Cost Estimation of Service Delivery in Cloud Computing

Sharlene Aminullah
s.aminullah@newcastle.ac.uk

MSc Internet Technologies and Enterprise Computing

Industrial Supervisor
Dr. Stuart Wheater, Arjuna Technologies Ltd.

Academic Supervisor
Dr. Carlos Molina-Jimenez, Newcastle University

August 2012
Abstract

Rapid advancements in cloud computing allowed multiple providers to offer basic computational resources (such as storage and compute power) to consumers (cloud users) as a digital service (infrastructure as a service, IaaS) with the benefits of ‘on-demand’ and ‘pay-per-use’ characteristics of cloud. Similar to computing resources, enterprise software is also expected to be rented as a service along with the cloud infrastructure. With the existing large variety of options, it is extremely hard for a consumer to assess different offers and choose a provider against another to suit their needs. Moreover, the combination of enterprise software providers and IaaS providers result in a complex paradigm, where the absence of a smooth integration between the two business purposes can leave cloud users without utilizing many business benefits offered by cloud computing. Furthermore, determining the cost to pay for these benefits faces many challenges: firstly, understanding the cloud resources and their significantly different pricing models. Secondly, understanding the deployment of the enterprise software in the cloud, finally and most importantly, estimating the potential cost of obtaining different services. To address these challenges, this dissertation proposes more promising and pragmatic alternative which is a novel cost estimation algorithm engraved in a broker engine. This alternative is proposed as a brokering service which allows cloud users to request for a pre-deployment cost estimation of a specific service. The service uses descriptions provided by software and IaaS providers which describe their offerings and pricing policies by means of a specification. The specification that was used in this dissertation was developed using Virtual Deployment Description Language (VDDL).
Declaration

I declare that this dissertation represents my own work except where otherwise stated.

Signed: _____________________________ Date: ________________
I would like to take this opportunity to extend my immense gratitude to those individuals who in one way or another contributed and extended their valuable assistance in the success of this project.

First and foremost, gratitude to Professor Santosh Shrivastava, for promoting me to do this project.

Arjuna Technologies Ltd, for giving me the opportunity to do the project with them.

My utmost gratitude to my industrial supervisor Dr. Stuart Wheater, chief scientist and co-founder of Arjuna Technologies Ltd. whose technical guidance, immense sincerity, invaluable supervision, outstanding patience and enthusiasm assisted in the completion of this project work.

Dr. Carlos Molina-Jimenez, my academic supervisor, whose remarkable patience, dedication and steadfast encouragement helped the completion of this dissertation. I truly thank you for taking your valuable time to review and appraise this dissertation.

Steve Caughey, the CEO and Co-Founder of Arjuna Technologies Ltd, for his valuable industrial insight and suggestions.

Dr. Nick Cook, my academic tutor, for his academic assistance and guidance.

Professor Aad Van Moorsel, head of the School of Computing, for his word of encouragement and assistance in direction of work.

Philip Ritchie and Muneera Parbeen for taking the time to review my dissertation, despite being far from Newcastle.

Sami Alajrami, my fellow mate in the School of Computing, for his encouragement and support during the project.

My better half, for his love, inspiration and sacrifices made. My mother and only brother, my in-laws, all my close friends and the beautiful couple Dr. Afsana and Dr. for the love and support provided throughout the project.
## Contents

Abstract ................................................................................................................................. i  
Declarations ........................................................................................................................... ii  
Acknowledgements ............................................................................................................... iii  

1. Introduction .......................................................................................................................... 1  
   1.1 Motivation ....................................................................................................................... 1  
   1.2 Problem Statement ........................................................................................................ 1  
   1.3 Suggested Solution ......................................................................................................... 2  
   1.4 Aim and Objectives ....................................................................................................... 3  
   1.5 Dissertation structure ................................................................................................. 3  

2. Background Research ........................................................................................................ 5  
   2.1 Cloud Computing Characteristic ..................................................................................... 5  
   2.2 Service Deployment in Cloud .......................................................................................... 5  
   2.3 Migration of Software Product ....................................................................................... 6  
   2.4 Specification Language ................................................................................................ 6  
   2.5 Pricing Models ............................................................................................................... 6  
      2.5.1 Pay-per-use ............................................................................................................... 6  
      2.5.2 Subscription ............................................................................................................ 7  
      2.5.3 Prepaid per-use ....................................................................................................... 7  
      2.5.4 Subscription and per-use ......................................................................................... 7  
   2.6 Pricing Schemes ............................................................................................................. 7  
      2.6.1 IaaS pricing schemes ............................................................................................... 8  
      2.6.2 Infrastructure Resource Pricing .............................................................................. 8  
   2.7 Cost Model Requirements ............................................................................................. 9  
   2.8 Why Service Brokering System? ................................................................................... 11  

3. Approach ............................................................................................................................ 12  
   3.1 Brokering Service Requirements ................................................................................... 12  
   3.2 Designing and Implementing Descriptions .................................................................. 12  
   3.3 Virtual Deployment Description Language (VDDL) Analysis ........................................ 12  
      3.3.1 Components of VDDL ............................................................................................. 12  
      3.3.2 Resources ............................................................................................................... 13  
      3.3.3 Attributes ................................................................................................................ 13  
      3.3.4 Services ................................................................................................................... 14  
      3.3.5 Connectors ............................................................................................................... 14  
   3.4 Designing and Implementing Cost Calculation Algorithm ............................................... 14  
   3.5 Designing and implementing a brokering Portal ............................................................. 15  

4. Description Documents .................................................................................................... 16  
   4.1 Descriptions Design Methodology ............................................................................... 16  
   4.2 Description Document .................................................................................................. 16  
   4.3 Cloud Description ......................................................................................................... 16  
      4.3.1 Collection of Cost Factors ....................................................................................... 17  
      4.3.2 Design .................................................................................................................... 17  
      4.3.3 Implementation ...................................................................................................... 18  
   4.4 Software Product Description ....................................................................................... 19
Cost Estimation of Service Delivery in Cloud Computing, at Arjuna Technologies Ltd.

4.4.1 Design ......................................................... 20
4.4.2 Implementation ............................................. 20
4.5 User Requirements Description .............................. 24
4.5.1 Design ......................................................... 24
4.6 Description Document Summary .............................. 25
5 Cost Estimation ................................................. 26
5.1 Cost and Estimation ........................................... 26
5.1.1 Non-algorithmic Methods ................................. 26
5.1.2 Algorithmic methods ...................................... 26
5.2 Cost Estimation Process ...................................... 27
5.3 Defining Cost Structure ...................................... 29
5.4 Description of the Cost Model ............................... 29
5.5 Cost Model ...................................................... 29
5.6 Cost Estimation Implementation ............................ 31
5.7 Conclusion ...................................................... 32
6 Brokering Service ................................................. 33
6.1 System Architecture .......................................... 33
6.2 System Components .......................................... 34
6.3 Brokering System Work Flow ............................... 34
6.4 Cost Estimation Algorithm .................................. 35
6.5 Brokering System’s Class Diagram ......................... 39
6.6 Design Assumptions and Decisions ......................... 39
6.7 Implementation ............................................... 40
6.7.1 Development Tools & Languages ....................... 41
6.7.2 Implementation Process ................................... 41
6.7.3 Source Code Structure ..................................... 42
6.1.1 Web Portal Implementation .............................. 42
7 Evaluation ......................................................... 46
7.1 Evaluating Against Aims and Objectives .................... 46
7.2 Quality of Background Research ............................. 46
7.2.1 Related Work ................................................. 47
7.3 Arjuna’s VDDL Platform ..................................... 47
7.4 Potential Integration with Arjuna’s AgilityTM Platform .... 48
7.5 Brokering Service Design and Implementation Process .... 48
7.6 Testing ......................................................... 50
7.7 Learning Outcomes .......................................... 50
8 Conclusion and Future Work .................................. 51
9 Glossary .......................................................... 52
10 References ......................................................... 53
Appendix A .......................................................... 56
1 Sample Description Documents ............................... 56
1.1 Machine Type ............................................... 56
1.2 Use Model Type ................................................ 56
1.3 Network Boundary Type ...................................... 57
1.4 Student Registration Application Type ...................... 57
1.5 Software Product (Student Registration) Implementation Type .... 58
1.6 EC2 Cloud Type ............................................... 60
1.7 S3 Cloud Type ........................................................................................................61
1.8 Storage Type..........................................................................................................62
1.9 EC2 Cloud Description (resource and pricing) for Europe ..............................62
1.10 A Software Product Description .......................................................................66
1.11 A Software Product Description .......................................................................75
1.12 A Software Product Description .......................................................................79
1.13 User Requirements Description .........................................................................84
Appendix B ..................................................................................................................86
  1 Configuration, Deployment and Execution Manual ..................................................86
    1.1 Pre-requisites ........................................................................................................86
    1.2 Configuration .........................................................................................................86
    1.3 Deployment ............................................................................................................86
    1.4 Execution ...............................................................................................................86
Appendix C ..................................................................................................................87
  1 UML design .............................................................................................................87
    1.1 Cost Estimation Classes .......................................................................................87
List of Figures

Figure 1.1: Overview of pre-deployment cost estimation as a brokering service.................................3
Figure 3.1: VDDL elements and their association ..................................................................................12
Figure 3.2: A Resource Type with matching attributes types for a Resource ...........................................13
Figure 3.3: An attribute is represented with a reference and a value ........................................................14
Figure 3.4: Services in two different port type (i) required service & (ii) provided service .....................14
Figure 3.5: Connector as a bridge between two services provided by a resource ....................................14
Figure 4.1: Phases of description features collection ..............................................................................16
Figure 4.2: XML Syntax for cloud description ..........................................................................................17
Figure 4.3: Architecture of a Cloud Description Template .........................................................................18
Figure 4.4: XML syntax for DublinEC2Cloud resource ..........................................................................18
Figure 4.5: Syntax for CloudPricing in a cloud description .....................................................................19
Figure 4.6: Syntax for Software Configuration in a software product description ....................................19
Figure 4.7: Example of Student Registration Standard Framework for Type description .......................20
Figure 4.8: XML syntax for DublinEC2Cloud resource ..........................................................................20
Figure 4.9: Design of a sample Software Product Description ..................................................................22
Figure 4.10: VDDL representation of Software Product Description (using 4 software components) .......22
Figure 4.11: Different descriptions of single service deployment ............................................................23
Figure 4.12: Product Using S3 environment .........................................................................................24
Figure 4.13: Typical example of a User Requirements Description ..........................................................25
Figure 5.1: Cost Estimation Methodology ...............................................................................................27
Figure 5.2: Data transfer between sender and receiver ..........................................................................28
Figure 5.3: Network traffic communication between system components ..............................................31
Figure 5.4: Data transfer cost ..................................................................................................................32
Figure 5.5: Service access point in a product ..........................................................................................32
Figure 6.1: High level architecture of Brokering System .........................................................................33
Figure 6.2: Description Documents Inventory with sample documents (TD: type description, SPD: 
software product description, CD: cloud description, URD: user requirements description) ...............34
Figure 6.3: Processing unit of Broker Engine ..........................................................................................35
Figure 6.4: Brokering System represented in a class diagram .................................................................39
Figure 6.5: Brokering Service Oriented Architecture .............................................................................41
Figure 6.6: Source Code packages ..........................................................................................................42
Figure 6.7: Selecting description documents and uploading feature .....................................................43
Figure 6.8: Uploaded description documents on the server .....................................................................43
Figure 6.9: User Requirement Form .......................................................................................................44
Figure 6.10: Estimated cost for individual software products against IaaS provider’s pricing policy ....45
1. Introduction

1.1 Motivation

Cloud computing can be regarded as a market place where several companies offer different services to the consumers (also called cloud users) under different charging models. These services are tailored to meet the cloud user’s business needs they provide through the key benefits of on-demand and self-service [1].

