which are used to filter services from those available in the knowledge base.

As shown in Figure 4.12, the system that we have designed for this purpose basically comprises of a topology modeling framework and a topology discovery and optimization framework. The topology modeling framework helps us enrich the topologies that we receive from the application developers or build the topologies from the requirements we get from the developers. The topology discovery and optimization framework does the task of filtering viable services from knowledge base as per the application requirements. It also helps us to find the optimal set of services from the filtered services and discover the suitable middleware required for the application. It also helps us to persist the topologies in the topology registry or search for existing topologies which would be suitable for the deployment of the application.

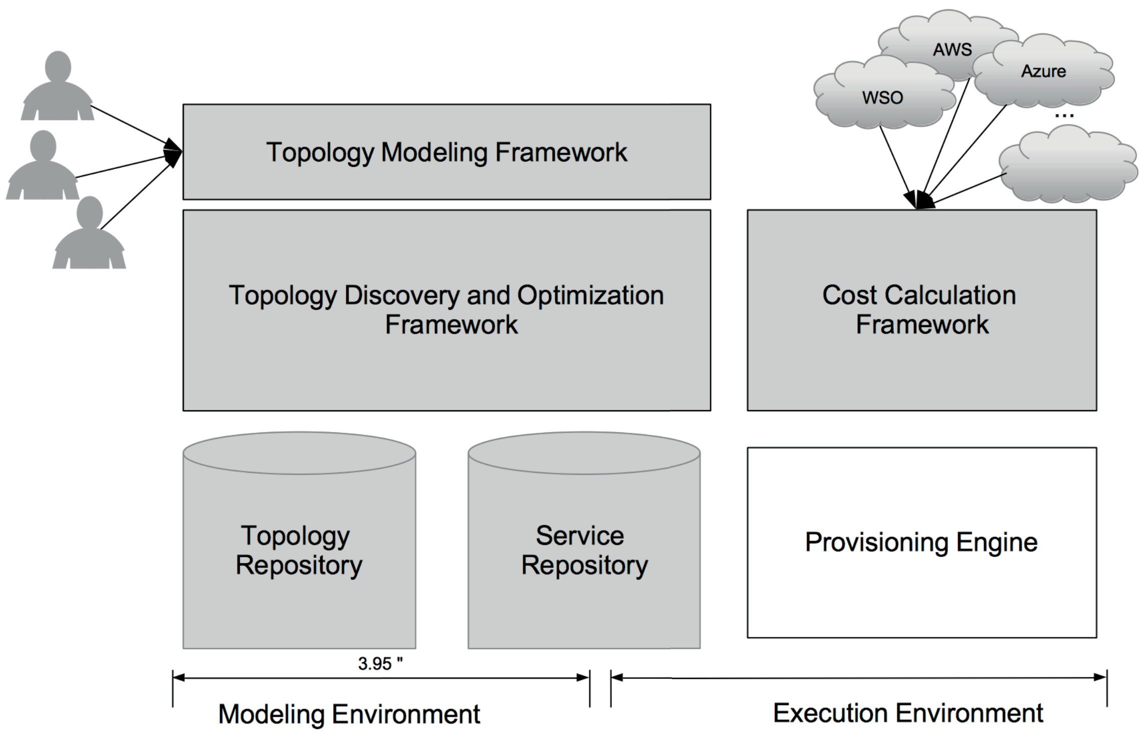


Figure 4.12: System Overview.

5 Design

In this chapter we present the architectural solution taken into account to build the system which fulfills the requirements specified in Chapter 4. We present the general architecture stating the main components of the system.

5.1 General Architecture

The system analyzes the requirements, finds out best suitable services as well as middleware solutions and designs topology alternatives for deployment of applications. The Winery modeling system and Nefolog knowledge bases are used to support the framework which will be used for discovery and generation of topologies as shown in Figure 5.1. They are described below:

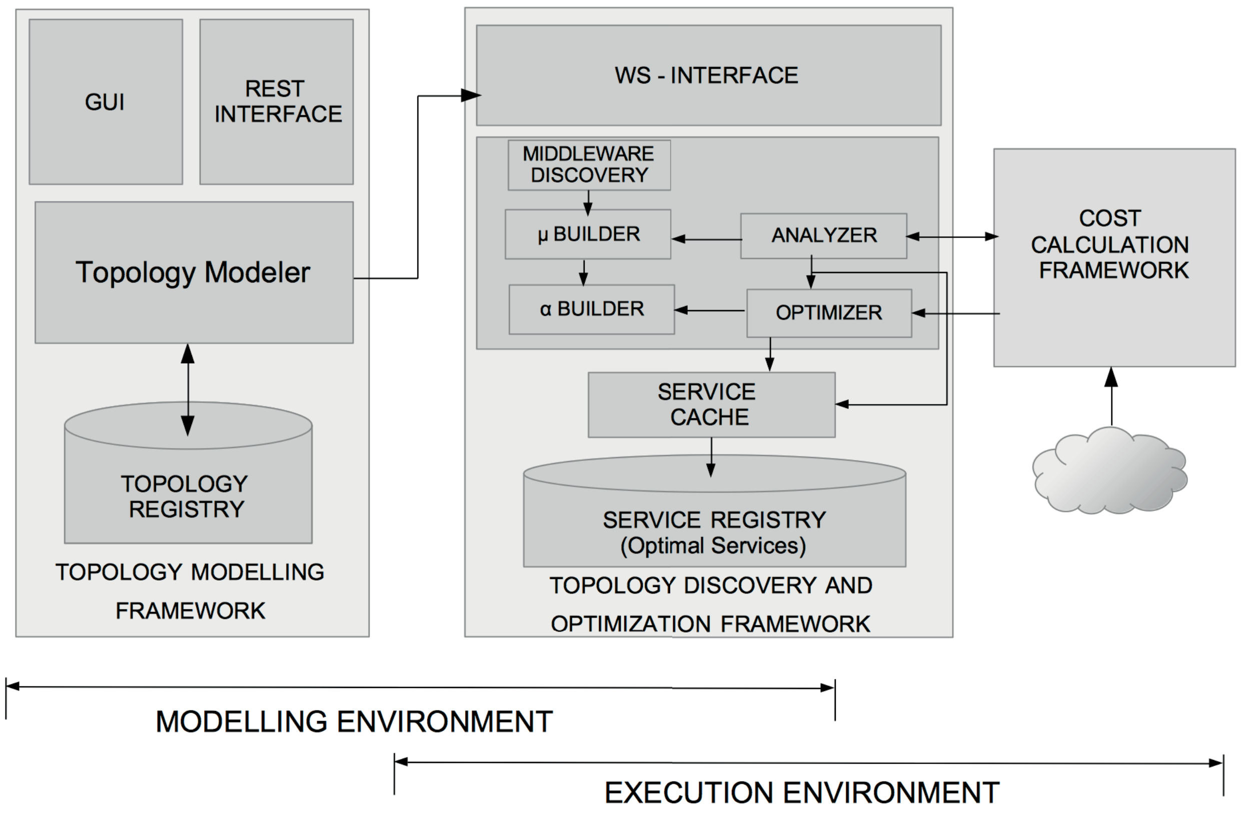


Figure 5.1: General Architecture of the System.

- **Topology modelling Framework**:We use the web based graphical modelling tool for TOSCA-based applications called Winery [KBBL13]. Application topologies and man-

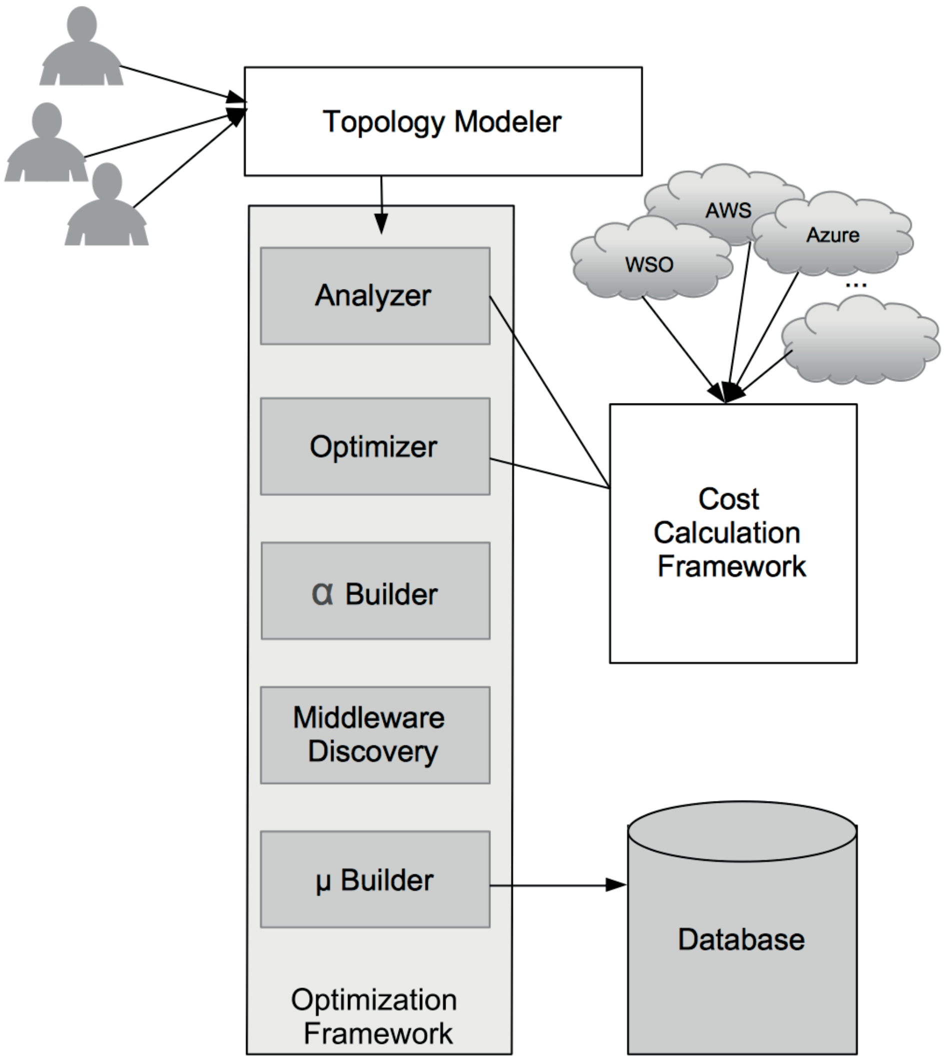


Figure 5.2: Data Flow within the System.

agement plans are modelled using it. In this thesis, the graphical user interface of Winery is extended to include an additional part for receiving application requirements and workload from the developer. The workload is specified under the properties of application service template.