Within that marketplace, there are companies that offer basic computational services such as compute power and disk storage using virtualised technologies. These services are called Infrastructure as a Service (IaaS). Consequently, companies that provide these services are called IaaS providers. A good example of an IaaS provider is Amazon AWS [4] that provides virtual machines (EC2), disk storage (S3) and other services [9]. Complementary to that, there are sophisticated enterprise software (regarded as software product) that are offered by multiple companies such as Content Management, Student Registration, E-Commerce etc. Whereas these software products require computational resources to be deployed onto and possibly automatically [2], the reason is to be used on-demand. Generally, software products are designed and developed for particular business needs (such as organisational accounts management, scientific data analysis etc.). A good example of a software product provider is IBM [5] that provides Enterprise Content Management (ECM) and Software Management (Rationale). In addition to that cloud infrastructure services, are intended and established to be utilised as host for running those software products on-demand and accomplish a certain purpose on a “pay for what you use” principle based on utility (such as gas, electricity and water) service pricing. Such occurrence can allow people to be more flexible in how they address challenges in a short period of time. For example, there is a necessity for massive parallel computation needing hundreds of machines only for few hours. Large scale short time analysis demands availability of computing resources for a short period of time. However, to get the most benefit from the cloud services cloud users would require a rapid deployment process, which demands a Total Cost of Use (TCU) prior to deployment of software products in the virtual environment that is called pre-deployment cost estimation. Subsequently, it is important for providers to expose such information which naturally can fall into a software product description contains deployment information and cloud description contains service and pricing information, because if accurate cost estimates are obtainable, preforming short term micro deployment has a real possibility of becoming the new dominate pattern of gaining access to computing resources. If people get the ability to do so then this would radically change the way they interact with computing resources.

It is worth pointing out that it is reasonable to expect that some IaaS providers will offer complete services, that is, both the software product and the IaaS to run it. Yet in this dissertation the assumption is that the software product and the IaaS are offered by different companies – we consider that this is a more general scenario.

A common business model used by IaaS provider is to charge consumers on a pay-per-use basis where they periodically pay for the resources they have consumed [6]. The IaaS can be regarded as a market place where several IaaS providers compete against each other offering equal or similar services at different prices and different conditions about the usage (for example, number of simultaneous users) of the service and its QoS (Quality of Service, such as response time). On the other hand a similar business models assumed to be used by software product providers, which is also regarded as a part of market place where software product providers offer various options of selecting a single software product description (for example ECM Description) for a specific service from a choice of different implementations (such as ECM Description 1, ECM Description 2) that differ in cost, type and amount of resources to run it. It follows that the price that a cloud user pays for using a service depends on a) the service provided by the software product provider; b) the type and amount of resources needed by the service, c) on the IaaS provider selected and most importantly on (d) cost estimation. “With the arrival of cloud computing has arrived an increase in choice”, “what is the point of being able to deploy an application in minutes, if it takes day to work out if it is within budget: need automatic cost estimation”

1.2 Problem Statement

A question that arises with the pay-per-use charging model is how much will it cost the cloud user to rent a given service from the cloud; for example, how much will cost me (cloud user) to run a Student Registration Service in the cloud for five days, knowing that it needs 100 GB of input data? This is called “cost estimation”
and is the central topic of this dissertation. A similar question that cloud users frequently ask is what IaaS provider can offer me the best alternative?

With a large number of options available, it is not trivial for a cloud user to assess different offers and choose which IaaS provider should be used and which software product to rent. Even in the situation where the required functionality is offered by a particular software product which is only deployable on a particular IaaS provider, it is still useful for the cloud user (say for budget planning) to have an estimation of the potential cost of the service before committing to use the service. Currently a cloud user would need to collect manually the pricing policies from different IaaS providers, evaluate them and make a decision. The author’s argument is that this is not the best alternative as it puts unnecessary burden on cloud users that, in worse cases, might not have the needed expertise or resources to conduct such evaluations with acceptable accuracy.

1.3 Suggested Solution

The author believes that a more promising and pragmatic alternative is to offer such evaluations as another service; precisely, as a brokering service. Therefore this dissertation envisions and discusses brokering services as a market place where companies with expertise in cost evaluations offer their services to cloud users in return for fees. They are called as brokers and reply on broker engines to estimate cost. In this dissertation a broker engine facilitates pre deployment cost estimation based on service requirement specified in higher level objectives.

Leaving specific implementation details aside, the following pieces of middleware will be at the heart of brokering services:

- **Descriptions**: are self-defined methodologies for specific needs, which are divided in three major groups:
  - **Cloud Description**: a document that describes the pricing policies of the IaaS provider with resources for deploying a software product. Examples of described elements are instance price and storage price for IaaS providers.
  - **Software Product Description**: a document that describes two attributes; (i) set of descriptions of deployment configuration and (ii) set of descriptions for cost estimation. This document specifies for example deployable components, deployment location, connection between components, the amount of data stored and so forth for a single software product.
  - **User Requirements Description**: a document that describes the specific requirements for a particular service. The document specifies information for example (i) the type of service, (ii) duration and (iii) amount of data input for the service.

- **Cost Estimation Algorithm**: an algorithm that computes the cost on the basis of multiple software products descriptions and clouds descriptions. In brief, this algorithm (i) processes the descriptions, (ii) extracts numerical values from the descriptions and (iii) returns an estimated cost for a single service.

- **Web Portal**: the interface to the cost estimation service. Via the portal a cloud user can request for pre deployment cost estimation. The portal also provides the management interface implemented by the broker for encapsulating the brokering service.

Ideally, software product descriptions and cloud descriptions need to be accessed and processed programmatically by the broker. For this to be possible, they need to be expressed in standard notations. Notations specify what operations the service provides and what charges the service incurs. The author believes that in the near future, standardisation bodies will take care of this issue and produce the needed standards. In the absence of such standard, in this dissertation they are defined by the author. The standards will be explained in detail in Section 3.3.2.

The expectation is that brokering services will be offered under different scenarios, Fig 1.1 shows the abstract view of a potential one. On the basis that it is general enough, the view will be used by the author in this dissertation to build and explain the arguments.
Figure 1.1: Overview of pre-deployment cost estimation as a brokering service

In figure 1.2, cloud user requests pre-deployment service cost estimation from the broker. The broker is an independent third party intermediary between the cloud user, software product providers (for example, Software Product Provider A) and IaaS providers (for example, IaaS Provider A). On the basis of a pre-conceived business agreement with both providers; the broker holds a set of software product descriptions provided by multiple software product providers and cloud descriptions from multiple IaaS providers. These descriptions are constructed on the basis of standard notations that are presumably provided by standard bodies. Upon receiving “cost estimation request” for a particular service, the broker engine matches all available software product descriptions that conform to the cloud descriptions and generates an approximation of possible cost for running the requested service in the cloud infrastructure.

1.4 Aim and Objectives

Aim

The aim is to develop automated techniques that enable estimation of the cost of service delivery in cloud computing environments to the cloud users. Though some research have been done into cost estimation in cloud computing discussed in chapter 2.12, pre-deployment cost estimation is currently an active research topic in need of deeper investigation. To that end, this dissertation will examine cost estimation from the perspective of being a brokering service.

Objectives

To achieve the deeper understanding of the possibilities associated with providing cost estimation as a brokering service the author built a broker engine. The main goal was to build a framework to allow cloud users to contact a broker, request a specific service and receive estimates of potential costs to run it. The primary objectives to accomplish the goal in this dissertation are summarised as follows:

- Develop standard notations for generic software products and cloud pricing policy descriptions.
- Design a formal specification of describing software products and cloud’s pricing policy.
- Build a base level cost model.
- Design a cost estimation algorithm for a broker engine.
- Implement a web portal for the cloud users to estimate pre-deployment costs.

1.5 Dissertation structure

This dissertation is structured as follows:
• Chapter 2: **Background Research**, presents contextual features to articulate the proposed solution understanding and discusses related approaches.

• Chapter 3: **Approach**, describes the methodology followed to provide cost estimation as a brokering service and outline directions of proposed work.

• Chapter 4: **Description Documents**, presents how the descriptions for cloud, software product and User Requirements were designed and implemented.

• Chapter 5: **Cost Estimation**, discusses how the cost model was designed and implemented

• Chapter 6: **Brokering Service**, describes the architectural design and implementation of the brokering system providing pre-deployment service cost estimation.

• Chapter 7: **Evaluation** evaluates the performance of this study and performs a critical analysis for each of the objectives we set out to achieve.

• Chapter 8: **Conclusion and Future Work**, brings to a close the dissertation and proposes future work that could have been done in this area.
2 Background Research

This chapter provides an outline of the cloud computing paradigm, where cost estimation fits in, discussing the definitions of cost factors followed by highlighting the state of the art and prior work in the fields of cost estimation of service delivery.

2.1 Cloud Computing Characteristic

The concept of IaaS (Infrastructure as a Service) in cloud computing has emerged from the need of, “on demand” resource provisioning to serve cloud user’s business goal. The initial referent has evolved from providing the abstraction of IT resources and services from the underlying infrastructure, later that followed the scaling mechanism in a shared, multitenant and elastic environment. Clear definition of Cloud is yet in the process of evolution; however Ian Foster has proposed some of its distinctive features in [7]:

“Cloud computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.”

The National Institute of Standards and Technology (NIST) in [8] also proclaimed cloud computing as “a model for enabling convenience” in terms of IT capacity on demand. Following five characteristics defined by NIST (National Institute of Standards and Technology) [8] highlights the major characteristics:

- **On-demand self-service**: Individuals can unilaterally provision computing capabilities, automatically without requiring human interaction.
- **Universal network access**: Capabilities are available through standard Internet-enabled devices.
- **Location independent resource pooling**: Dynamic processing and storage demands are balanced across a common infrastructure with no particular resource assigned to any individual user. Focusing a sense of location independence higher level of abstraction is introduced.
- **Rapid elasticity**: Consumers can increase or decrease capacity at will in any quantity at any time.
- **Pay per use**: Optimised resource consumptions are charged fees based on their usage for particular computation.

2.2 Service Deployment in Cloud

In service oriented architecture (SOA) a service deployment is the process to make the service ready for use [34]. The process refers to a configuration of a collection of services that specifies the information that is necessary to instantiate the services, as well as dependencies on other services. Service deployments can be created for applications that deploy or access the services. Service deployment configuration can be stored in described locations of (i) Service Metadata Repository, (ii) Container Deployment File or (iii) XML file. Metadata is generally interpreted as “data about data” or “data about applications” [35]. In the context of this dissertation metadata would be the information about the software products. Service deployment metadata can be exported to an XML file, which can be operated to import service deployment metadata into a repository. Repository generally refers to a location for storage, which can either be persistent (all time availability) or non-persistent (available when compute instance is online). Storing service deployment metadata reduces administration and maintenance for the file content. A service deployment contains: (i) service deployment groups – collection of deployments, (ii) services and service initialization data. Within each service deployment group, services must be defined within each group. Service definitions contain the following information:

- **Service types** (interfaces) - designate which service interfaces are implemented by the service.
- **Service configuration** - specifies the executable code that is used to create the service, the service's optional configuration data, and the service's configuration user interface.
- **Service dependencies** - specify other services on which the service depends. When a service is defined, other services must be defined on which that service depends. For example, the Authentication Service uses the Logging Service. Therefore, when an Authentication Service is defined in a service deployment, the Logging Service must be defined as a dependency.
- **Service names** (for remote access only) - specify named services for remote access.
2.3 Migration of Software Product

The essence of IT industrialisation has been inspired from the specialised industries in mass production (such as auto industry) lines, which implies the necessity to standardize offered services that increases automation significantly. To utilize cloud technology at its highest potential providers face major challenges such as portability, standardization of service definitions [26] etc. However, the ability to move cloud services and their components between providers (that is software product providers and IaaS providers) ensures an adequate and cost-efficient IT environment and provides selection service among multiple providers. Although research has addressed issues such as mobility and migration of software products a functional level [27, 28] cost factors have a major impact on cloud service usage on the basis of management and operational tasks [26]. Delegation of software product components into virtual computing environment implies the need of standard programming models that are able to utilise cloud potential by wider adoption. Research has been done in to provide standardised management of cloud services. Standardised format can assist quick deployment of complex software products (as in applications) in the cloud.

2.4 Specification Language

One of the many requirements of cloud hosted applications (as software product in this dissertation) is the standardised specification for describing entities (such as providers). Specification language allows modelling internal organisation of cloud service. With the language software products provider can provide more transparency for IaaS providers [29]. Describing software products and cloud using an abstract model the complexity of the cloud platform can be hidden from the cloud users who are merely interested in back-end of cloud technology. Despite the advantage of existing specification language there are many obstacles for solving the “deployment problem” between software product providers and IaaS providers. The “deployment problem” can be solved only with a description written in computer amendable notation. Research has addressed similar technology to capture the management aspects of cloud reusability. Topology and Orchestration Specification for Cloud Applications (TOSCA) [30] is a standardised work from OASIS that facilitates the portable, automated, and reusable management of cloud services life cycle. Similar research has also found another technology, Distributed Application Description Language (DADL) [31], which is primarily used to describe the behaviour and needs of distributed applications and used via a cloud-computing infrastructure. Cloud# [32] on the other hand another specification language which has been designed to express cloud service delivery in terms of deployment.