The Topology Modeler creates graph based visual application topology model by using node and relationship template in the service template. The Topology Modeler attaches relationship constraint, deployment artifact, requirement, capability and policy to the node and relationship template of the topology [KBBL13]. In this thesis, the graphical user interface of the Topology Modeler is extended so that the application developer can specify the application performance requirements. The performance requirements are specified in the policy template of specific policy type[Gan]. Subsequently, the application developer can attach them as policies to the individual application component or node template level of the application topology. Each policy has specific policy template and type.

- **Cost calculation Framework**: Nefolog[XA+13] contains knowledge base and collection of decision support services which can be represented as JSON or XML and are defined by different URIs which can also present the requirements of users. Knowledge Base is a relational database and data are organized by tables which are linked by foreign keys. In this thesis, we are going to use the Nefolog to retrieve the viable services fitting the requirements of the application. Also, we are going to retrieve the cost for the services from the Nefolog system which will be used as a parameter for the determination of optimal set of services.

- **Optimization Framework**: The optimization framework is designed to get the requirements from the user and finds out the optimal α and μ topologies for the application. A detail insight into the processes can be found in the Figure 5.2 which shows the interaction between various parts of the framework and also the Topology Modelling and Cost Calculation frameworks.

  As we can see in the Figure 5.2, the framework receives the requirements for the application from the topology modeler. The analyzer then interprets the hard constraints and fetches the compatible services from the cost calculation framework. The compatible services are passed on to the Optimizer which uses the NSGA2 algorithm to find the pareto optimal set of services.Once the pareto optimal services are discovered for each tier, they are used to build pareto optimal set of α topologies. Then the middleware alternatives are found for the application. Finally the μ builder is used to build concrete application deployment alternatives. It uses the pareto optimal α topologies and the middleware discovered in previous steps to build the μ topologies or application deployment alternatives which are returned to the user. The user then selects one of the application deployment alternatives to use for his application which is then persisted in the database. The topology which is persisted can be reused later for another application with similar requirements.

5.2 Resource Model

In the section above, the architecture of the system exposes the resources like: topologies, services and middleware. The other resources like hard constraints, sub topologies, requirements and filters are not explicitly defined but are dealt in detail in the next section. A REST Application Programming Interface (API) consists of an assembly of interlinked resources which are known as the REST API's Resource model [Mas11]. Resources for a REST-based service must be defined as, *REST uses a resource identifier to identify the particular resource involved in an interaction between components*,[Fie00].

The resource model represents the resources and the relations among them as: topologies, sub topologies, application specific sub topologies, reusable sub topologies, requirements, hard constraints, services, middleware and filters. We use a class diagram to represent the resource model which can be seen in figure 5.3. The relation between the resources is depicted using a class diagram. Concrete application topologies are made from sub topologies which are of two types application specific and reusable. It goes on to show that the reusable sub toplogies consists of services and middleware and the compatible services are discovered by using filters on all the available services.

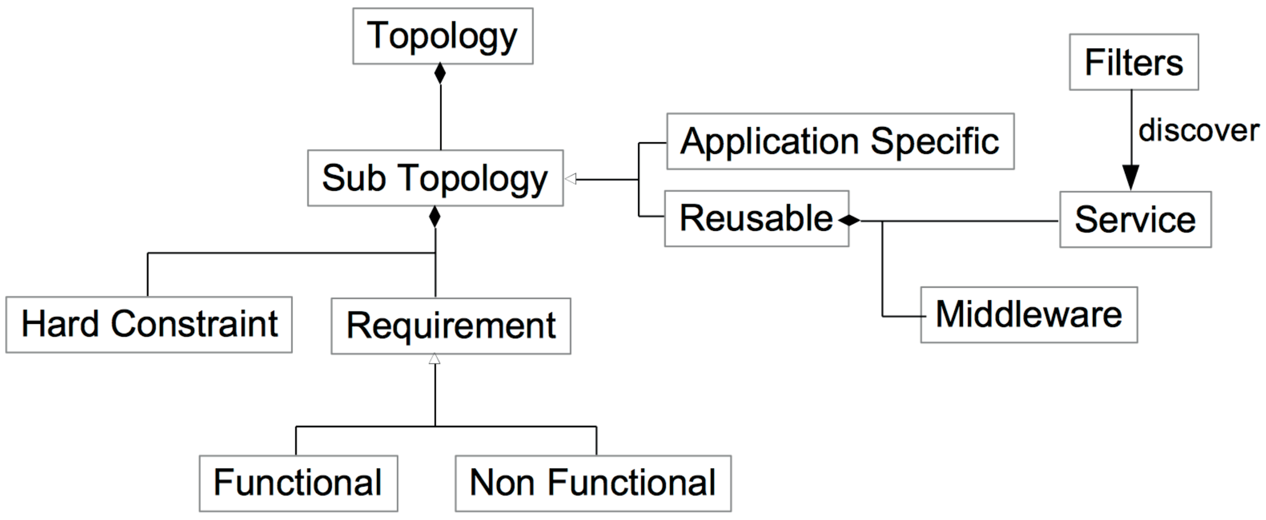


Figure 5.3: Resource Model representation using Class Diagram

5.3 RESTful API

REST API design rules are dealt in detail in the previous section 2.8.1. The need for following the design rules mainly provides consistency for the design and sticking to standards make it easier for being used.

5.3.1 API Design

The resources for this prototype can be basically explained as one group for modelling and other for execution. The first one deals with the modelling of α and μ topologies and enriching existing topologies. The one for execution deals with the main tasks of the optimization framework like analyzing the requirements, building optimal deployment alternatives, etc. To address each resource uniquely, an easy to understand URI is assigned for each.

- The first part of the URI shows which resource the client is addressing to. Hence, the first part is going to be /modelling, /execution. These URIs represent the modelling environment and the execution environment respectively.
- The second part of the URI shows the individual operation of the environment which is to be carried out, e.g. /execution/analyze would be for analyzing the requirements.

The proper use of HTTP methods GET, POST, PUT and DELETE has been defined for the resources. The description about the effect a HTTP request will have on the resource, are described below:

- /resource
  - GET: shows a limited list of the resources on the system. If the URI contains a query which are defined later in this section, the list of resources returned will meet the criteria specified in the URI. This is accessible to all the users in the prototype we design.
  - POST:adds a new resource of the specific type to the system taking the information embedded in the body of the request.This method will be used to persist the topology selected by the developer.
  - PUT: This method is applicable to the developer for providing the requirements and workload for the applications.
  - DELETE: This can be used to remove a resource from its parent.
- /resource/x (x=resource identifier)

We make sure to stick to the rules specified in the section 2.8.1 to design the URI and use only nouns for the expression. To stick to the guidelines for better readability of the responses by the user, custom response messages are used. The response is made available to the client in XML format. Some of the URLs with the request response characteristics are put in table below:

Description	**Enter the requirements**
Access Control	Application Developer
HTTP Request	POST
HTTP URI	/topologies/subtopologies/storeRequirements
URL Params	none
Query Params	• value : none • criteria: none
Post Params	Requirements of the application.
Response	The requirements have been successfully uploaded. The requirement id is *x*
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.1: Description of REST API *Enter the requirements*.

Description	**Display the compatible services**
Access Control	Application Developer
HTTP Request	Get
HTTP URI	/topologies/subtopologies/getCompatibleServices
URL Params	none
Query Params	• value : requirementsID • criteria: none
Post Params	none
Response	Set of compatible services.
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.2: Description of REST API *Display the compatible services*.

Description	**Display the Pareto Optimal services**
Access Control	Application Developer
HTTP Request	Get
HTTP URI	/topologies/subtopologies/discoverOptimalServices
URL Params	none
Query Params	• value : requirementsID • criteria: none
Post Params	none
Response	Set of Optimal services.
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.3: Description of REST API *Display the Pareto Optimal services*.

Description	**Display the pareto optimal set of alpha topologies**
Access Control	Application Developer
HTTP Request	Get
HTTP URI	/topologies/subtopologies/discoverAlphaTopologies/
URL Params	none
Query Params	• value : requirementsID • criteria: none
Post Params	none
Response	Set of pareto optimal alpha topologies.
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.4: Description of REST API *Display the pareto optimal set of alpha topologies*.

Description	**Display the set of suitable middleware**
Access Control	Application Developer
HTTP Request	Get
HTTP URI	/topologies/subtopologies/reusable/middleware/displayMiddlewares
URL Params	none
Query Params	• value : requirementsID • criteria: none
Post Params	none
Response	Set of suitable middlewares.
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.5: Description of REST API *Display the set of suitable middleware*.