2.5 Pricing Models

A pricing model is a framework for billing users with the associated cost of offered services, for example, providers offering users to use 100 instances per hour per month at the rate of five pence per hour. Constant unit price facilitates pay-per-use pricing model whereas committing to a long term contract on constant price of service unit denotes subscription pricing model according to Weinhardt et al [22]. However, there two types of pricing schemes were identified in cloud computing: (i) fixed pricing and (ii) dynamic pricing [23]. During investigation of various IaaS provider's offerings four types of pricing models have been identified [9, 10, 11, 12]. The following analysis discusses the key features of each pricing model.

2.5.1 Pay-per-use

This pricing model has been followed from utility metring concept identified in [6]. This literarily means pay only for the usage amount, while removing the need of any upfront payment for extra resource. In general computing resources are charged at per unit of time usage [13]. The model is most commonly used for underlying infrastructure such as server instance which includes CPU, RAM, disk storage, bandwidth capacity. Price for all the server instance attributes therefore vary depending on the IaaS provider’s pricing policy, how they provision a single instance with specific resource size and capacity. For example, standard instance offering by Amazon [9] is “Small Instance: 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of local instance storage, 32-bit or 64-bit platform”. Following the instance charge other resources are following usage base policy (e.g. data usage for network bandwidth). Data transfer is charged at per GB in and out, block storage is charged at per GB data storage per period of time. This type of pricing model removes long time commitments and provides elasticity by offering on-demand resource usage strategy [14].
2.5.2 Subscription

Subscription pricing model allows consumers to subscribe computing resources according to their need in advance. Generally the subscription is identified by an agreement or a type of contract stating period of time. Dedicated Servers or Reserved Instances often offered in a bundle as a package. Unlike pay-per-use model this model requires an upfront payment following a termed commitment (long/short). To attract more cloud users’ providers offer this pricing in a discounted rate.

2.5.3 Prepaid per-use

Prepaid per-use pricing model requires consumers to buy credit up front and the charge for on-demand usage gets billed from that credit.

2.5.4 Subscription and per-use

This model put restrictions on mandatory advance renting for Dedicated Servers, however provides flexibility of on-demand servers need which is billed at per-use charges.

A consolidated evaluation of used pricing models for IaaS providers’ [9, 15, 16, 17] are presented in the table 2.1 bellow:

<table>
<thead>
<tr>
<th>Pricing Model</th>
<th>Offering Type</th>
<th>Charge</th>
<th>Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pay-per-use/Pay-as-you-go</td>
<td>On-demand Instances</td>
<td>Per hour</td>
<td>• Amazon on-demand and spot instances</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Rackspace cloud servers</td>
</tr>
<tr>
<td>Subscription</td>
<td>Dedicated Instances</td>
<td>Upfront</td>
<td>• Rackspace servers (monthly)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• GoGrid dedicated servers (monthly)</td>
</tr>
<tr>
<td>Prepaid per-use</td>
<td>On-demand Instances</td>
<td>Per hour use deducted from prepaid credit</td>
<td>• GoGrid cloud servers (hourly),</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ElasticHosts hourly-burst cloud servers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Joyent SmartMachines (daily)</td>
</tr>
<tr>
<td>Subscription with per-use</td>
<td>Dedicated Instances</td>
<td>Upfront periodic pay</td>
<td>• ElasticHosts monthly cloud servers with daily usage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Joyent monthly SmartMachines with daily usage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Amazon reserved instances (1 or 3 years)</td>
</tr>
</tbody>
</table>

2.6 Pricing Schemes

In cloud computing the pricing scheme is an outline of the overall pricing approach which encompasses the principles for pricing the specific product involved in the service offerings [23]. The initial approach towards cost modelling requires an in-depth understanding of Cloud Computing Services pricing scheme. Several analyses have been conducted to generate a realistic view of IaaS provider’s computing model and their associated pricing policies. The selection of the service providers is based on the research conducted in [18]. To support the findings more literatures published in the similar filed have been consulted and they can be found in [18,19, 20, 21]. The target has been to explore the cost factors to architect the estimation model for service delivery in Cloud Computing. The prolific definitions reinforced generating a general statement for Cloud Computing Pricing. It has become evident from the analysis that there is a wide range of pricing schemes available for various cloud offerings. A list of the analysed services by various IaaS providers is presented in chapter 2.10.1.
2.6.1 IaaS pricing schemes

Estimating a pre-deployment cost for infrastructure resources offered by various providers’ demands an evaluation of current market offerings and pricing schemes in business practice. This section highlights consolidated discoveries of infrastructure services offered by various IaaS providers in current marketplace. Therefore the list contains the products specialised for IaaS in current marketplace.

Table 2.2: IaaS pricing schemes offered by IaaS providers

<table>
<thead>
<tr>
<th>Service Provider</th>
<th>Pricing Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Web Service</strong></td>
<td></td>
</tr>
</tbody>
</table>
| Elastic Compute Cloud (EC2) [9]   | - Price per hour depending on memory in GB, processing units, storage space in GB and platform (32-bit or 64-bit)  
- Optional reservation of further units  
- Price per transferred GB (outbound and inside to another Amazon Web Service) |
| Simple Storage Service (S3) [41] | - Price per GB  
- Price per transferred GB (outbound); Inbound data transfer is free of charge  
- Price per 1,000 or 10,000 queries, respectively (PUT, COPY, POST or LIST) |
| **Cisco**                        |                                                                                |
| Unified Service Delivery [63]    | - Price per GB stored only regardless of inbound or outbound |
| Cloud Storage [64]              | - Price per GB stored only regardless of inbound or outbound |
| **Google**                      |                                                                                |
| Compute Engine [12]             | - Pricing per machine type depending on virtual cores, memory, local disk space in per hour basis  
- Networking charge depends on zone, region and internet  
- Storage charged at per TB in monthly basis |
| Storage Engine [65]             | - Pricing per TB data store (additional storage upon request)  
- Network pricing for per TB stored with inbound and outbound |
| **HP**                          |                                                                                |
| Cloud Compute                   | - Per hour charges with no minimum and long term commitments  
- Charge on standard instance type, estimated hourly rate with monthly service hour |
| **IBM**                         |                                                                                |
| SmartCloud                      | - Per hour charges  
- Price for virtual compute resource with Operating system along with reserved or unreserved capacity, platform (32-bit or 64-bit)  
- Persistent storage charged per GB per hour, storage Input and Output access requests per million  
- Internet Data Transfer charge at per GB inbound and outbound data |
| **Rackspace**                   |                                                                                |
| Cloud Files                     | - Price per saved GB (monthly rate of storage per GB)  
- Bandwidth charged per GB transferred  
- Price per query for file size type |
| Cloud Servers                   | - Price per hour depending on memory in GB, processing units, storage space in GB and platform (windows & non windows)  
- Price per transferred GB (outbound and inside to another CloudServers) |

2.6.2 Infrastructure Resource Pricing

**Instance Pricing:** Evidently the evaluation results from the table 1.2 shows that the pricing policies are dominant by usage-dependant hourly charge. Although charges are claimed to be hourly basis, cloud users are billed on an average of 730 hours per month. Additionally if cloud user has used an instance for 10 hours per day only for 15 days in a month then the charge would be on average of 5 hours for the entire month.
Storage Pricing: Many providers’ promotional trick is low price offerings for persistent storage, however it is evident that cloud users are often unaware of the hidden costs for inbound or/and outbound data transfer within the provider’s own infrastructure. The pricing scheme gets more complex when different network boundary (e.g. zone, region) charges are applied on top of regular bandwidth price. However, eventual cost depends on internet download rates.

Network Traffic Pricing: A common practice between the providers is to offer basic package at a fixed rate in addition to service extension on per request. Then again the study shows extra usage outside of the basic bundles costs at a higher rate than cloud user’s inclination [9, 11, 12]. Alternatives are offered to back this business policy with discount on usage dependant prices. Consequently the pricing schemes are offered in combination with various pricing models, such as pay-per use, pay-as-you-go, and subscriptions described in chapter 2.3.

The following questions have been noticeably common by comparing IaaS provider’s pricing schemes:

Table 2.3: Questions to estimate cost of resource usage

<table>
<thead>
<tr>
<th>Pricing</th>
<th>Metering exercise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pricing</td>
<td>• How many CPU hours/Instances required</td>
</tr>
<tr>
<td></td>
<td>• How much bandwidth will be consumed</td>
</tr>
<tr>
<td></td>
<td>• How much storage required</td>
</tr>
<tr>
<td></td>
<td>• How much data transactions occurred for storage</td>
</tr>
</tbody>
</table>

Outcomes to the metering exercise queries are then categorised in one logical group of cost parameters. A cost parameter in this dissertation is the dimension of the cost model. In the case of constructing the cost estimation model choice of parameters will assist defining the cost. Common parameters used by current IaaS providers are consolidated in the following table.

Table 2.4: Cost Parameters for IaaS service charge

<table>
<thead>
<tr>
<th>Cost Type</th>
<th>Cost Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>IaaS Service Charge</em></td>
<td>• <em>Instance Size</em> – size of virtual machine offered by IaaS provider</td>
</tr>
<tr>
<td></td>
<td>• <em>Computing power</em> – processing power of the virtual machine</td>
</tr>
<tr>
<td></td>
<td>• <em>Storage capacity</em> – amount of data a given storage container can hold</td>
</tr>
<tr>
<td></td>
<td>• <em>Inbound data transfer amount</em> - data coming into the virtual machine</td>
</tr>
<tr>
<td></td>
<td>• <em>Outbound data transfer amount</em> – data going out of the virtual machine</td>
</tr>
<tr>
<td></td>
<td>• <em>Provider’ internal data transfer</em> – data used within IaaS environment</td>
</tr>
<tr>
<td></td>
<td>• <em>Number of queries</em> – number of access call to the storage container</td>
</tr>
<tr>
<td></td>
<td>• <em>Domain</em> – deployment location of a virtual machine</td>
</tr>
<tr>
<td></td>
<td>• <em>Basic service charge</em> – price tag selected for single resource usage by provider</td>
</tr>
</tbody>
</table>

2.7 Cost Model Requirements

The aim of this dissertation is to provide pre-deployment calculations of software product running costs on the different cloud platforms. Pre-deployment is the state which exists before an application that is deployed in a virtual environment. The term cost estimation refers to the possible charges for running the deployed software product in that virtual environment.

Deploying software products (called application) in the cloud is not a trivial task. The task requires information about virtual environment creation, service configuration etc. Often the configuration requires facts and figures about nodes in the deployment along with specific parameters defined by the users. In addition, deployment details contain complex hierarchy of interdependent services that must be configured in the correct order. Hence deployment becomes a complex task. To estimate cost for such a complex deployment requires an ordered investigation for essential factors. Based on the experiment in section 2.3 the following requirements have been developed to construct a pre-deployment cost estimation model for cloud services:
Software Configuration Parameters

They describe the composition of loosely coupled independent components of a system. Components are regarded as part of the starting platform for service-oriented architectures (SOA), whereby a component is converted by the Web service into a service and subsequently inherits further characteristics beyond that of an ordinary component. To develop standard notation for software product in the approach suggested in [section 1.4] measurement elements need to be identified. Hence the project advocates the need to define software implementation parameters for cloud deployment configuration. A parameter is the dimension to construct a software product deployment model in the cloud.

Deployment Factors

They describe the metadata for defining cloud service deployment [30] – that is, application files providing the service’s functionality. Structure of a cloud service is captured in a service topology using the deployment factors as nodes and relationships. Nodes characterize the service’s components, and relationships connect and structure nodes into the topology [26]. A Topology Template defines the deployment topology model of a service. Providing particular service often require complex environments in which to run. Setting up these environments involves many steps that must be repeated each time the application is deployed. In order to minimize errors and save time, it is important that these steps are automated. A deployment service therefore will describe the nodes and services they require, and then automatically provision, and configure the application on-demand. This process should be simple and repeatable, whereby a node specifies the occurrence of a Node Type as a component of a service during deployment. A Node Type defines the properties of such a component and the operations (via Interfaces) available to manipulate the component. Hence the next stage is to identify and consolidate deployment factors for various applications.

Cloud Pricing Factors

In the event that a single IaaS provider is not able to provide low-priced infrastructure resources, it becomes necessary to provision cost estimation for resources from multiple IaaS providers. Analysis of IaaS provider’s pricing model therefore has allowed to identify chargeable factors for application deployment. It has been learnt that there are many pricing models available in the current market, per-user, per-click, flat rate, subscription, multi-tier packages, full custom, freemium, CPU power etc.