Description	**Display the application distribution alternatives**
Access Control	Application Developer
HTTP Request	Get
HTTP URI	/topologies/discoverDistributionAlternatives
URL Params	none
Query Params	• value : requirementsID • criteria: none
Post Params	none
Response	Set of application distribution alternatives.
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.6: Description of REST API *Display the application distribution alternatives*.

Description	**Persist the selected application distribution alternative**
Access Control	Application Developer
HTTP Request	Post
HTTP URI	/topologies/persistSelectedTopology
URL Params	Selected application distribution alternative
Query Params	• value : none • criteria: none
Post Params	none
Response	Set of application distribution alternatives.
Error Response	• 500 Internal Server Error. • 400 Bad Request - If the function is not recognized. • 400 Bad Request - If the input parameters provided are incorrect.

Table 5.7: Description of REST API *Persist the selected application distribution alternative*.

6 Implementation and Validation

In this chapter we describe the challenges and problems during the implementation phase to fulfill the requirements specified in Chapter 4 and the design presented in Chapter 5 and then discuss about validation of the system.

6.1 Implementation

The implementation of this work is mainly about building a framework which seamlessly interacts with the knowledge base, modelling framework, implementation of genetic algorithm for optimization and the cache mechanism for storing the intermediate data. The implementation of the genetic algorithm has been done by extending and rewriting parts of the jmetal framework. The modelling framework has been extended and the knowledge base is used to query on it and retrieve the cloud services. To maintain the stateless behaviour and to be able to use the format of request response, REST architectural style is chosen for designing the services.

6.1.1 RESTful Interface

To initiate the implementation, the XMLs have been developed for the Hypertext Transfer Protocol (HTTP) request messages and JAXB used for marshalling the java objects to XML and vice versa.

XML Schema

An XML schema for the request Discover Topology is shown in Listing 6.1

```
<complexType name="DiscoverTopologyType">
      <sequence>
        <element name="userData" type="tns:UserDataType"></
           element>
      </sequence>
    </complexType>

    <complexType name="Requirements"></complexType>


    <complexType name="Specification"><sequence>
```

```
        <element name="sds" type="tks:tRequirementRef"></element>
      </sequence></complexType>

    <complexType name="UserDataType">
      <sequence>
        <element name="Objectives" type="tns:ObjectivesType"></
           element>
      </sequence>
    </complexType>

    <complexType name="ObjectivesType">
      <sequence>
        <element name="Objective" type="string" maxOccurs="
           unbounded" minOccurs="1"></element>
      </sequence>
    </complexType>

    <element name="discoverTopology" type="
       tns:DiscoverTopologyType"></element>
```

Listing 6.1: XML Schema

Marshalling and Unmarshalling

We have used JAXB for marshalling and unmarshalling. Marshalling is the process of converting objects to XML and the conversion of XML to object is called as unmarshalling. This is used to ease and standardise the process and for this purpose we need to create Java class and annotate the class as root element for the XML and each variable as an XML element. After the class is created and elements are annotated as XML, we can marshal an instance of the class to an XML. The code for this can be seen in Listing 6.2

```
JAXBContext jxb = JAXBContext.newInstance(Resource.class);
Marshaller jxbMarshaller = jaxbContext.createMarshaller();
```

Listing 6.2: JAXB Marshalling

The code for unmarshalling an xml to an object of the annotated class can be seen in Listing 6.3

```
JAXBContext jxb = JAXBContext.newInstance(Resource.class);
Resource r = (Resource) jxb.createUnmarshaller().unmarshal(ht1.
   getContent());
```

Listing 6.3: JAXB Unmarshalling

6.1.2 Pricing Knowledge Base Interaction - Nefolog

This part of the implementation deals with building queries to fetch data from Nefolog using HTTP requests in an iterative way. First we need to fetch the requirements by analyzing the XML received from Winery. Then we query Nefolog with the requirements given by the application developer and when we get the data, we need to convert the HTTP content to string and analyze it to build the queries for the next step and this is to be repeated till we get the required services.

The analysis of the XML received from Winery is done using XPath query and the requirements are put in a map with the element type as key and the element as value. This can be seen in the Listing 6.4.

```
HashMap<String, ArrayList<String>> reqSet = new HashMap<String,
    ArrayList<String>>();
Requirement<String, ArrayList<String>> req1 = new Requirement<
    String, ArrayList<String>>();
InputSource xml = new InputSource("MediaWiki1.xml");
NodeList requirementNodes = (NodeList) XPathFactory.newInstance()
    .newXPath()
        .compile("//*[local-name()='Requirements']/*[local-name()
            ='Requirement']")
          .evaluate(xml, XPathConstants.NODESET);

for (int i = 0; i < requirementNodes.getLength(); i++) {
  System.out.println("Inside␣Loop" + i);
  String nodeName = requirementNodes.item(i).getChildNodes().item
      (1).getChildNodes().item(1)
            .getChildNodes().item(1).getNodeName();
  String textContent = requirementNodes.item(i).getChildNodes().
      item(1).getChildNodes().item(1)
            .getChildNodes().item(1).getTextContent();
                         ArrayList<String> list;
  if (reqSet.containsKey(nodeName)) {
    list = reqSet.get(nodeName);
    list.add(textContent);
  }
  else {
    list = new ArrayList<String>();
    list.add(textContent);
    reqSet.put(nodeName, list);
  }
}
Iterator<HashMap.Entry<String, ArrayList<String>>> entries =
    reqSet.entrySet().iterator();
```

```
while (entries.hasNext()) {
  HashMap.Entry<String, ArrayList<String>> entry = entries.next()
     ;                        req1.getRequireMap().put(entry.getKey
     (), entry.getValue());
}
```

Listing 6.4: Analyze XML

After the map is created, it is sent to the server as a part of an HTTP request which is stored in the server and the server returns an id to the client which can be used for accessing the specific requirements for further steps. We will be using a REST client to send this map to our server which will be shown in details in the validation section.

When we have the requirements map in the server, we get the key and values which are used for building queries for Nefolog. The building of queries for Nefolog can be seen in the Listing 6.5. Once the query is built, an HTTP request is done which can be seen in the Listing 6.6 and after we get the response, it is converted to a String and returned for further analysis. The string we receive as a result of querying on Nefolog and converting the HTTP response is then analyzed using XPath to find the data required to build the URL for the next query to Nefolog.

```
Iterator<HashMap.Entry<String, ArrayList<String>>> entries =
    requirements.entrySet().iterator();
String value = "";
String key = "";
HashMap.Entry<String, ArrayList<String>> entry = entries.next();
value = entry.getValue().get(0);
key = entry.getKey();
String nefologHost = prop.getProperty("nefologHost");
String url = nefologHost + "/" + key + "/" + value;
String response = httpRequest(url);
```

Listing 6.5: Query Nefolog

```
CloseableHttpClient httpclient = HttpClientBuilder.create()
        .setSSLHostnameVerifier(new NoopHostnameVerifier()).build
           ();
HttpGet get = new HttpGet(url);
CloseableHttpResponse response1 = httpclient.execute(get);
HttpEntity ht1 = response1.getEntity();
BufferedHttpEntity buf1 = new BufferedHttpEntity(ht1);
String responseContent1 = EntityUtils.toString(buf1, "UTF-8");
return responseContent1;
```

Listing 6.6: HTTP Request

6.1.3 Cache Mechanism

The cache mechanism, we are using to store the intermediate data is Ehcache which is an open source standards based cache which is widely used in Java projects. For using, it first we have to write a class to instantiate the cache by creating the Cache manager and the cache and return an instance of the cache. The class should also have the methods to put and get resources. The implementation of it can be seen in the Listing 6.7

```
private static GlobalCache instance = null;
private static CacheManager cacheManager;
private static Cache<Integer, Resource> serviceCache;

public synchronized static GlobalCache getInstance() {
  cacheManager = CacheManagerBuilder.newCacheManagerBuilder().
     with(new               CacheManagerPersistenceConfiguration
     (new File("resources/","myData"))).
     withCache("persistent",
     CacheConfigurationBuilder.newCacheConfigurationBuilder()
     .withResourcePools(
     ResourcePoolsBuilder.newResourcePoolsBuilder().heap(10L,
        EntryUnit.ENTRIES).disk(10L,
     MemoryUnit.MB, true))
     .buildConfig(Integer.class, Resource.class))
     .build(true);

  serviceCache = cacheManager.createCache("serviceCache",
                        CacheConfigurationBuilder.
     newCacheConfigurationBuilder().
    buildConfig(Integer.class, Resource.class));
  return instance;
}
public void put(Resource res) {
  serviceCache.put(res.getConfigId(), res);
}

public List<Resource> getAllResources() {

  List<Resource> resources = new ArrayList<Resource>();
  Iterator iterator = serviceCache.iterator();
  while (iterator.hasNext()) {
    Cache.Entry<Integer, Resource> c = (org.ehcache.Cache.Entry<
       Integer, Resource>)           iterator.next();
    resources.add(c.getValue());
  }
  return resources;
}
```

Listing 6.7: Cache Management

When we have the cache ready and we get the required resources from Nefolog as HTTP response, then response then needs to be converted to an object and then put into the cache. This has been done as shown in Listing 6.8

```
GlobalCache cache = GlobalCache.getInstance();
JAXBContext jxb = JAXBContext.newInstance(Resource.class);
Resource r = (Resource) jxb.createUnmarshaller().unmarshal(ht1.
   getContent());
cache.put(r);
```

Listing 6.8: Put Resources in Cache

6.1.4 Optimization Algorithm

The genetic algorithm used by us for finding out the optimal services is NSGAII and we use the jmetal framework for implementing the same.

JMetal Framework

JMetal[DN11] stands for Metaheuristic Algorithms in Java and is an object-oriented Java-based framework for multi-objective optimization with meta-heuristics. It has the implementations included for solving most of the common numerical optimization problems. The features and object-oriented architecture of the framework allows to experiment with the provided classic and state-of-the-art techniques, develop own algorithms, solve optimization problems and integrate jmetal in other tools. [ND14] We have used jMetal 4.5 and would like to explain a bit about its architecture using the class diagram. The class diagram can be seen in the Figure 6.1. The classes that we have extended are outlined in grey and the classes created by us are presented as grey boxes. As per [ND14], the basic architecture of jmetal relies in that an Algorithm solves a Problem using one (and possibly more) SolutionSet and a set of Operator objects.A generic terminology has been used to name the classes in order to make them general enough to be used in any meta-heuristic. In the context of evolutionary algorithms, populations and individuals correspond to SolutionSet and Solution jMetal objects, respectively.The main decisions to be taken while using the framework is to design the Solution and the problem. [ND14]

Solution Design The representation of the solution strongly depends on the problem and selecting a specific representation has a great impact on the behavior of meta-heuristics and, hence, in the obtained. results. Figure 6.2 depicts the basic components that are used for

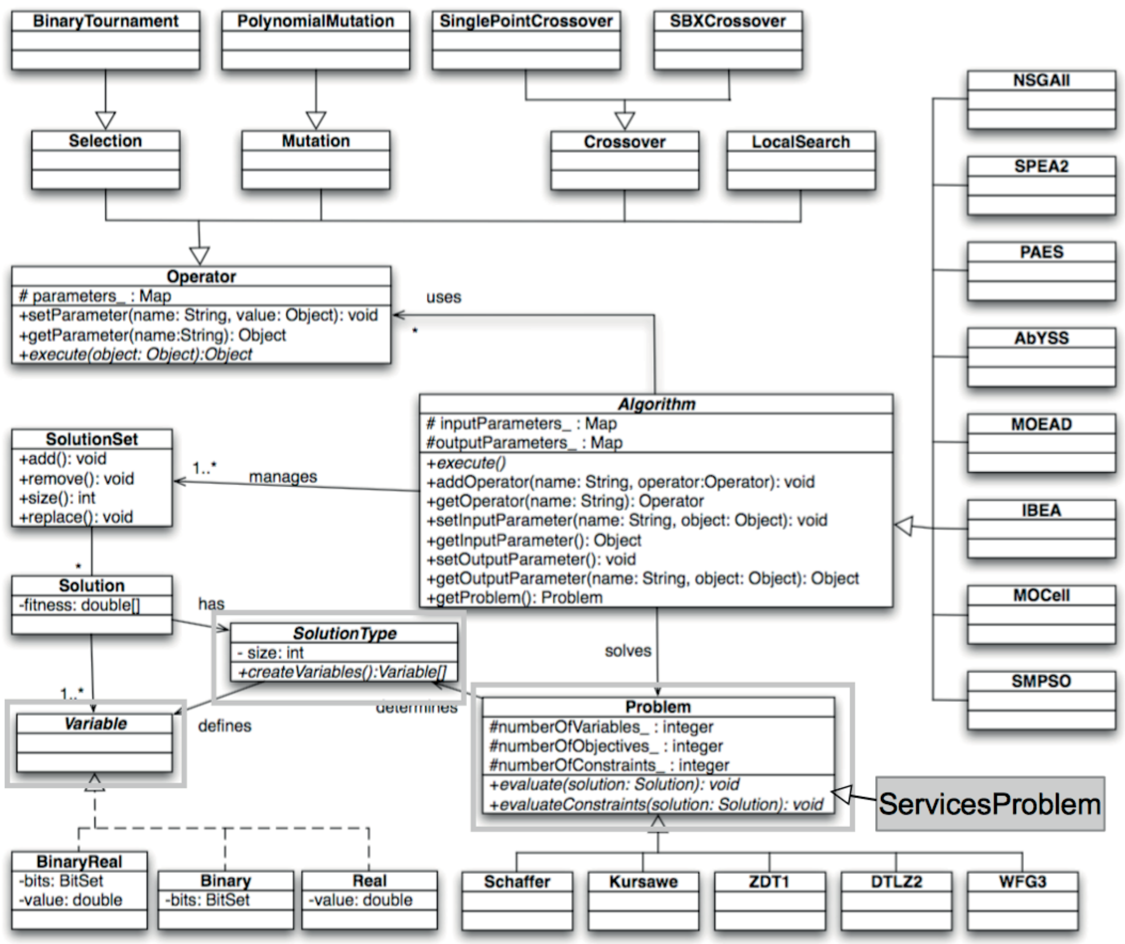

BinaryTournament
PolynomialMutation
SinglePointCrossover
SBXCrossover
Selection
Mutation
Crossover
LocalSearch
Operator
# parameters_ : Map
+setParameter(name: String, value: Object): void
+getParameter(name:String): Object
+execute(object: Object):Object
uses
NSGAII
SPEA2
PAES
AbYSS
MOEAD
IBEA
MOCell
SMPSO
Algorithm
# inputParameters_ : Map
#outputParameters_ : Map
+execute()
+addOperator(name: String, operator:Operator): void
+getOperator(name: String): Operator
+setInputParameter(name: String, object: Object): void
+getInputParameter(): Object
+setOutputParameter(): void
+getOutputParameter(name: String, object: Object): Object
+getProblem(): Problem
SolutionSet
+add(): void
+remove(): void
+size(): int
+replace(): void
1..*
manages
Solution
-fitness: double[]
has
SolutionType
- size: int
+createVariables():Variable[]
solves
determines
1..*
Variable
defines
Problem
#numberOfVariables_ : integer
#numberOfObjectives_ : integer
#numberOfConstraints_ : integer
+evaluate(solution: Solution): void
+evaluateConstraints(solution: Solution): void
ServicesProblem
BinaryReal
-bits: BitSet
-value: double
Binary
-bits: BitSet
Real
-value: double
Schaffer
Kursawe
ZDT1
DTLZ2
WFG3

Figure 6.1: Class Diagram for jMetal4.5 [ND14]

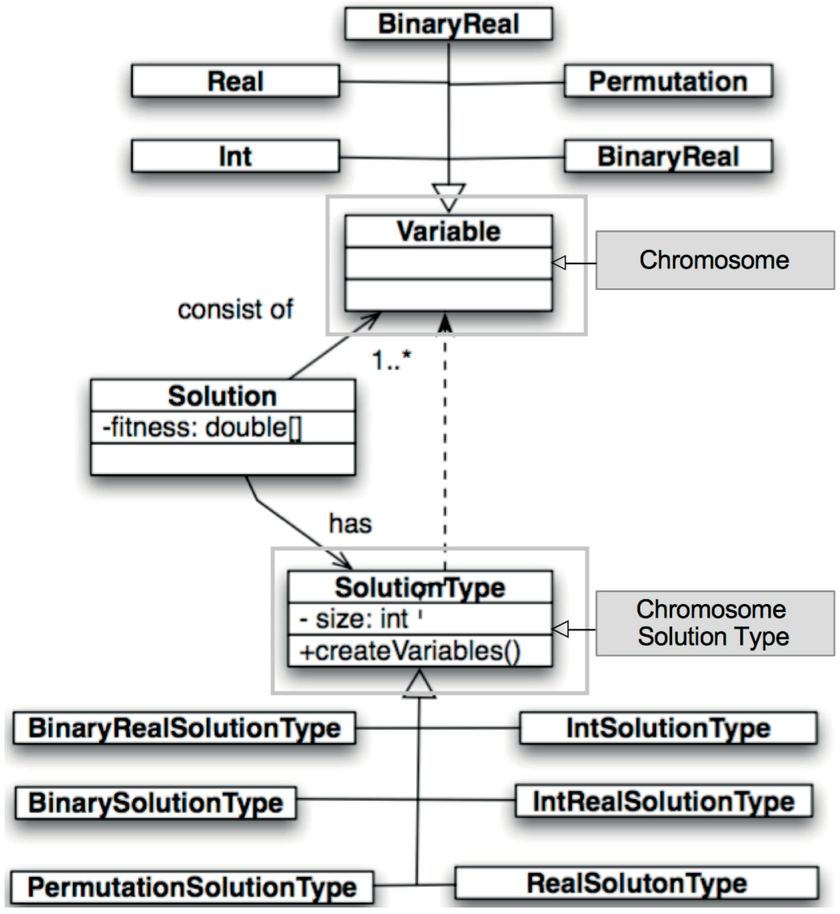


Figure 6.2: Solution Representation in jMetal4.5 [ND14]

representing solutions into the framework. Solution basically comprises of set of Variable objects and an array to store the fitness values. Each Solution is associated with a solution type which allows to define the variable types of the Solution and creating them. [ND14]

Problem Definition In jMetal, all the problems inherits from class Problem. This class contains two basic methods: evaluate() and evaluateConstraints(). Both methods receive a Solution representing a candidate solution to the problem; the first one evaluates it, and the second one determines the overall constraint violation of this solution. All the problems have to define the evaluate() method, while only problems having side constraints need to define evaluateConstraints(). A key design feature in jmetal is that the problem defines the allowed solutions types that are suitable to solve it. [ND14]

Extensions and Adaptations in JMetal Framework

As we know, the jmetal framework has implementations for solving only numerical problems so we need to make some modifications to use it for our purpose. Mainly, we need to create the problem and solution for our purpose. For using the jmetal framework, we have made necessary changes in it and exported it as a jar and then we have imported it in our main project and are doing further required modifications mainly in the algorithm i.e. NSGAII class. We also have written a helper class in our project to help us run the algorithm and copied the implementation of the algorithm in our project to be able to pass the instance of the cache.