Categorised Pricing Model Elements

Various pricing model evaluation has aided to categorise related factors from IaaS providers’ offerings. They are for example pricing deals, price lists, scheduled discounts, packaging & licensing, value metric. This information can be categorised into two classes (i) Price Details, providers suggested retail pricing and (ii) Price Structure – the ratio of the fixed cost to variable cost. The table shows how the cloud computing attributes and elements can be placed into these two category, which highlights the factors that need to be considered during cost model construction.

Table 2.5: Categorised Cloud Pricing Elements, Attributes and Models

<table>
<thead>
<tr>
<th>Price Model elements</th>
<th>Cloud Computing Attributes</th>
<th>Cloud Computing Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price Details</td>
<td>• Pricing Deal</td>
<td>• Full custom</td>
</tr>
<tr>
<td></td>
<td>• List Prices</td>
<td>• Flat rate</td>
</tr>
<tr>
<td>Price Structure</td>
<td>• Scheduled Discounts</td>
<td>• Free Tier</td>
</tr>
<tr>
<td></td>
<td>• Packaging and Licensing</td>
<td>• On-demand, Subscription</td>
</tr>
<tr>
<td></td>
<td>• Value Metric</td>
<td>• Multi-tier packages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Per-user, Per click</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Compute Power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Data Amount</td>
</tr>
</tbody>
</table>
Chargeable Factors

Estimating cost for service use requires identifying the dynamics that a charge will be applied on. Therefore chargeable factors have been identified from pricing details and price structures for application deployments. Factors are defined using different units. A unit can possibly be a component of the system, amount of data the system requires to store locally, bandwidth or a combination of compute, network traffic and storage resources. Each of this unit becomes the “atomic factor” that derives application component workflow analysis, network traffic analysis, and storage consumption strategy. A factor for example is as Amazon uses per hour charging for any infrastructure resource [4].

Cloud Pricing Strategies

Investigating pricing strategies means articulating pricing options for each chargeable factor. The mode then requires definitions of strategies that require to be established before cost model construction. For example amazon charges for only outgoing data from amazon’s cloud environment [4].

2.8 Why Service Brokering System?

In general a broker is an intermediary company between consumers and providers [36]. In cloud computing a broker assists communication between the cloud user and the IaaS providers. In the proposed broker based model, IaaS providers register their capabilities and pricing policies and software product providers register their operational capabilities of the software product with the broker. As a result cloud users can query a central broker system about the category of service (for example Student Registration, Payroll or Scientific Analysis) that the cloud users need and the cloud infrastructure providers that are available in a marketplace to match the service. For example, airline companies may provide real time flights (such as London to New York) for traveller based on travelling destination and budget. A cost estimation application for flights on a web browser can use the traveller’s information to recommend flights to the traveller around the time of travelling. The system can also use other companies (such as hotel provider) to aggregate traveller requirement and deliver sophisticated requirements based services.

Despite being less dynamic then stock market cloud market pricing does change from time to time. For example, Amazon WS published news on 8th of December 2009 stating [39]:

Table 2.6: Announcement of Amazon Web Services price change in 2009

<table>
<thead>
<tr>
<th>“AWS Announces Pricing Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>We’re excited to announce some price changes for Amazon S3 and Amazon EC2, as well as a special data transfer promotion that will run until June 30, 2010”.</td>
</tr>
</tbody>
</table>

Cloud user would therefore need most recent market data to make a best decision. In that sense periodical change being made by the IaaS providers therefore can be supported by broker’s system. The system was also designed to break the reality of provider lock in where cloud users may choose to select a IaaS provider despite offering either less or more expensive options.
3 Approach

This chapter describes the approach that will be taken investigating how software products and cloud can be described for estimating the cost produced by a brokering system.

3.1 Brokering Service Requirements

As mentioned in section 1.3 the pieces of middleware required for brokering system will be developed by: Designing and implementing description documents, cost estimation algorithm, a broker engine and a broker web portal.

3.2 Designing and Implementing Descriptions

Cloud IaaS offerings have emerged as a great platform for distributed computations [37]. Rapid resource provisioning is empowered by the layer of virtualisation in the cloud; however a great deal of complex infrastructure lies at the back of the virtualisation. One of the hidden aspects is the network topology. In cloud computing network topology refers to the technique of the rented virtual machines are interconnected within the cloud environment. Moreover, understanding how that topology information can be captured for extracting cost information requires a mechanism of designing the descriptions in a generic manner to support reusability of the descriptions. Therefore, three sets of descriptions of (i) cloud description, (ii) software product description and (iii) user requirements description were created prior to building cloud brokering system that will be elaborated in the chapter 6.

3.3 Virtual Deployment Description Language (VDDL) Analysis

VDDL is an XML based technology developed in December 2007 by Stuart Wheater at Arjuna Technologies Ltd which is originated from Arjuna Research Team, Computing Science Department, University of Newcastle upon Tyne and formed in 2002 via MBO from Hewlett-Packard.

The specification was developed as a single approach to describe the requirements of the definition of the application topology that assists to remove any impedance between requirements and availability of resources. VDDL is a typed language, in which all types are abstract definitions of the capabilities of a concept such as resource, service etc. Types are dynamic identifiers to all concepts within the language which is able to take on multiple types. The language serves its purpose by defining four major description types:

(i) Deployment Description – captures user requirements for needed service.

(ii) Environment Description – captures IaaS provider’s description such as pricing.

(iii) Product Description - captures deployment topology of a preferred service.

(iv) Type Description - captures the standard information for the above descriptions (deployment, environment and product).

3.3.1 Components of VDDL

To support the type system in VDDL, the language provides several common elements as: resource, attributes, ports (used to be referred to Services) and connectors that can be used to aid with the description and requirements of an application. Figure 3.1 represents the elements of VDDL where a resource represents a deployable component of an application.

![Figure 3.1: VDDL elements and their association](image)
For example, an e-Commerce application requires a server to host the application on the Internet; therefore in this particular example the server is an application component that can be represented as a resource displayed in figure 3.1. A resource exposes services and also have requirements for services exposed by other resources. For example, a server provides a hosting service to deliver web pages on the request to clients using the Hypertext Transfer Protocol (HTTP); this means delivery of a HTML document can be a requirement presented as a resource. A service within the language represents some form of behaviour that a resource exposes. The feature is currently referred as a port in VDDL. A port interacts with either by client applications or other components of an application. Resources and services also expose attributes which are configurable items of the resource or the port. Finally there are connector elements which define the technique in which two services or attributes should be connected together when the application resources are deployed. The following sections of the document visit each of the elements in turn giving a wider explanation of their role within the language.

3.3.2 Resources

The fundamental idea of VDDL is based around the concept of ‘Resource’, which abstractly refers to a deployable application. No implementation or internal operation is exposed by VDDL; instead the language simply allows for the specification of a resource in terms of how it is constructed and the capabilities it has. A resource has a mapping between itself and one or more pre-defined resource types. The type of the resource defines the capabilities of the resource and the potential behaviour it can exhibit.

‘Resource’ and ‘type’ can be demonstrated by specifying some resource types for the middle tier of an application. Supposing each of the business units within the middle tier are Web-Services that run within an Application Server (e.g. JBossAS). Therefore, an application server is a resource that needs a method to define what exactly the resource can do and how they can be deployed and that can be defined by a ‘Type’ for the resource that provides resource specific information. For example, an application server exhibits its capabilities in ‘Application Server Type’ and JBoss application server exhibits its capabilities in ‘JBoss Type’. The following diagram shows how VDDL constructs an application requirement using resource type:

![Figure 3.2: A Resource Type with matching attributes types for a Resource](diagram)

In figure 3.2 a single resource is represented as the rectangular, which expresses the requirements of an application in a ‘Resource Type’, where the requirements are represented as attributes. Supporting the ability to structure the application requirements in a recursive manner VDDL defines the notion of containing pre-determine types for any particular resource, represented in a diamond shaped in figure 3.2. A resource within another resource is represented as ‘containable resource’ in VDDL. This mechanism facilitates the creation of the hierarchy of an application requirement. For example, an application server (AS) requires an operating system (OS) to run on, therefore, AS is a containable resource within OS. In this particular example ‘contains’ relation is a means of modelling the way in which certain applications are composed, which can be represented in VDDL as: a ‘Operating System Type’ contains an ‘Application Server Type’.

3.3.3 Attributes

Attributes are the configurable information of VDDL elements, such as resource, port or connector.
Cost Estimation of Service Delivery in Cloud Computing, at Arjuna Technologies Ltd.

As shown in figure 3.3, the largest rectangular is a resource, smaller rectangulars are attributes and the box with ref and value box is just a representation of an attribute. The values of attributes are used in evaluation of application requirements. VDDL also mandates an association between an attribute and one or more than one attribute types. These attribute values are allowed to be customised to be set and referenced throughout the deployment process. For example, in a JBoss resource the incremental deployment capability for fast deployment in VDDL is presented with an attribute within a resource as shown in figure 3.3, where a reference can be ‘deployment capability’ and value can be ‘incremental’.

3.3.4 Services

Within VDDL a service is a particular behaviour that can be exposed by a given resource and accessed over a given types connections. A service is only made available when the resource that exposes a port is deployed. Port. For example, a port from figure 3.4 shows two different services that an Application Server provides such as (i) hosting application and (ii) need authentication service.

As with resources, ports can conform to one or more pre-defined types. Resources can make available ports of a certain type and can also define requirements for services of a certain type. For example, a ‘metered port type’ represents that the port can be considered for calculation. As shown in figure 3.4, both the dark and white circles represent the port that an application server exposes for the services.

Service Dependencies

Not only can a resource make available given services, it can also define requirements for services exposed by other resources. For example, a service dependency for an application server is the requirement for an authentication server.

3.3.5 Connectors

A connector defines a method through which two ports should be connected. For example, a connector connects an application service to an authentication service provided each service is represented as a resource.

A connector is a relationship between itself and one or more connector types. Perhaps the best description of a connector is an abstract bridge (figure 1.7) between two ports. In the case of services this may define the communication channel between the two allowing data to be exchanged over the network.

3.4 Designing and Implementing Cost Calculation Algorithm

Cost estimating is defined as the art of predetermining the lowest realistic charge of an item or activity which assures a normal profit [38]. Rapidly changing knowledge base of cloud environment increases the need for having an intelligent procedure to detect the new changes. Similarly software product marketplace bringing new
software products opens more choices for the cloud users also needing a clever mechanism to spot the new deviations. A cost calculation algorithm therefore will aid to offer approximation against defined cost data (provided by pricing policies for a new software product description) and will also solve the problem of choice for cloud users.

### 3.5 Designing and implementing a brokering Portal

A portal was designed and implemented to hide all the complexities of data processing, analysis and commutation for cost estimation. Most importantly the system was designed to run through a web browser, the most common platform for Internet users to provide dynamic interaction for requesting and getting a response. The portal also gives a single point of contact for cloud users and management for a broker by provisioning the benefits of leveraging multiple service providers (both software product and IaaS providers).
4 Description Documents

This chapter elaborates the methodology that was followed to design and implement the description documents required for the brokering system. A running example will be used to show and discuss the descriptions for that specific example: A cloud user requires a Student Registration Service to run on the cloud infrastructure and make a cost estimation request to a broker.

4.1 Descriptions Design Methodology

An iterative process was followed to design and implement the required set of descriptions for the brokering system. Although brokering system is capable of undertaking other tasks (such as deployment of a service) cost estimation was the primary goal for this dissertation.

![Diagram of Phases of Description Features Collection](image)

**Figure 4.1: Phases of description features collection**

The description design process followed three distinct phases described in figure 4.1: Initially process started with a definition phase by finding major features to incorporate into the description model, encoding phase by following a mechanism to translate the features to the cost modelling in VDDL and finally the refinement phase where chosen features were advanced through a sophistication for more accurate estimation.

4.2 Description Document

A description document is an XML file, which defines a set of rules for cost estimation in a format that is readable by the broker engine. A typical description document contains an identifier (i.e. URL specific to a type) that makes the documents identifiable for the broker engine. In this dissertation the identifier is particularly designed for the standard notations which are described in a set of type-specific properties. These properties are positioned into three levels [26]: **types** define reusable entities and their properties, **interaction templates** form the cloud service’s topology using these types, which are then instantiated as **instances** of the described cloud service. Since concrete types aren’t part of the specification (VDDL), therefore it was standardized for the respective domain (such as software product or cloud).