The helper class has been created within our main project and has been extended from the NSGAII_main.java in the jMetal framework[DN11]. Listing 6.9 shows the helper class created by us

```
public class NSGA2Helper {

public static void execute(GlobalCache cache) throws
   ClassNotFoundException, InstantiationException,
   IllegalAccessException, MalformedURLException, SAXException,
   IOException {
  try{
    Problem   problem   ; // The problem to solve
    Algorithm algorithm ; // The algorithm to use
    Operator  crossover ; // Crossover operator
    Operator  mutation  ; // Mutation operator
    Operator  selection ; // Selection operator
    HashMap  parameters ; // Operator parameters
    QualityIndicator indicators ; // Object to get quality
       indicators
    Logger logger_      = Configuration.logger_ ;
    FileHandler fileHandler_ = new FileHandler("NSGAII_main.log")
       ;
    logger_.addHandler(fileHandler_) ;
    indicators = null ;
    problem = new genetic.ServicesProblem("Chromosome");
    algorithm = new genetic.NSGAII(problem,cache);
    // Algorithm parameters
    algorithm.setInputParameter("populationSize",4);
    algorithm.setInputParameter("maxEvaluations",100);
    // Mutation and Crossover for Real codification
    parameters = new HashMap() ;
    parameters.put("probability", 0.9) ;
    parameters.put("distributionIndex", 20.0) ;
    crossover = CrossoverFactory.getCrossoverOperator("
       SBXCrossover", parameters);
    parameters = new HashMap() ;
    parameters.put("probability", 1.0/problem.
       getNumberOfVariables()) ;
    parameters.put("distributionIndex", 20.0) ;
    mutation = MutationFactory.getMutationOperator("PolynomialMutation", parameters);                    // Selection Operator
    parameters = null ;
    selection = SelectionFactory.getSelectionOperator("BinaryTournament2", parameters) ;
    // Add the operators to the algorithm
    algorithm.addOperator("crossover",crossover);
    algorithm.addOperator("mutation",mutation);
    algorithm.addOperator("selection",selection);
    // Add the indicator object to the algorithm
    algorithm.setInputParameter("indicators", indicators) ;
    // Execute the Algorithm
    long initTime = System.currentTimeMillis();
    SolutionSet population1 = algorithm.execute();
    SolutionSet population = algorithm.execute(cache);
    long estimatedTime = System.currentTimeMillis() - initTime;
    // Result messages
    logger_.info("Total␣execution␣time:␣"+estimatedTime + "ms");
    logger_.info("Variables␣values␣have␣been␣writen␣to␣file␣VAR");
    population.printVariablesToFile("VAR");
    logger_.info("Objectives␣values␣have␣been␣writen␣to␣file␣FUN");
    population.printObjectivesToFile("FUN");
  }
  catch(JMException e){e.printStackTrace();}
}
}
```

Listing 6.9: Put Resources in Cache

Creating the Problem We have created our problem called ServicesProblem as per the given guidelines in the framework. The problem created by us can be seen in Listing 6.10. As we can observe, it extends class Problem and a constructor method is defined for creating instances of this problem, which has two parameters: a string containing a solution type identifier and the number of decision variables of the problem. As a general rule, all the problems should have as first parameter the string indicating the solution type. After the constructor, the evaluate() method is redefined; in this method, after computing the two objective function values, they are stored into the solution by using the setObjective method of Solution.

```
public class ServicesProblem extends Problem {

  public ServicesProblem(String solutionType, GlobalCache cache)
```

```
 {
    numberOfVariables_   = 4 ;
    numberOfObjectives_  = 2 ;
    numberOfConstraints_ = 2 ;
    problemName_         = "Services";
    serviceTypes_ = new HashSet<String>();
    instanceTypes_ = new HashSet<String>();
    operatingSystems_ = new HashSet<String>();
    chrList_ = new ArrayList<Chromosome>();
    instanceNumbers_ = new int[] {1};
    List<Resource> allResource = cache.getAllResources();
    Iterator<Resource> iterator = allResource.iterator();
    while(iterator.hasNext())
    {
      Chromosome chr = new Chromosome();
      Resource temp = iterator.next();
      serviceTypes_.add(temp.getProvider());
      instanceTypes_.add(temp.getName());
      chr.setServiceType(temp.getProvider());
      chr.setInstanceType(temp.getName());
      chr.setInstanceNumber("1");
      if (temp.getContent() !=null)
      {
        if(temp.getContent().getPerformance()!=null)
        {
          String temp1 = temp.getContent().getPerformance()[0].
             getName();
          if(temp1.equals("os"))
          {
            operatingSystems_.add((String)(temp.getContent().
               getPerformance()[0].getValue()));
            chr.setOS((temp.getContent().getPerformance()[0].
               getValue()));
          }
        }
      }
      chrList_.add(chr);
    }
    if (solutionType.compareTo("Chromosome") == 0)
      solutionType_ = new ChromosomeSolutionType(this) ;
    else {
      System.out.println("Error:␣solution␣type␣" + solutionType +
          "␣invalid") ;
      System.exit(-1) ;
    }
  }
  public void evaluate(Solution solution) throws JMException {}
  public void evaluate(Solution solution, GlobalCache cache) throws JMException {
    String [] x = new String[4] ; // 4 decision variables
    double [] fx = new double[2] ;  // 2 functions
    String temp1 = new String();
    boolean bool= false;
    x = getRandom(solution,cache);
    for (int i = 0; i < (tempRes.getContent().getPerformance()).length; i++) {
      System.out.println("hahaha␣"+tempRes.getProvider()+"---"+tempRes.getName());
      System.out.println("Looking␣for␣SLA");
      System.out.println(tempRes.getContent().getPerformance()[i].getName());
      if(tempRes.getContent().getPerformance()[i].getName().trim().equalsIgnoreCase("sla")){
        temp1=(tempRes.getContent().getPerformance()[i].getValue());
        bool=true;
        break;
      }
    }
    if (bool)
      fx[0] = Double.parseDouble(temp1);
    else
      fx[0] = 0.99; //Default
    try {
      fx[1] = getCost(cache);
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println("SLA␣is␣---"+fx[0]+"Cost␣is␣---"+fx[1]);

  }

    solution.setOverallConstraintViolation(total);
    solution.setNumberOfViolatedConstraint(number);
  }

  public String[] getRandom(Solution solution, GlobalCache cache) throws JMException
  {
    counter=counter+1;
    String [] x = new String[4] ;
    double [] fx = new double[2] ;

    for (int i = 0; i < x.length; i++) {
      x[i] = solution.getDecisionVariables()[i].getValues();
    }
    Chromosome chr = new Chromosome();
    chr.setServiceType(x[0]);
    chr.setInstanceType(x[1]);
    chr.setOS(x[2]);
    chr.setInstanceNumber(x[3]);
    List<Resource> allResource = cache.getAllResources();
    Iterator<Resource> iterator = allResource.iterator();
    int k= 0;
    Resource temp = new Resource();
    while(iterator.hasNext())
    {
      temp = iterator.next();
      if ((x[0]==temp.getProvider())&&(x[1]==temp.getName())&&(
         temp.getContent() !=null)
          &&(temp.getContent().getPerformance()!=null)&&(x[2]==
             temp.getContent().getPerformance()[0].getValue()))
      {
        System.out.println("Inside␣ifs"+x[0]+"␣---␣"+x[1]+"␣---␣"
           +x[2]+"␣---␣"+x[3] );
        k=k+1;
        tempRes=temp;
      }
    }
    if(k==0)
    {
      x =getRandom(solution, cache);

    }
    return x;
  }

  public double getCost(GlobalCache cache) throws
     ClientProtocolException, IOException,
     XPathExpressionException
  {
    int cId= tempRes.getConfigId();
    int Hour= tempRes.getHour();
    int month= tempRes.getMonth();;
    int io= tempRes.getIO();;
    int storage= tempRes.getStorage();;
    int ExtNetEgress= tempRes.getExtNet();;
    String loc_zone= tempRes.getLocZone();;
    double cost= 0;
    costCalculator = prop.getProperty("costCalculator");
    String url = costCalculator + "configid=" + cId + "&Hour=" +
       Hour+ "&month=" +month+
      "&i/oOperation=" + io + "&GBStorage=" + storage + "&
         GBExternalNetworkEgress=" +
      ExtNetEgress + "&location_zone=" + loc_zone;
    CloseableHttpClient httpclient = HttpClientBuilder.create()
      .setSSLHostnameVerifier(new NoopHostnameVerifier()).build
         ();
    HttpGet get = new HttpGet(url);
    CloseableHttpResponse response1 = httpclient.execute(get);
    HttpEntity ht1 = response1.getEntity();
    BufferedHttpEntity buf1 = new BufferedHttpEntity(ht1);
    String response = EntityUtils.toString(buf1, "UTF-8");
    XPath xPath = XPathFactory.newInstance().newXPath();
    String path1 = "/resource/content/result/cost";
    InputSource i1 = new InputSource(new StringReader(response));
    NodeList uriNode = (NodeList) xPath.compile(path1).evaluate(
       i1, XPathConstants.NODESET);
    if((uriNode).item(0)!=null){
      cost = Double.parseDouble(uriNode.item(0).getTextContent().
         substring(1, 8));
    }
    else
      cost = 50000; //Default
    return cost;
  }

}
```

Listing 6.10: Problem class for our project.

Designing the Solution In the Listing 6.10 , we can see that we don't use any of the generic solution types given by jmetal like integer,binary,real,etc. Rather we use a solution type called ChromosomeSolutionType which we have defined in the package jmetal.encodings.SolutionType which can be seen in the Listing 6.11

```
public class ChromosomeSolutionType extends SolutionType {
  public ChromosomeSolutionType(Problem problem) {
    super(problem) ;
  } // Constructor

  public Variable[] createVariables() {
    try{
      Variable[] variables = new Variable[4];
      variables[0] = new ServiceTypes(problem_.getServiceTypes())
         ;
      variables[1] = new InstanceTypes(problem_.getInstanceTypes
         ());
      variables[2] = new OperatingSystems(problem_.
         getOperatingSystems());
      variables[3] = new NumberOfInstances(problem_.
         getInstanceNumbers());
      return variables ;
    }
    catch(Exception E){return null ;}
  }
}
```

Listing 6.11: Defining the Chromosome Solution Type.

In the Listing 6.11, we can see the solution type consists variables of four types called ServiceTypes, InstanceTypes,OperatingSystems and NumberOfInstances so we had to define all these types in the package jmetal.encodings.variable. As an example, the definition of ServiceTypes has been shown in the Listing 6.12. The other types have also been defined in the same way.

```
public class ServiceTypes extends Variable{
  private Set<String> serviceTypes_;
    private String value_;
  public ServiceTypes() {
  }
  public ServiceTypes(Set<String> st){
    serviceTypes_ = st;
  }
  public Set<String> getServiceTypes_() {
    return serviceTypes_;
  }
  public void setServiceTypes_(Set<String> serviceTypes_) {
    this.serviceTypes_ = serviceTypes_;
  }
  public ServiceTypes(Variable variable) throws JMException{
    serviceTypes_ = variable.getServiceTypes();
  }
  @Override
  public Variable deepCopy() {
    try {
      return new ServiceTypes(this);
    } catch (JMException e) {
      Configuration.logger_.severe("Chromosome.deepCopy.execute:␣
         JMException");
      return null ;
    }
  }
  public String getValues() {
    Random generator = new Random();
    int randomIndex = generator.nextInt(serviceTypes_.size());
    String[] setArray = (String[]) serviceTypes_.toArray(new
        String[serviceTypes_.size()]);
    value_=setArray[randomIndex];
    return value_;
    }
}
```

Listing 6.12: Service Types as an example of the added Variables.

We have also defined a class called Chromosome which defines the decision encoding required to solve our problem and can be seen in the Listing 6.13. In order to make the copy constructor work, we had to include the serviceType, instanceType, operatingSystem and numberOfInstances in the variable class.

```
public class Chromosome extends Variable{
  private String serviceType_;
  private String instanceType_;
  private String os_;
  private int instanceNumber_;
  public Chromosome() {
  }
  public Chromosome(String st, String it, String os, int ni){
    serviceType_ = st;
    instanceType_ = it;
    os_ = os ;
    instanceNumber_ = ni;
  }
  public Chromosome(Variable variable) throws JMException{
    serviceType_ = variable.getServiceType();
    instanceType_ = variable.getInstanceType();
    os_ = variable.getOperatingSystem();
    instanceNumber_ = variable.getNumberOfInstance();
  }
  public String getServiceType() {
    return serviceType_;
  }
  public void setServiceType(String st) {
    serviceType_ = st;
  }
  public String getInstanceType() {
    return instanceType_;
  }
  public void setInstanceType(String it) {
    instanceType_ = it;
  }
  public String getOS() {
    return os_;
  }
  public void setOS(String os) {
    os_ = os;
  }
  public String getInstanceNumber() {
    return instanceNumber_;
  }
  public void setInstanceNumber(String ni) {
    instanceNumber_ = ni;
  }
  public Variable deepCopy(){
    try {
      return new Chromosome(this);
    } catch (JMException e) {
      Configuration.logger_.severe("Chromosome.deepCopy.execute:␣
         JMException");
      return null ;
    }
  }
}
```

Listing 6.13: Definition of encoding Variable Chromosome.

6.2 Validation

In this section, we will be validating the prototype of our system for the Mediawiki application which runs wiki-based projects. We will be validating the scenarios for storing the requirements given by the user, finding out the compatible services and using the genetic algorithm to get the set of optimal services. We will be using the Postman[1] API Client for the validation purposes.

6.2.1 Store Requirements

The system stores the requirements given by the client and returns an *Requirement ID* which can be used by the developer later in order to fetch services as per the requirements. Here a Post request is done on the URL */topologies/subtopologies/storeRequirements*. The HTTP Post request for this operation can be seen in the Listing 6.14. The request and response from Postman for this can be seen in the figure 6.3.

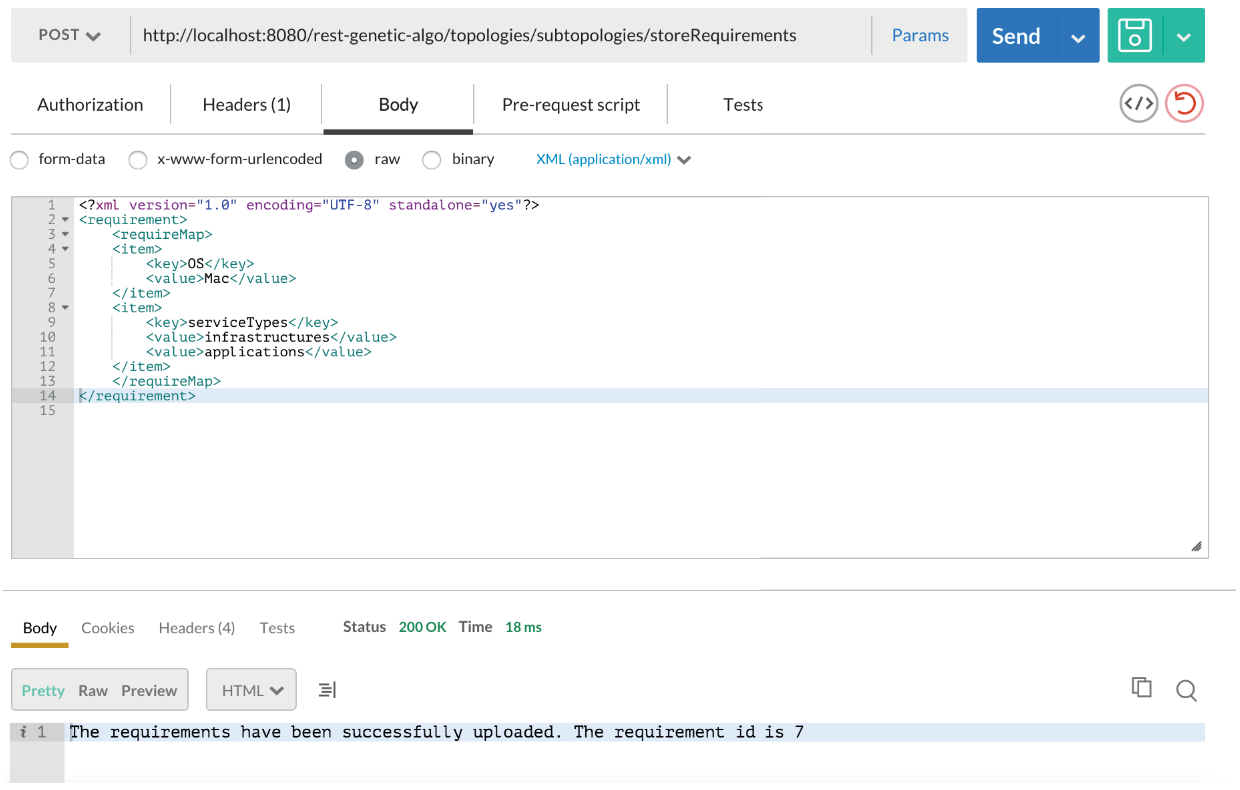
```
POST /rest-genetic-algo/topologies/subtopologies/
    storeRequirements
HTTP/1.1
Host: localhost:8080
Content-Type: application/xml
Cache-Control: no-cache
Postman-Token: d24142c0-6858-f82d-0e0a-0893e507555d

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><
    requirement><requireMap><item><key>OS</key><value>Mac</value
    ></item><item><key>serviceTypes</key><value>infrastructures</
    value><value>applications</value></item></requireMap></
    requirement>
```

Listing 6.14: Post Request for Store Requirements

6.2.2 Get Compatible Services

Once the requirements are stored by the system, the application developer can pass the requirement id for the further operations. In this case a Get request is done using the URL *http://localhost:8080/rest-genetic-algo/topologies/subtopologies/getCompatibleServices?requirementsID=7*. When the application developer hits this URL, the server fetches the requirements saved for the requirement id 7 and fetches all the services feasible for the given requirements from Nefolog and returns it to the client. The HTTP Get request for this operation can be seen in

[1]http://www.getpostman.com/



```
http://localhost:8080/rest-genetic-algo/topologies/subtopologies/storeRequirements
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<requirement>
    <requireMap>
    <item>
        <key>OS</key>
        <value>Mac</value>
    </item>
    <item>
        <key>serviceTypes</key>
        <value>infrastructures</value>
        <value>applications</value>
    </item>
    </requireMap>
</requirement>
```

Status 200 OK Time 18 ms

```
The requirements have been successfully uploaded. The requirement id is 7
```