4.3 Cloud Description

A Cloud Description defines the formal description of pricing policy of the IaaS provider with resources for deploying a service, including pricing structure, properties and value as the actual price. The concept of a “Cloud Description” is used to specify IaaS provider’s infrastructure resource model. Typically cloud pricing is defined for IT services are provisioned in an IT infrastructure. For this project the Cloud Description was constructed with the high level knowledge of current IaaS providers’ offerings (in particular Amazon WS’s pricing model [9]). A simple example for cloud description explained as, information about infrastructure resource price as (i) instance price, (ii) storage price and (iii) data transfer price, which are represented in XML syntax as:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:environmentDescription name="http://amazon.com/europeEC2Clouds"
xmlns:d="http://schema.arjuna.com/vddl">
```

16


The above few lines are part of a very complex cloud description document, available in A.1.9 in appendix, that displays the XML syntax starting with the established XML standard notation "<?xml version="1.0" encoding="UTF-8"?>. The name=http://amazon.com/europeEC2Clouds is the more important scheme, which represents the location where the specific description of cloud can be accessed from. The line xmlns:d=http://schema.arjuna.com/vddl refers that the description was designed in VDDL and name="dublinEC2Cloud" describes the pricing of a specific resource, which represents a geographical location of cloud infrastructure. Similarly, information of infrastructure resources such as instance, storage and data transfer is represented as attribute name="smallInstancePrice", attribute name="storagePrice" and attribute name="interMachineOut". The actual price is presented by value="1.0". More details mechanism of finding the features and reason for the specific feature selection would be discussed from section 4.2.1.

4.3.1 Collection of Cost Factors

It was evident from current IaaS provider’s pricing model evaluation [consolidated research result in section 2.10.1] that infrastructure resources are priced using the criteria of: (i) what is consumed, (ii) what is stored and (iii) what is transferred over the network. Therefore, answers to these questions gave the model such cost factors as (i) resource is consumed, (ii) information is stored and (iii) data is transferred over the network. Charging policies provided by the IaaS provider specifies the actual price as policy for example, according to Amazon WS’s resources are charged [9] for (i) virtual machine – using per-hour-per-instance pricing (ii) persistent data per-TB pricing and (iii) bandwidth – per-GB data transfer pricing. Cloud Pricing model for a cloud description in this dissertation therefore was built considering Amazon WS’s current pricing model [9]. Cost factors to be described in a cloud description were captured for: (i) size of required instances for the required service, (ii) amount of incoming and outgoing data and (iii) amount of storage used/allocated (e.g. Disk Space for persistent storage).

4.3.2 Design

All the cost estimation knowledge required by the broker, in particular how the infrastructure resources are charged - was captured in an Interaction template (that describes the detail break down and the association for a Resource). This knowledge includes information about the IaaS provider’s pricing policies (that is, service usage charge). The assumption for the cloud pricing policy for data transfer in different locations was that each of the business units within the IaaS provider are zone that run within a region. A method was defined to capture what exactly each of these resources can do and how they can be deployed. A type was defined for each of these resources that provide required information. The following diagram shows deconstruction of the pricing units of IaaS provider’s policy into their separate constructs:

---

Figure 4.2: XML Syntax for cloud description

---
Cost Estimation of Service Delivery in Cloud Computing, at Arjuna Technologies Ltd.

In the diagram the pricing units were decomposed from the IaaS provider’s pricing categories into three separate resources. At the root of the tree there is the IaaS provider’s pricing resource which exhibits the information of an individual IaaS provider’s pricing policy resource (VDDL feature) type. Secondly there is a Region that exhibits the specific pricing of a Region Type. Finally there is a zone which maps to a Zone Type that describes the prices applied for the resources exhibit the policy of a cloud zone pricing. For example, "Dublin EC2 Cloud" represents Amazon WS’s the most granular pricing unit, which therefore becomes the end node in the tree following “Amazon WS’s Pricing” as the root of the tree. More details are elaborated in the next section.

4.3.3 Implementation

Cloud Description was designed on the basis of modeling a Resource and a Resource Type. A Type presents a definition of a resource capability, which contains a mapping between itself to an instance of an actual description.

Standard notation was created using the type description feature from VDDL. For example, a resource used in an environmentDescription, available in A.1.9 in appendix, as mentioned below:

```xml
<d:environmentDescription name="http://amazon.com/europeEC2Clouds"
xmlns:d="http://schema.arjuna.com/vddl">
  <d:resource name="dublinEC2Cloud">
    ...
  </d:resource>
</d:environmentDescription>
```

Figure 4.4: XML syntax for DublinEC2Cloud resource

Creation of Standard Notation as Type

Implementation of cloud description initiated by creating machine type, storage type, network boundary type, application type, EC2Cloud type and S3Cloud type following the cloud description itself as europeEC2Clouds, asiaEC2Clouds and europeEC2Clouds, all description documents are available in Appendix A. A type was implemented to describe a network boundary for a resource (i.e. machine). The type refers to the location of that machine that provisioned in the cloud. The specification for the network boundary was specified using a resource named Zone and the <d:resource name="dublinEC2Cloud"> was defined by configurable attributes such as <d:attribute name="smallInstancePrice"> specifies the price for the size of machine DublinEC2Cloud offers. The attribute refers to the resource machine as <d:resource name="machine">. Type is used to reuse already existing components detailing its functionality for example, <d:resourceType name="ec2CloudPricingType"> describes pricing type which only applies to a resource of “dublinEC2Clouds”. For example, to estimate the machine cost for various products, the system requires the number of machines allocated for the deployment in each product. Here, machine is the common entity required in calculation for each software product description, therefore, can be reused in the computation. However, the typeDescription for cloud, available in A.1.6 in appendix, is described below:

```xml
<d:typesDescription name="http://amazon.com/ec2CloudTypes"
xmlns:d="http://schema.arjuna.com/vddl">
  ...
</d:typesDescription>
```

18
Cost Estimation of Service Delivery in Cloud Computing, at Arjuna Technologies Ltd.

Cost Factors in Cloud Description
The following parameters were considered to describe IaaS provider’s offering and to have abstract specification for a generic pricing model that can be applicable to all IaaS providers.

- `smallInstancePrice`: rent pricing for machine size
- `storagePrice`: for amount of static storage used by a deployed service
- `interMachineIn, interMachineOut, intraMachine`: assigning data transfer charge between machines.
- `intersS3StorageIn, intersS3StorageOut, intraS3Storage`: Data transfer “in” and “out” refers to transfer into and out of IaaS provider’s allocated environment (initially Region then next boundary Zone and finally provider’s Elastic Compute or Simple Storage environment).

Cloud description template was designed on the basis of following standard notation for measuring the cost parameters:

- `perHourRentalPriceType`: basic charging unit for measuring machine rent
- `perMBPriceType`: basic charging unit for measuring data transfer cost
- `storageAmountPriceType`: basic unit charge for measuring storage amount cost
- `perGBPriceType`: basic unit charge for measuring data storage cost

4.4 Software Product Description
A Software Product Description document specifies the formal configuration description of the software product that is required by the cloud user. Pre-deployment cost estimation of particular software product is derived from the cost factors described in this document, which is authored by the software provider. The cost factors described in the document contains deployment infrastructure information such as software component configuration, data transfer amount between components, required storage etc. The assumption was that the deployment configuration details can only be collected from the software developer’s knowledge base. Factors for cost-aware applications are therefore captured by locally or remotely running the software at developer’s site. Collected cost factors from the test can later be refined to bring the estimation closer to accurate. Although more details will be elaborated later, a simple example for “Student Registration” software product description, available in A.1.11 in appendix, is represented in following XML syntax in figure 4.6:

```
<d:productDescription name="http://sap.com/studentRegistrationProduct2"
xmlns:d="http://schema.arjuna.com/vddl">
    <d:product name="studentRegistration">
        <d:attribute name="machineSize" value="small">
            <d:resource name="AuthenticationServer">
                ...
            </d:resource>
        </d:attribute>
    </d:product>
</d:productDescription>
```

Figure 4.6: Syntax for Software Configuration in a software product description

The root element `productDescription` describes a particular deployable software product, in which the sub elements (such as resource, attribute) describe the configurable cost factors of that software product.
4.4.1 **Design**

To create a Software Product Description for a cost-aware application, an understanding of (i) deployment configuration and (ii) components behaviours (captured by software developer from local or remote test) is required. Consequently, a standard notation is required to facilitate the creation of these descriptions.

**Creation of Standard Notation as Type**

Initial requirement to recursively structure a software product description was to design the necessary standard notation. Standard notion is a generic framework to describe a software product and designed using VDDL’s *Type* feature.

A *software product* represents the kind of service the software provides and it might have different descriptions in which each of these descriptions has different deployment configuration details. For example,

![Diagram](image)

**Figure 4.7**: Example of Student Registration Standard Framework for Type description

In figure 4.7, the software product type (Student Registration - SRApp) provides a Student Registration Service. A description document describes how a service is provided over the cloud environment. For example, SRApp can be deployed with 1 large and 2 small machines or with 1 medium and 1 large machines. The cost estimation will vary depending on the differences in the different descriptions of the software product type. However, a generic model of the software product description can be created from a generic software component structure elaborated in the later section.

4.4.2 **Implementation**

*Software Product Description* was designed on the basis of defining *Resource* and *Resource Type*. A *Type* presents a pre-definition of a resource for its capabilities, which contains a mapping between itself to an instance of an actual description.

Standard notation was created using the *type description* feature from VDDL. For example, a resource used in a *productDescription*, available in A.1.10 in appendix, is mentioned below:

```xml
<d:productDescription name="http://sap.com/studentRegistrationProduct1"
xmlns:d="http://schema.arjuna.com/vddl">
  <d:product name="studentRegistrationProduct">
    <d:resource name="dublinEC2Cloud">
      ...
    </d:resource>
  </d:environmentDescription>
</d:productDescription>
```

**Figure 4.8**: XML syntax for DublinEC2Cloud resource

**Cost Factors in Software Product Description**

The following parameters were considered to describe software product provider’s offering and to have abstract specification for a generic service deployment model that can be applicable to all specification of a single software product description.

- **networkBoundary**: geographical location of the virtual machine provisioning
Software Product description template was designed on the basis of following standard notation for measuring the cost parameters:

- networkBoundaryType: basic charging unit for different variants of boundary cost
- machineSizeType: basic charging unit for measuring machine size cost
- meteredPortType: basic unit charge for measuring data transferred amount cost

Software Product Description Model

A typical software product description can be designed in two sections, (i) software components and (ii) interaction between the components. For example the following table represents the components requirements for Student Registration software that can be deployed in Amazon’s EC2 Environment.

**Table 4.1 Table: Representation of an Application Deployment Description on EC2 environment**

<table>
<thead>
<tr>
<th>Component Types</th>
<th>Between</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Components</td>
<td></td>
</tr>
<tr>
<td>Application Server (web app1)</td>
<td>Authentication Server (web app2)</td>
</tr>
<tr>
<td>Student Database (DB1)</td>
<td>User Database (DB2)</td>
</tr>
<tr>
<td>Interaction between Components</td>
<td></td>
</tr>
<tr>
<td>Application Server</td>
<td>and</td>
</tr>
<tr>
<td>Authentication Server</td>
<td></td>
</tr>
<tr>
<td>User Database (DB2)</td>
<td></td>
</tr>
</tbody>
</table>

Software components are represented as Resource and Connector was used to represent interaction medium for those components. Defined software components of a software product are then allocated for deployment infrastructure. The deployment requirements are (i) machine (virtual instance), (ii) domain. Machine is where a system component will be deployed and domain represents where in the Cloud the machine is allocated. The diagram in figure 4.6 is used to explain a typical example of a software product description, which is designed for Amazon’s EC2 cloud environment [9]:

Before going into functional behaviour a few design choices need to be addressed as described below:

- **Service Connection** – is the communication medium between a software product description and a service. The design supports different services offering through a single product. For example, Product1 provided by SAP offers two different services such as (i) Student Registration and (ii) Student Login Service.
- **Boundary** – to represent the outer range of a geographical location, particularly the domain that is passed via network traffic.
- **Domain** – is the provisioning of the virtual instance.
- **Software Component** – is the basic functional unit of a system.
- **Connector** – is the communication medium.
- **Connection point** – integrates network traffic communication.
In the figure 5.6, machine1 and machine 2 are virtual instances. In machine1 two components of “web app1” & “DB1” and in machine2 another two components of “web app2” & “DB2” will be deployed. VDDL representation of this structure, available in A.1.11 appendix, is as follows:

```xml
<d:productDescription name="http://sap.com/studentRegistrationProduct2"
    xmlns:d="http://schema.arjuna.com/vddl">
    <d:product name="studentRegistrationProduct">
        <d:resource name="machine1">
            <d:type name="type">
                ...
            </d:type>
            <d:resource name="webapp1">
                ...
            </d:resource>
            <d:resource name="db1">
                ...
            </d:resource>
        </d:resource>
        <d:resource name="machine2">
            <d:resource name="webapp2">
                ...
            </d:resource>
            <d:resource name="db2">
                ...
            </d:resource>
        </d:resource>
        <d:connector name="studentDBConnector">
            <d:type name="type">
                ...
            </d:type>
        </d:connector>
        <d:connector name="authenticationConnector">
        </d:connector>
        <d:connector name="userDBConnector">
        </d:connector>
    </d:product>
</d:productDescription>
```

Figure 4.10: VDDL representation of Software Product Description (using 4 software components)
In figure 4.10 the XML syntax represents an actual software product description in VDDL. A functional unit of the software is represented as a resource along with the machine that the components will be deployed on. Each of the resources belong to a type, which describes the component behaviour, such as `<d:resource name="webapp1">`, is described as application type using `<d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentWebApp1Type">` which resides inside another resource that represents the virtual machine as `<d:resource name="machine1">`.

In Amazon’s S3 environment

A similar approach is used to design a software product using S3 environment. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web [41]. The following table represents the persistent storage requirements for the same (mentioned in EC2 environment) Student Registration Service.

<table>
<thead>
<tr>
<th>Component Types</th>
<th>Interaction between Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Components</td>
<td>Between</td>
</tr>
<tr>
<td>Application Server</td>
<td>Blob1</td>
</tr>
<tr>
<td>Authentication Server</td>
<td>Blob2</td>
</tr>
<tr>
<td>Student Database</td>
<td>Blob3</td>
</tr>
<tr>
<td>User Database</td>
<td>Blob4</td>
</tr>
<tr>
<td>Machine 1</td>
<td></td>
</tr>
<tr>
<td>Machine 2</td>
<td></td>
</tr>
<tr>
<td>Machine 3</td>
<td></td>
</tr>
<tr>
<td>Machine 4</td>
<td></td>
</tr>
</tbody>
</table>

The deployment requirements are (i) machine (virtual instance), (ii) domain and (iii) S3 storage. This feature was designed using Windows Azure Blob [40] storage example. The following diagram is used to explain a typical example of a Product for S3 environment.
In the figure 4.12, the software product is designed especially for S3 Storage. Storage is a service provided by the IaaS provider to store large amounts of unstructured data in a blob that can be accessed from anywhere in the world via HTTP or HTTPS. Blob is a container for storing data [40]. Here blob was used to represent the internal application storage.

4.5 User Requirements Description

A User Requirements Description document describes the structure of a service requested by a cloud user. The document specifies required service information, for example (i) the type of service, (ii) duration and (iii) amount of data input for the service. However, the document supports augmenting more sophisticated service-related information for aiding more accurate cost estimation. For example, requirement for a Payroll Service may require number of users to be assigned.

Every new service (e.g. Payroll, Scientific Analysis) that needs to be deployed requires working out the vital information for deployment. In this dissertation the particular example chosen for a user requirements description uses three parameters (i) “Service Type”, (ii) “Duration” and (iii) “Input Data” which impact the cost estimation. However, this dissertation provides the flexibility to construct new templates for different applications and their attributes. The decision has been purely based on a simplistic view of the proof of concept that can be refined further to provide more cost factors to the cloud user. That means the accuracy depends on those cost factors and the more information can be collected from the cloud user to provide more accurate cost estimation.

An example of another factor could be how many number of users will be using the system and how fast user might be entering data into the system depending on how fast users type. In this case an attribute for “Data Input Amount” would have impact on the data traffic in proportionate to user’s “Typing Speed”. This is how User Requirement attribute specific to application can be derived into the model. The ideal user interface could be that user is presented with a drop down option of available applications to be run on the cloud and depending on the service selection the User Requirement form would change to collect more information from the cloud user.

4.5.1 Design

User Requirements Description specifies service request information for cost estimation. An understanding of (i) type of service and (ii) service capabilities are essential requirement for designing a User Requirements Description. Consequently, a standard notation is required to facilitate the creation of these descriptions.
User Requirements Description document describes cloud user’s specific service need, which can be modelled in a generic manner. Presumably, all possible cloud users would identify what service they require and for how long the service may require for. These two factors highlighting service usage pattern were considered as the minimum required information to model a generic framework for a User Requirements template. Along with two factors cloud users are believed to be known about the rough approximated data input amount, which therefore is considered as another cost factor. The following syntax shows an example of a standard user requirements description, available in A.1.2 appendix, type:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://ieee.org/useModelTypes"
 xmlns:d="http://schema.arjuna.com/vddl">
  <d:attributeType name="serviceDurationType"/>
  <d:attributeType name="serviceOperationType"/>
  <d:attributeType name="dataInputAmountType"/>
  <d:resourceType name="costConstraintType">
    <d:attributeType name="serviceDuration" ref="#serviceDurationType"/>
    <d:attributeType name="serviceOperation" ref="#serviceOperationType"/>
    <d:attributeType name="dataInputAmount" ref="#dataInputAmountType"/>
  </d:resourceType>
</d:typesDescription>
```

**Figure 4.13: Typical example of a User Requirements Description**

The type description for User Requirements Description is presented in `<d:typesDescription>` element. The document represent a standard notation for the cost factors as `<d:attributeType>`s. The common cost factors are represented as a `resource type` naming as `costConstraintType`, which will facilitate adding more constraints under the same category.

### 4.6 Description Document Summary

Having designed and implemented the description of software products and cloud on the basis of standard notations as `type description`, a cost model to be designed to utilise these descriptions and provide cost estimations by analysing the cost factors specified in description documents. The cost model creation is described in the next chapter.
5 Cost Estimation

Given the scope in section 1.4 of cost estimation model some assumptions have been made based on the analysis of IaaS providers pricing models in section 2.10.1. This chapter introduces Cost Estimation, discusses the design of the Cost Estimation Model (CEM) in different perspectives and presents all the base arguments for the assumptions made for each individual decision.

5.1 Cost and Estimation

Cost is the amount paid or payable for the acquisition of materials or services [38]. Cost of service therefore is measured by the resources used to attain it. Money cost is not necessarily the same as economic cost. "Economic cost" implies the use of resources – virtual machines, storage, etc. Currencies (such as USD/GBP) are used merely as a convenient common denominator for aggregating numerous heterogeneous physical quantities into meaningful "packages" for purposes of analysis and decision making. The art of approximating the probable worth of acquisition of materials or services on the hand is called cost estimation [38]. There are two major categories of cost estimation methods: algorithmic and non-algorithmic. Algorithmic models vary widely in mathematical sophistication. Some are based on simple arithmetic formulas using such summary statistics as means and standard deviations [9]. Others are based on more sophisticated mathematic model line [38] and differential equations [30]. To improve the accuracy of algorithmic models, one needs to adjust or calibrate the model to specific circumstances. On the other hand non-algorithmic models vary on analysis mechanism. An overview of algorithmic and non-algorithmic methods is presented below:

5.1.1 Non-algorithmic Methods

A non-algorithmic method is a comparison and inference analysis mechanism [42]. In order to use the Non Algorithmic methods, some information about the previous similar project’s estimation is required and usually estimation process in these methods is done according to the analysis of the previous datasets. Here are three example methods are discussed to give an insight to non-algorithmic methods [43, 44]. Analogy costing requires one or more completed projects that are similar to the new project and derives the estimation through reasoning by analogy using the actual costs of previous projects. Bottom up approach an estimate for the overall system is produced from the aggregated result of the estimation of each component of the software system. The requirement for this approach is that an initial design must be in place that indicates how the system is decomposed into different components. Top down approach is the opposite of the bottom-up method. An overall cost estimation for the system is derived from global properties. The total cost is then split up among the various components.

5.1.2 Algorithmic methods

An algorithmic model (also known as parametric model) produces a cost estimate using one or more mathematical algorithms and a number of variables considered to be the major cost drivers [42]. These models estimate effort or cost based primarily on the Hardware/Software size, and other productivity factors known as cost driver attributes. The algorithmic methods are based on mathematical models that produce cost estimation as a function of a number of variables, which are considered to be the major cost factors. Algorithmic models follow the following structure:

\[ e = f(x_1, x_2, ..., x_n) \]

Where, \( e \) denotes an effort, \{\( x_1, x_2, ..., x_n \}\} denote the cost factors. The cost calculation in existing algorithmic methods differs in two aspects (i) selection of cost factors, and (ii) form of the function \( f() \). Algorithmic models rely on the identification and quantification the cost factors. It is evident that the most comprehensive set of cost factors are proposed and used by Boehm et al in the COCOMO II model [6]. These cost factors can be divided into four types, including computer factors (used in this dissertation) of execution time constraint; main storage constraint; computer turnaround constraints; platform volatility.

Investigating two types of cost estimation methods has provided a strong platform to model the suggested solution for this thesis. The cost model construction and requirements are discussed in the later sections.
5.2 Cost Estimation Process

Cost estimation process is to determine the chargeable factors for running a service (such as Infrastructure as a Service). There were two major aspects for the cost estimation and they are (i) initial cost and (ii) on-going cost. An initial cost is the deployment cost for certain components. For example there could be a requirement for a certain amount of data (100 GB) uploaded at initial deployment. Once they are uploaded then data upload amount may decrease drastically. Hence cost incurred for transferred data will only be included in initial deployment cost. On-going cost comes from the instances that are already deployed and running, this refers to usage deployment cost [30]. Cost estimation for initial deployment and on-going will vary. To give more simplistic view, initial deployment cost has been considered for this dissertation.

Structured description documents for cloud, software product and user requirements are the source for cost estimation process. Upon defining the documents, the information needs to be utilised to build the model itself. Prior to building the cost model, the major features for the model required to be specified and they are (i) cost type and (ii) cost factors. Details of these two features will be discussed in later sections.

The methodologies to define a cost estimation model, the following steps were followed:

- Analyse required service and relevant factors
- Identify the chargeable factors and for each one, identify the units that would be applicable to the service.
- Apply pricing strategy provided in the Cloud description
- Identify, integrate and apply the cost model for an estimate cost.

These steps are illustrated in figure 5.1.

![Cost Estimation Methodology](image)

**Figure 5.1: Cost Estimation Methodology**

Cost Factors Design Decisions

Cost models are fundamentally used on chronological data that is different than computer-based simulations. The model required to be built using a logical process based on objective evaluation of the path between the invention and the outcome. Thus, the logical process followed in cost modelling is elaborated as follows:
As mentioned in the figure 5.1 cost estimation methodologies requires defining the chargeable factors and atomic units for three categories of infrastructure *resource* pricing that are instance, network bandwidth and storage.

- **Resource Charging Unit**: The various commercial offerings differ in price, where a single IaaS provider may choose to provide a flat rate billing model, e.g. per user charge or per month charge. It is evident from IaaS provider’s offering evaluation discussed in section 2.10.1, that instance usage is charged at per-hour basis, input/output data traffic is charged at per-gigabyte usage basis and storage is charged at per-terabyte. However, due to the complexity of multiple providers, a generic charging unit required being decided and entire duration of a software product deployment is considered for cost calculation for network traffic. The choice is to facilitate the fact that at the beginning of running a service cloud user might require uploading large amount of data. Large amount of data may not get generated over the duration, therefore charging per hour policy for inbound and outbound data transfer did not seem a good option.

- **Local Storage vs. Additional Storage**: The assumption is that the software product *description* documents describe internal communication of software components in proportion to user input. The purpose of providing a system that reduces the burden for software product expertise, where cloud users may not be aware certain software specific information. For example, internal components of a software product can be dealing with 1TB of data dynamically at run time, which cannot be stored in the local storage provided by virtual machine, which assisted the need of acquiring additional storage.

- **Single pricing band for additional storage**: In practical cloud users might be requiring dynamic deployment within an existing deployment. For example, cloud user running a Payroll service may require deploying data analysis component only at the end of the month on an existing deployed service. The process refers that the deployment is dealing with existing cloud user’s accounts which has previously used standard instance, storage and bandwidth allowance and charging rate provisioned by IaaS provider. However, additional deployment can easily fall into an unrealistic price band. In favour with simplification of the cost model a single price band is therefore considered. However, the model is designed to support different pricing band according to providers pricing model.