Figure 6.3: Request and Response for Store Requirements using Postman API Client

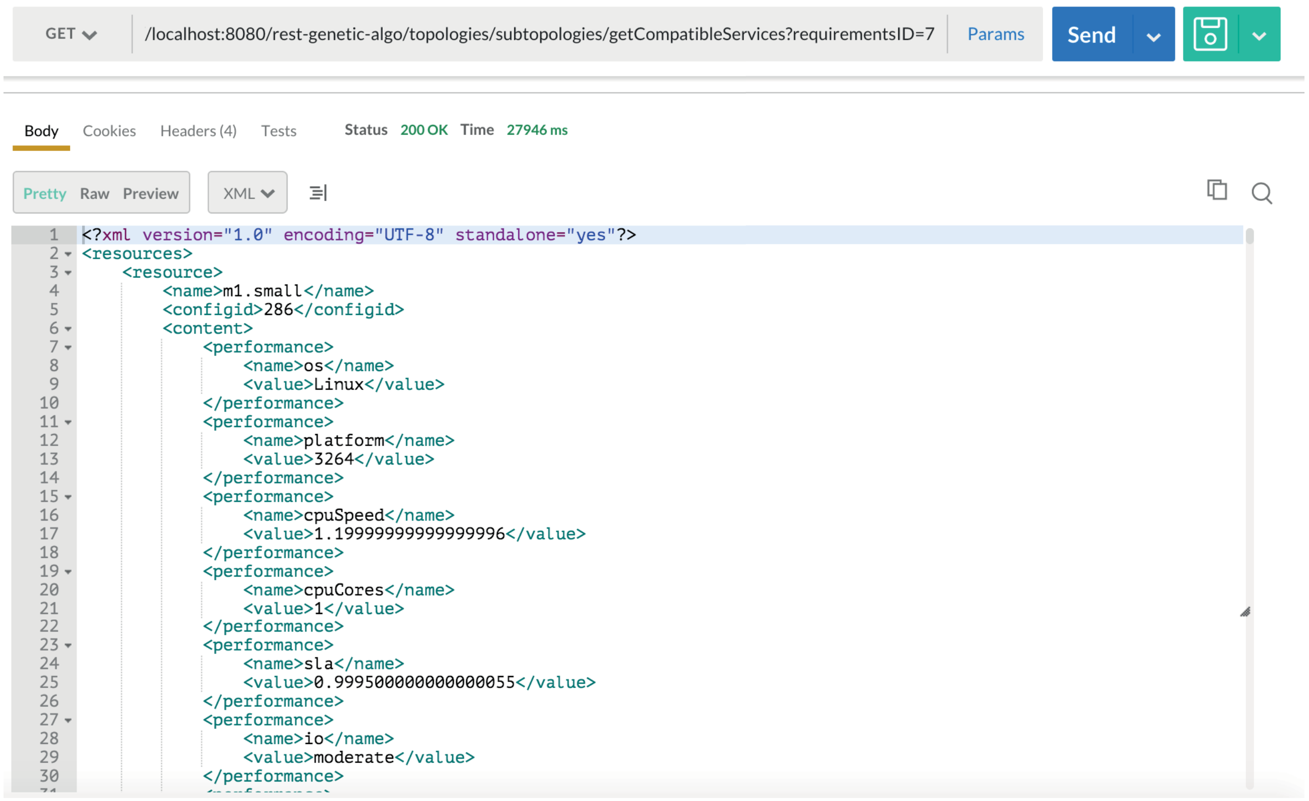
the Listing 6.15. The request and response from Postman for this can be seen in the figure 6.4.

```
GET /rest-genetic-algo/topologies/subtopologies/
    getCompatibleServices?requirementsID=7 HTTP/1.1
Host: localhost:8080
Content-Type: application/xml
Cache-Control: no-cache
Postman-Token: ee9ac8a8-69d4-a84c-35c5-26f33d35d57d
```

Listing 6.15: Get Request for Compatible Services

6.2.3 Discover Optimal Services

A HTTP Get request is done in order to get all theusing the URL *http://localhost:8080/rest-genetic-algo/topologies/subtopologies/discoverOptimalServices?requirementsID=7*. When the application developer hits this URL, the server fetches the requirements saved for the requirement id 7 and fetches all the services feasible for the given requirements from Nefolog and returns it to the client. The HTTP Get request for this operation can be seen in the Listing 6.16. The request and response from Postman for this can be seen in the figure 6.5.



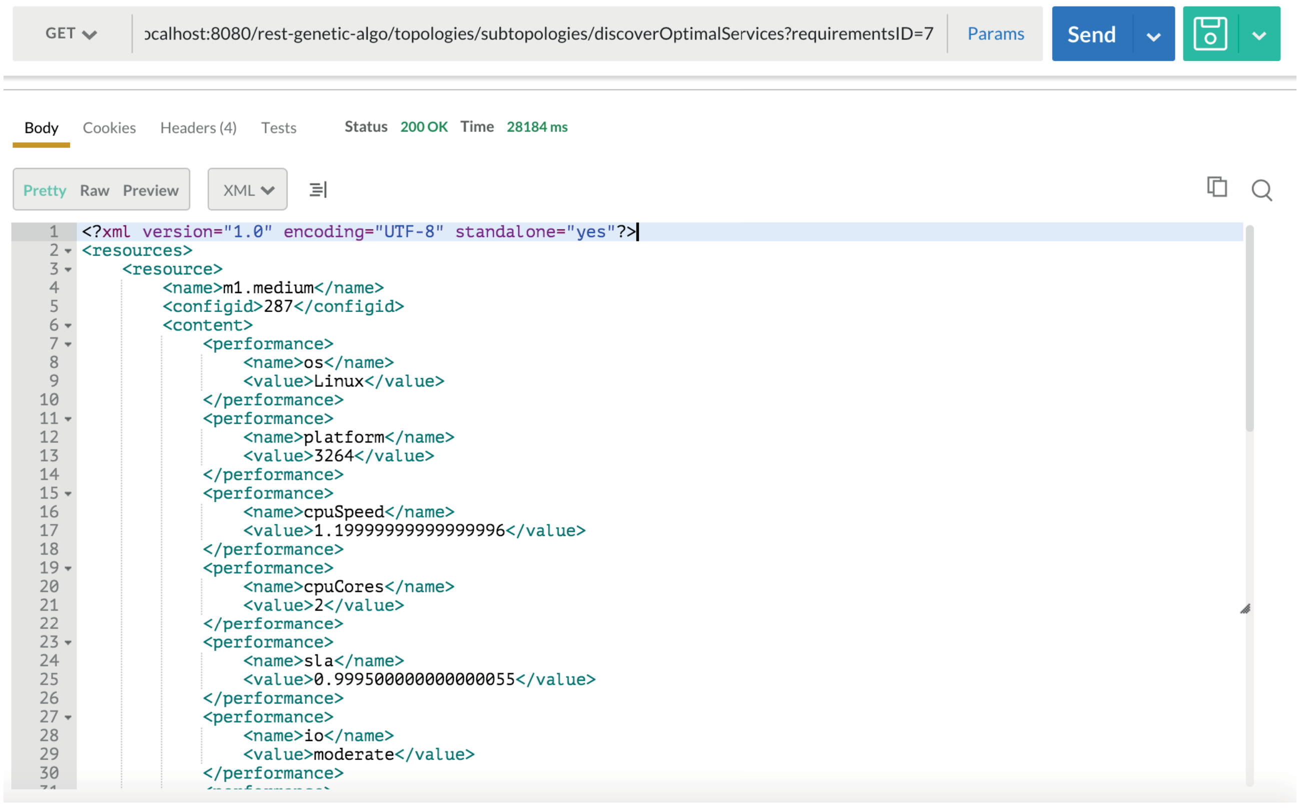
```
GET /localhost:8080/rest-genetic-algo/topologies/subtopologies/getCompatibleServices?requirementsID=7
Status 200 OK  Time 27946 ms
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resources>
    <resource>
        <name>m1.small</name>
        <configid>286</configid>
        <content>
            <performance>
                <name>os</name>
                <value>Linux</value>
            </performance>
            <performance>
                <name>platform</name>
                <value>3264</value>
            </performance>
            <performance>
                <name>cpuSpeed</name>
                <value>1.19999999999999996</value>
            </performance>
            <performance>
                <name>cpuCores</name>
                <value>1</value>
            </performance>
            <performance>
                <name>sla</name>
                <value>0.99950000000000055</value>
            </performance>
            <performance>
                <name>io</name>
                <value>moderate</value>
            </performance>
```

Figure 6.4: Request and Response for Get Compatible Services using Postman API Client

```
GET /rest-genetic-algo/topologies/subtopologies/
    discoverOptimalServices?requirementsID=7 HTTP/1.1
Host: localhost:8080
Content-Type: application/xml
Cache-Control: no-cache
Postman-Token: 0c680200-df6a-fc18-1fc6-106f7270d0a5
```

Listing 6.16: Get Request for Optimal Services



```
ɔcalhost:8080/rest-genetic-algo/topologies/subtopologies/discoverOptimalServices?requirementsID=7
```

Status 200 OK Time 28184 ms

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resources>
    <resource>
        <name>m1.medium</name>
        <configid>287</configid>
        <content>
            <performance>
                <name>os</name>
                <value>Linux</value>
            </performance>
            <performance>
                <name>platform</name>
                <value>3264</value>
            </performance>
            <performance>
                <name>cpuSpeed</name>
                <value>1.1999999999999996</value>
            </performance>
            <performance>
                <name>cpuCores</name>
                <value>2</value>
            </performance>
            <performance>
                <name>sla</name>
                <value>0.99950000000000055</value>
            </performance>
            <performance>
                <name>io</name>
                <value>moderate</value>
            </performance>
```

Figure 6.5: Request and Response for Discover Optimal Services using Postman API Client

7 Outcome and Future Work

Deployment of applications in the cloud environment is done to reduce the cost required for the building and maintenance of IT infrastructure as well as reap the benefits of the cloud infrastructure. The components of an application can be distributed partially or fully on cloud services which may belong to the same or different vendors. Application topologies are built for this which also provide an opportunity to deploy parts of application on physical servers. However there exists lack of support for building application topologies taking into consideration the possibility of deployment on physical servers and finding the optimal deployment options taking into consideration multiple objectives. This thesis focuses to provide the support for building topologies for deploying applications while optimized services or deployment options to be used in the topology for each tier is selected basing on the requirements and objectives provided by the application developer.

We begin the work by listing down the relevant research challenges within the motivation for this work, list of abbreviations and outlines of this thesis which are presented in the first chapter. In Chapter 2, we discuss about the basic concepts of topics and technologies which are relevant to this thesis. Here, we give an introduction about cloud computing, cloud application topology and different topology specification languages such as TOSCA and GENTL, RESTful services and REST API design. We also talk about the various optimization algorithms we have taken into consideration and other techniques like case based reasoning and feature diagrams. Chapter 3 deals with other works which are closely related to this masters thesis like [FFH13], [SBB^{+}15], [MR12] and [OGW^{+}14]. In this chapter, we also discuss about how this works lack in fulfilling the objective of this thesis.

After discussing all the fundamental and related works, we lay down the concept in Chapter 4. We begin this chapter by putting forward a comparison among the discussed optimization algorithms and justifying why we have selected the one we are using. Then we give an overview about the steps to be followed to complete the objective of discovering cloud services and topologies for an application. We go on to describe how we would implement the genetic algorithm with a detailed step by step explanation using an example. The functional and non functional requirements of the system and the use cases are also detailed in this chapter at the end of which we have an overview of the whole system.

Once the concepts are ready, we go on to design the system, which is explained in chapter 5. It presents the architectural solution taken into account to build the system which fulfills the requirements specified in Chapter 4. We discuss the architecture of the whole system and then go on to design the resource models and RESful APIs for the system. The implementation and validation of the system as per the concepts and design of chapter 4 and 5 is shown in chapter 6. Here we show, how we have used and extended the existing tools and frameworks for the implementation of our system.

Further future works involve the possibility to include statistical analysis for verification of the provided optimized topologies which would require the use of utility functions to validate the same. Similarity analysis can also be used to compare two requirements and return the optimal topology used earlier if the requirements are similar. There is also a possibility to use graph based databases in order to accelerate the process of executing the algorithms and searching for topologies.

Bibliography

[All08] J. Allspaw. *The Art of Capacity Planning: Scaling Web Resources*. " O'Reilly Media, Inc.", 2008.

[ARSL14] V. Andrikopoulos, A. Reuter, S. G. Sáez, and F. Leymann. A GENTL Approach for Cloud Application Topologies. In *Service-Oriented and Cloud Computing*, pages 148–159. Springer, 2014.

[ASLW14] V. Andrikopoulos, S. G. Sáez, F. Leymann, and J. Wettinger. Optimal distribution of applications in the cloud. In *Advanced Information Systems Engineering*, pages 75–90. Springer, 2014.

[Bat05] D. Batory. *Feature models, grammars, and propositional formulas*. Springer, 2005.

[BBH$^+$13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner. OpenTOSCA–a runtime for TOSCA-based cloud applications. In *Service-Oriented Computing*, pages 692–695. Springer, 2013.

[BBLS12] T. Binz, G. Breiter, F. Leyman, and T. Spatzier. Portable cloud services using tosca. *IEEE Internet Computing*, (3):80–85, 2012.

[BFH03] F. Berman, G. Fox, and A. J. Hey. *Grid computing: making the global infrastructure a reality*, volume 2. John Wiley and sons, 2003.

[BGPCV11] L. Badger, T. Grance, R. Patt-Corner, and J. Voas. Draft cloud computing synopsis and recommendations. *NIST special publication*, 800:146, 2011.

[Bis11] J. Bishop. Cloud or Not: 4 Cloud Deployment Models, 2011.

[Bro11] J. Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[BS12] S. Binitha and S. S. Sathya. A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.

[BVB08] J. Broberg, S. Venugopal, and R. Buyya. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[dev10] dev2ops. What Is DevOps, 2010.

[DN11] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[FFH13] S. Frey, F. Fittkau, and W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 512–521. IEEE Press, 2013.

[Fie00] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[Gan] K. Ganguly. Performance Aware Cloud Application Topology Enrichment. Master's thesis, University of Stuttgart.

[GGQ$^+$13] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8):1230–1242, 2013.

[HCDWV05] R. Hassan, B. Cohanim, O. De Weck, and G. Venter. A comparison of particle swarm optimization and the genetic algorithm. In *Proceedings of the 1st AIAA multidisciplinary design optimization specialist conference*, pages 1–13, 2005.

[Htt12] M. Httermann. *DevOps for developers*. Apress, 2012.

[Inc] A. E. Inc. What Is REST.

[Inc15] A. E. Inc. What Is Cloud, 2015.

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery–a modeling tool for TOSCA-based cloud applications. In *Service-Oriented Computing*, pages 700–704. Springer, 2013.

[KP13] N. Kumbharana and G. M. Pandey. A Comparative Study of ACO, GA and SA for Solving Travelling Salesman Problem. *International Journal of Societal Applications of Computer Science*, 2(2):224–228, 2013.

[LFM$^+$11] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, and S. Dustdar. Moving applications to the cloud: an approach based on application model enrichment. *International Journal of Cooperative Information Systems*, 20(03):307–356, 2011.

[Mas11] M. Masse. *REST API design rulebook*. " O'Reilly Media, Inc.", 2011.

[MG11] P. Mell and T. Grance. The NIST definition of cloud computing. 2011.

[Mit96] M. Mitchell. Genetic Algorithms, 1996.

[MR12] M. Menzel and R. Ranjan. CloudGenius: decision support for web server cloud migration. In *Proceedings of the 21st international conference on World Wide Web*, pages 979–988. ACM, 2012.

[MTS13] B. Moltkau, Y. Thoß, and A. Schill. Managing the Cloud Service Lifecycle from the User's View. In *CLOSER*, pages 215–219, 2013.

[ND14] A. J. Nebro and J. J. Durillo. jMetal 4.5 User Manual. 2014.

[OGW+14] P.-O. Ostberg, H. Groenda, S. Wesner, J. Byrne, D. S. Nikolopoulos, C. Sheridan, J. Krzywda, A. Ali-Eldin, J. Tordsson, E. Elmroth, et al. The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 26–31. IEEE, 2014.

[SALS15] S. G. Saez, V. Andrikopoulos, F. Leymann, and S. Strauch. Design Support for Performance Aware Dynamic Application (Re-) Distribution in the Cloud. *Services Computing, IEEE Transactions on*, 8(2):225–239, 2015.

[San15] G. Santiago. Performance and Cost Evaluation for the Migration of a Scientific Workflow Infrastructure to the Cloud. 2015.

[SBB+15] J. Soldani, T. Binz, U. Breitenbücher, F. Leymann, and A. Brogi. TOSCA-MART: A Method for Adapting and Reusing Cloud Applications. 2015.

[SGGB] S. Singhal, S. Goyal, S. Goyal, and D. Bhatt. A Comparative, Study of a Class of Nature Inspired Algorithm. In *Proc. of the 5th National Conference; INDIACom-2011*.

[SHT06] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering, 14th IEEE international conference*, pages 139–148. IEEE, 2006.

[Sta13] O. Standard. Topology and Orchestration Specification for Cloud Applications Version 1.0. 25, 2013.

[Ste02] T. L. Sterling. *Beowulf cluster computing with Linux*. MIT press, 2002.

[TBRC+08] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 359–359. IEEE, 2008.

[WAL] J. Wettinger, V. Andrikopoulos, and F. Leymann. Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications.

[WBL14] J. Wettinger, U. Breitenbücher, and F. Leymann. DevOpSlang–bridging the gap between development and operations. In *Service-Oriented and Cloud Computing*, pages 108–122. Springer, 2014.

[Wik15] Wikipedia. Case-based reasoning — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 7-September-2015].

[XA+13] M. Xiu, V. Andrikopoulos, et al. The Nefolog & MiDSuS Systems for Cloud Migration Support. *Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Technical Report*, 8, 2013.

[YJB08] S. Yunxing, G. Junen, and G. Bo. An ant colony optimization algorithm based on the nearest neighbor node choosing rules and the crossover operator. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 1, pages 110–114. IEEE, 2008.

[ZCB10] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[ZSL$^+$11] K. Zhu, H. Song, L. Liu, J. Gao, and G. Cheng. Hybrid genetic algorithm for cloud computing applications. In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 182–187. IEEE, 2011.

[ZZZ06] P. Zhao, P. Zhao, and X. Zhang. A new ant colony optimization for the knapsack problem. In *Computer-Aided Industrial Design and Conceptual Design, 2006. CAIDCD'06. 7th International Conference on*, pages 1–3. IEEE, 2006.

All links were last followed on December 7, 2015

Acknowledgement

I am sincerely thankful to my mentor and supervisor Santiago Gómez Sáez from the University of Stuttgart for his help, guidance, motivation and support during all the phases of my master thesis. I would also like to thank Dr. Vasilios Andrikopoulos for his advices and Prof. Frank Leymann for giving me this wonderful opportunity to do my master thesis at the Institute of Architecture of Application Systems. I am also thankful to my family and friends for their help and moral support during the tenure of my thesis.

Abhilash Mishra

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that 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.

Stuttgart, December 7, 2015 ____________________

(Abhilash Mishra)