- **Network Boundary**: While investigating IaaS pricing, it came to light that different pricing models are applied to different resources allocated in different locations. Bandwidth calculation depends on what traffic comes into the service, what traffic is generated and distributed within the service. If network boundary is generated then count of incoming and outgoing traffic becomes a rational choice. To provide more abstract design for bandwidth calculations, notion of network boundaries has been chosen.

- **Local IP vs. Elastic IP**: An Elastic IP address is a static IP address that is designed for dynamic cloud computing. Elastic IP addresses allow masking virtual machines or availability zone failures by programatically remapping the public IP addresses to any instance associated with the account. A cost factor described by Amazon WS charges for additional Elastic IPs associated with any particular virtual instance [9]. Consequently an additional charge is applied to the total estimated cost. For simplistic reason the assumption was that the IP is accessed via local IP address in this dissertation.

- **Data transfer count**: To calculate network bandwidth cost for particular service research evidence in chapter 2 highlights the fact “how much data transferred (in and out) between resources” can be monitored by a resource connection point. The approach inspired from peer-to-peer interaction pattern will allow the model to be generic to possible cloud descriptions.

![Data transfer between sender and receiver](image)
5.3 Defining Cost Structure

Identification of cost types have been initially created on the basis of real Software product configuration information that requires to be deployed on the cloud infrastructure. The identification approach follows a typical decision-making process starting with a strategic decision to source Service requirement and ending with the Cloud Infrastructure Service [45]. An overview of the different identified cost types, corresponding cost factors, which are unique by item or cost type and the single phases of the decision-making process, is presented in the following table 5.1.

Table 5.1: Cost Type and description

<table>
<thead>
<tr>
<th>Cost Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Cost</td>
<td>Total cost of virtual machines described for a software components deployment</td>
</tr>
<tr>
<td>Network Bandwidth Cost</td>
<td>Total cost of data transferred from and to a deployed service</td>
</tr>
<tr>
<td>Storage Cost</td>
<td>Total cost of storage allocated for a service</td>
</tr>
</tbody>
</table>

5.4 Description of the Cost Model

A Cost model is an estimating tool consisting of one or more cost estimation relationships, estimation methodologies, or techniques used to predict the cost of a system or one of its lower level elements [38]. To present a transparent cost model this section highlights and elaborates a description of the general model design. The aim is firstly to assign cost factors that influence the cost types and then present the general underlying formula design that is applied for each cost type. The assignment of the cost factors to the identified cost types is represented in table 5.2.

Table 5.2: Categorised factors for cost estimation.

<table>
<thead>
<tr>
<th>Cost Type</th>
<th>Cost Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Cost (ins)</td>
<td>service operation(so), duration(duration), machine type(vmtyp), machine size(vmsiz), number of amount(numm)</td>
</tr>
<tr>
<td>Network Bandwidth Cost(bnd)</td>
<td>number of virtual machines(nvm), type of connector(con), number of connectors(ncon), type of port(po), number of ports(npo), data in (din), data out (dout), intra(dint)</td>
</tr>
<tr>
<td>Storage Cost(sto)</td>
<td>duration(duration), data input amount (dinp)</td>
</tr>
</tbody>
</table>

5.5 Cost Model

For the proposed cost model the assumption is that the pre-deployment total cost of a service equals the summation of all cost types: \( TC_{ser} = \sum C_f^{ct} \), where the total amount of a cost type \( ct \) (shown in the exponent) equals the sum total of all involved cost factors \( f \) (shown in the index) such as\( \sum C_f^{ct} \). The cost model considers the duration as the complete period of time during which a Cloud Computing Service is going to be used respectively. As a general rule, this period is considered as hourly basis, hence the usage time \( T_{usage} \) of a service is calculated as \( T_{usage} = T_{duration} \times 24 \), where the minimum duration is in day basis.
**Instance Cost** (ins): Instance cost depends on the price of each virtual machine and duration of usage. The mathematical model for the total instance cost \( C_{ins} \):

\[ C_{ins} = N_{\text{machine size}} \times T_{usage} \times \text{Pr}_{\text{instance}} \]

**Equation 5.1: Instance Cost calculation**

The calculation factors including instance price \( \text{Pr}_{\text{instance}} \) is extracted from cloud description document, number of machine of size \( N_{\text{machine size}} \) is extracted from software product description document, and the usage time \( T_{usage} \) is calculated from the factor duration that is described in the user requirements description document. Instance price is associated different instance size (such as small, medium and large).

**Network Bandwidth Cost** (bnd): Calculation of network bandwidth cost depends on the amount of data transferred from and to ports applying total input and output boundary price on that transferred data for a single component. Boundary price is extracted from cloud description document where each boundary (such as Region and Zone) is mapped to a particular price. The calculation also considers a cost factor of total data input amount into the system which is extracted from a user requirements description document. The equation that follows the calculation for total network bandwidth cost is as follows:

\[
TC_{\text{bnd}} = D_{\text{out}} \times D_{\text{out}}^{\text{price}} + D_{\text{in}}^{\text{price}} + D_{\text{from}}^{\text{price}} + D_{\text{to}}^{\text{price}}
\]

**Equation 5.2: Network Bandwidth Cost**

**Storage Cost** (sto): The storage cost depends on total storage amount and associated price for the storage. Storage amount is extracted from the software product description document and the storage price is extracted from the cloud description document. The equation that follows the calculation for total storage cost is as follows:

\[ C_{sto} = T_{\text{amou}} \times \text{S}_{\text{price}} \]

**Equation 5.3: Storage Cost**

Storage capacity is described in terabyte for the cost model, however amount of data inbound and outbound is calculated in gigabyte, therefore total storage amount is converted into gigabyte for the storage cost. Considering the cost factors as storage capacity \( S_{dc} \), the total storage amount \( T_{\text{amou}} \) is determined as \( T_{\text{amou}} = S_{dc} \div 1024 \).

**Total Cost Estimation**: of running a service for a software product on the cloud infrastructure can be calculated by adding total cost of instance, network bandwidth and storage and total cost (total):

\[ TC_{\text{ser}} = TC_{\text{ins}} + TC_{\text{bnd}} + TC_{\text{sto}} \]

**Equation 5.4: Total Estimated Cost for instance, network bandwidth and storage**
5.6 Cost Estimation Implementation

The cost estimation model was implemented in VDDL. This section elaborates the process that was used to implement the cost estimation model for particular software product description.

Instance Cost Model

In order to calculate the cost an instance is represented using resource. To describe resource’s features a type was created where the resource feature is implemented as an attribute. One of the type was “machine” (called instance). The size of the machine was defined in attribute as machine size and the value of that attribute was used to define the actual measuring unit of small, medium or large.

Network Traffic Model

To calculate network traffic cost, the system is designed to establish a connection between individual components. The following diagram shows how a connection can be represented in VDDL.

![Network Traffic Diagram](image)

**Figure 5.3: Network traffic communication between system components**

In the figure 5.3 “web ap1” represents an application server and “web app2” represents an authentication server. The white circle represents service need and dark circle represents service provide. The double arrow represents a data transfer medium between two components. An arrow towards “web app2” means data transfer from “web ap1” to “web app2” which is a request for service need. The arrow towards “web ap1” means data transfers from “web app2” to “web ap1”.

VDDL allows presenting the connection using a feature called Connector. Connector has two ports. Port is used to represent connection point. Calculating data produced from each port gives inbound and outbound data transfer count for an individual port. This design choice was made to introduce more abstract and high level charging scheme for network traffic. Once amount of data transferred between a connector is calculated then a boundary price can be applied. More details in later section.

Applying Boundary Price

It was necessary to have a mechanism to describe boundary to calculate network traffic cost. To facilitate this feature the following diagram explains how this has been designed.
Figure 5.4: Data transfer cost

In the figure 5.4 a system component “web app” is designed to be deployed in a machine in “Zone A”. Then again that zone belongs to another domain of “Region A”. On the other hand another system component of a database “Student DB” is designed to be deployed on a machine which is located in “Zone B” and that belongs to “Region B”. To apply boundary price to the network traffic passed between these two components the system would require to find the common domain first. Once the domain is found, which in this case “EC2 Cloud”, a path will be constructed to apply each boundary and its associated price to the cost estimation. Each path will be created for each port. In the diagram the white circle representing database service need creates the path of \{machine1, zone A, region A\}. That means three different prices will be applied for that single port. Same applies to the other port (i.e. dark circle, service provide). Although this is an over optimistic design, VDDL feature has supported to achieve the design and get a result for the proof of concept of this project.

Service data Input/Output

Connectors were used to calculate data transfer amount and cost between two components. This is possible because Connector is consists of two ports and those ports must be connected before calculation. The port that is used for the service connection point for a product is a non-connector port. Therefore, the calculation conducted for the total amount of data input through the system and then total amount of data output through the service operation port. This input data amount therefore would be expected from cloud user. Initially cloud user has to define how much data would be input through the service access point.

Figure 5.5: Service access point in a product

In the figure 5.5 the double arrow with dotted line represents a communication between the entire service and the actual operation. The arrow going inward (towards service operation port) means a service request and data communication. The arrow going outward (towards non connector port) represent amount of data that will be produced by all system components. While designing the application developers would be expected to provide the required information. The assumption is that the application developer would have more knowledge about system components interaction and amount of data travels between components.

Storage Cost Model

Similarly a storage container was implemented using resource. Standard notation for storage was created as storage type, which describes the capacity. The cost factor of storage amount for the cost estimation was described as an attribute and the value of the attribute was initialised to describe the actual amount.

5.7 Conclusion

The analysis of relevant cost factors of software product deployment is an important pillar of cost estimation mechanism. For initial generic cost modelling in this dissertation some restrictive assumptions were made that supported in taking a particular focus on cost estimation of service delivery.
6 Brokering Service

Having designed description documents and cost model, broker requires a one point service where all the artefacts are used to provide cost estimation as service to cloud users. This section therefore elaborates how the brokering system that provides the cost estimation service was designed and implemented for this dissertation.

6.1 System Architecture

As illustrated in figure 6.1, the brokering system consists of: a (i) web portal, a (ii) broker engine, and (iii) system users (actors). The web portal provides an interface to the brokering system, while the broker engine performs data (i.e. cost factors) processing, extraction, analysis and cost estimation. In order to provide cost estimation broker needs cost factors, which are described in structured descriptions. Consequently a (i) description inventory and (ii) cost estimation algorithm was built for the broker engine.

![Figure 6.1: High level architecture of Brokering System](image)

The system users were categorised as (i) cloud user and (ii) admin. Cloud user accesses the web portal user interface and requests for service cost estimation by filling up a custom web form and admin holds the
administrative rights to upload the description documents the web portal. The reason for designing a web based application was to provide cloud users a remote access to the system. The Detailed system components’ description is discussed in the following section.

6.2 System Components

Web portal and Broker engine components were designed to provide system operations.

Broker Engine: consists of two sub-components: (i) Description Inventory (ii) Cost Estimation Algorithm

i. Description Inventory: the inventory is a component of the virtual deployment description language (VDDL) runtime platform. The description inventory in this dissertation is considered as a non-relational database for various structured descriptions, which are all necessary information for computation.

![Description Inventory Diagram]

Figure 6.2: Description Documents Inventory with sample documents (TD: type description, SPD: software product description, CD: cloud description, URD: user requirements description)

The description inventory holds the extracted VDDL objects and their associated values from the XML files, which are later accessed by the cost estimation algorithm. TD1 in the figure.6.2 refers to the cost factors (e.g. machine size) associated with a type description, which are extracted from the first type description document (e.g. MachineType.xml). Similarly, TD n refers to the factors extracted from the nth type description document. Similar applies to SPD (software product description) and CD (cloud description)

- Descriptions: are the cost factors that are described in the description documents using VDDL syntax.
- Description documents: are XML documents created in virtual deployment description language. Such descriptions described in chapter 4, include type description (i.e. standard notation to define a description document), environment description (i.e. IaaS provider’s pricing information for infrastructure resource), product description (e.g. software product description for Student Registration) and deployment description (e.g. User Requirements for a particular service).

ii. Cost Estimation Algorithm: this component performs produce a cost estimate using one or more mathematical algorithms. A number of variables considered to be the major cost drivers and the model estimates the cost based primarily on the resource factors known as cost driver attributes.

6.3 Brokering System Work Flow

The brokering system contains four processing units for the brokering service. Each of the units is associated with independent task of uploading, resolving and processing XML files and estimating the cost. Before running the system, the type descriptions are added to the brokering system and the description inventory is created by the broker. Then the workflow of the cost estimation brokering service proceeds as shown in figure 6.3.

34
Uploading Description Documents: This task allows the system admin to upload software product and cloud description documents (as XML files) on the server at any point of time. The process allows admin locally via the web portal to select and upload the XML files. A URL is used in a typical description document for associating the documents to the standard notations, which can be processed by the descriptions resolve processing unit. For example, a description document contains a URL (document identifier) for cloud description of amazon’s EC2 cloud. When the description processing unit needs to scan all the descriptions into the inventory this URL definition facilitates to detect the specific required description among many.

Processing XML Files: having the XML files available on the server this task parse the XML descriptions and store as an inputstream. If the XML documents are described using incorrect VDDL notations then the unit feeds back error as a system problem.

Descriptions Resolve: after adding the descriptions to the inventory this task performs a description document’s dependency check. Cost estimation process requires cloud and software product structure description which are generated following standard notations (type descriptions). These standard notations require to be added to the brokering system prior to estimation. Where the descriptions from the documents are extracted and to a particular the description inventory. The system is designed in a manner that a missing type description document for a cloud or software product description will throw an error in the description resolve stage. The system admin will be notified in case any discrepancies found.

Descriptions Process: this task processes the dynamic descriptions (VDDL objects), which facilitates broker engine extracting the value associated with specific cost factors described in these documents. The procedure specifically processes the information defined using deployment description language and make them ready for the algorithm to use for cost calculation.

Cost Estimation: process starts when a cloud user fills up a web form with a required service and request for cost estimation. Consequently user requirement descriptions are generated by the system and added to the description inventory. Immediately after the process finishes cost estimation algorithm triggers estimation process, which extracts cost factors from the description inventory, and then the numerical values associated with the cost factors are mathematically processed by the equations discussed in chapter 5 from the cost model and generate cost estimation results. The process starts by analysing all available software product descriptions in the inventory, selects the descriptions that matches the user requested service descriptions, discards all other available software product descriptions. Then the algorithm checks, which description document conforms to available cloud pricing policies in the cloud description documents. Having description documents comparisons the algorithm mathematically processes cost factors and presents approximated cost for a single software product via the web form.

6.4 Cost Estimation Algorithm

This section discusses how the author uses the cost model discussed in chapter 5, to estimate cost.

Pseudo Code

The pseudo code presented below demonstrates the logic of the Cost Calculation Algorithm. Set of cloud, product and user requirements are the information added in inventories (dynamicDescriptionInventory) which are extracted from an XML document.

Algorithm: CostEstimation
Cost Estimation of Service Delivery in Cloud Computing
at Arjuna Technologies Ltd.

**Inputs:**
- userReqList: Set of User Requirements,
- cloudList: Set of Cloud Descriptions,
- productList: Set of Product Descriptions,
- instanceList: Set of Instances,
- networkTrafficList: Set of NetworkTraffic,
- storageList: Set of Storages,

**Variables:**
- totalInstanceCost: float
- totalNetworkTrafficCost: float
- totalStorageCost: float
- userReq: Resource
- cloud: Resource
- product: Resource

**Returns:**
- totalCost: Estimated Total Cost for a Product

```plaintext
Begin function CostEstimation ()

// Calculate total cost for each product against list of clouds.
for (every cloud in the cloudList)
{
    Get product_list
    for (every product in the productList)
    {
        Calculate
            totalInstanceCost for a product (product, cloud, userReq);
            totalNetworkTrafficCost for a product (product, cloud, userReq);
            totalStorageCost for a product (product, cloud);
        
        totalCost = (totalInstanceCost + totalNetworkTrafficCost + totalStorageCost);
    }
}

return totalCost;
End
```

A total cost comparison algorithm was designed where a list of estimated total costs for each product are compared and presented in a descending order. The process allows cloud users to see the cheapest cost among the list. The pseudocode for the algorithm is presented below:

**Algorithm:** CostComparison

**Inputs:**
- costEstiList: Set of totalCost,

**Variables:**
- totalCost1: float
- totalCost2: float
- comparedCost: float

**Returns:**
- comparedCost: Estimated Total Cost for a Product

```plaintext
Begin function CostComparison ()

// Compare total cost for each product and sort in descending order.
{
    Get (totalCost1 and totalCost2 from costEstiList)
    comparedCost = Compare totalCost2 to totalCost1
}

Sort costEstiList

return comparedCost;
End
```
Instance Cost Algorithm

Pseudo Code

The pseudo code presented below demonstrates the logic of the total instance cost Algorithm.

Algorithm: calculateInstanceCost

Inputs:
- product : Resource,
- cloud : Resource,
- duration : float,
- smallInstancePrice : float,
- mediumInstancePrice : float,
- largeInstancePrice : float,

Variables:
- cost : float
- usageTime : float
- machineSize : String

Returns: cost : Estimated Total Cost for Instances

Begin

function calculateInstanceCost(product, cloud, duration, smallInstancePrice, mediumInstancePrice, largeInstancePrice)
{
    for machineSize get SetOfMachineViaType(product)
    Calculate cost += count MachinesOfSize(product, machineSize) * usageTime * getInstancePrice(cloud, machineSize);
}
return cost;
End

To calculate network bandwidth multiple algorithms were created. For example,
- Calculating traffic cost between components
- Calculating traffic cost for connectors that connects a service to a deployment template
- Calculating traffic cost for connectors that connects software components
- Calculating traffic cost for the port that represents a service
- Calculating traffic cost for the connectors within a software product description

Traffic Cost Algorithm

Pseudo Code

Algorithm: getTrafficCost

Inputs:
- trafficPath1 : List of networkBoundaries,
- trafficPath2 : List of networkBoundaries,
- dataProducedFromPort1 : float,
- dataProducedFromPort2 : float,
- rootBoundaryType : String,
- boundaryPriceMap : Set of String and its BoundaryPrice

Variables:
- boundaryType : String,
- totalCost1 : float,
- totalCost2 : float,
- comparedCost : float,
totalCost : float,
boundaryTypePrice : BoundaryPrice,

Returns: totalCost : Estimated Total Cost for a Product

Begin
function getTrafficCost ()

// calculate traffic cost between two components.
{
  for (every boundaryType in trafficPath1)
  {
    boundaryTypePrice = Get Price for boundaryType;
    if (boundaryTypePrice is found)
    {
      totalCost += boundaryTypePrice + dataProducedFromPort1;
      totalCost += boundaryTypePrice + dataProducedFromPort2;
    }
  }
  for (every boundaryType in trafficPath2)
  {
    boundaryTypePrice = Get Price for boundaryType;
    if (boundaryTypePrice is found)
    {
      totalCost += boundaryTypePrice + dataProducedFromPort2;
      totalCost += boundaryTypePrice + dataProducedFromPort1;
    }
  }
  rootBoundaryTypePrice = Get Price for rootBoundaryType;
  if (boundaryTypePrice is found)
  {
    totalCost += (dataProducedFromPort1+ dataProducedFromPort2) * rootBoundaryTypePrice;
  }

  return totalCost;
}
End
6.5 Brokering System’s Class Diagram

The UML class diagram in figure 6.4 presents the blueprints of the brokering system. The diagram models the classes that build the system and displays the relationships among those classes. The classes describe the characteristics and capabilities of the system objects. All the abstraction from low level VDDL processing is done by the classes with “ServiceEJB” suffix in class name (such as, ProductServiceEJB, CloudServiceEJB). The prefix “Cost” denotes the classes which are involved in the calculation. The suffix “Bean” denotes classes which provide presentation capability for the web portal. Classes with no association are used to represent a single object in the system, such as BoundaryPrice class only represents the price information. The diagram also shows the associations between classes, for example an arrow from ProductServiceEJB to DynamicDescriptionEJB denotes that DynamicDescriptionEJB is used by ProductServiceEJB.

![Class Diagram](image)

Figure 6.4: Brokering System represented in a class diagram

6.6 Design Assumptions and Decisions

The brokering system was architected assuming there would be rapid addition of new software product descriptions from various providers, for instance HP or IBM provides new descriptions for an existing software product (such as Content Management). New description documents can be added to the broker database in day to day basis by uploading the documents (in this dissertation XML files). Similarly new cloud pricing policy described in a cloud description document based on new standard notations can also be uploaded into the system by the admin.

The system was designed considering the fact that in the course of software product description change, the unique feature of the cost model stays intact. However, if the cloud pricing model changes or software providers changes the previously described specifications then the broker engine would require to be changed.

The brokering system has coped with two most radically changeable aspects, where one is software product descriptions and the other one is IaaS providers adding and removing pricing models from and to the broker.
database (i.e. inventory). These two computations for cost estimation can be done without any implementation changes. However, if a new service is added to the database then a new service request form will be customised to present to the cloud users. Consequently, new IaaS pricing models and software product descriptions from multiple providers would impact on estimation model change. The change therefore, confirms that in the event of new software product design, a developer would require new features to be specified. Incidentally, the process is facilitated by a “Standard Body” approving for the feature description. At that point IaaS providers need to get into that to bind the software product description and redesign their pricing model so that software can be accurately charged against and broker would therefore need to change its cost estimation algorithm to a different one that considers the new feature description.

The idea of brokering service is to be provided notionally by the third party intermediary companies, called as broker, but the software product or IaaS providers themselves will be discussed from this particular point. The system is designed considering that not only software product companies but also cloud users themselves can author a software product description and only needs to check cost estimation against available IaaS providers pricing policies. In view of the use case the suggested solution provides more abstract and generic model where software products can be used as a plugin and can be evaluated against cloud pricing.

Stating the broker, IaaS provider, software product provider and cloud user are independent system entities, a new scenario can be constructed to express the adaptability of the system. For instance, the software product provider wants to provide services along with software products then the provider may provide in-house brokering services only access to the provider’s own software products. This way the providers would own the brokering service and the software products at the same time. Only third party descriptions would require to be injected in the brokering service model which is IaaS provider’s descriptions. Also if the provider wants to capture some of the cost then they may have reserved internal pool of machines where preferable deployments can be made into the software provider’s own infrastructure. This way software provider can eliminate potential profit margin which external IaaS providers will be charging for using their infrastructure. Doing that would allow providers to retain their profit. In this case software product providers would only require going out and looking for more infrastructure resources if they run out of their own local resources. Therefore, the suggested solution is a general architecture which presents different dimensions if the system component of software product and broker belongs to same circle or vice versa. On the other hand when the components are owned by different stakeholders then the structure of the solution would be a different structure.

Client and Broker – One to Many: If the system allows one user requirement for cost estimation then individual brokers can offer composite prices. This is because IaaS providers may have business agreement between broker and IaaS providers. The agreement may include various deals resulting flexible pricing. This will introduce a competitive market for brokers to offer competitive cost estimation. Therefore client will have options to choose between brokers. This competitive approach would force to provide better improvement for the service.

All of the segments of this dissertation were designed following a generic model. This is because the generic model can be applied to many other required circumstances, especially where the other models are more complex and difficult to explore the details of it. Describing “something” can be referred to data transfer pattern, for instance data travelling in, out or within components of a system. This choice was made to provide more generic and flexible cost estimation.

6.7 Implementation

The implementation was completely based on the design outlined in the section 6.1 in this chapter. As such the basis for implementation was already in place. This section will display the completed implementation of the brokering system and explain the implementation process.
6.7.1 Development Tools & Languages

Brokering system components were developed at Arjuna with an Intel Core™ i7 processor and Windows® 7 Home Premium (64-bit). Eclipse Indigo Service Release 2 (Build id: 20120216-1857) was used as an integrated development environment (IDE).

JBOSS Tools 3.3.0 was used for a set of Eclipse Indigo plugins that supports JBoss and related technology (CDI, JBoss AS, JSF 2.0, (X)HTML, Seam, Maven) that were used to build the system [55]. The system was hosted on JBoss 7.1 Runtime Server (version: jboss-as-7.1.1.Final) [56] and was built as a Maven project in JbossTools project. Maven is a software project management and comprehension tool which uses project object model (POM) to manage project’s build, reporting, and documentation.

EJB 3.0 Specification the server-side component architecture for Java Platform, Enterprise Edition (Java EE 6) was used for the backend development of the brokering system. The reason to use Java was because VDDL only supports Java framework.

The web portal was implemented as an XHTML page using RichFaces technology, which provides an advanced UI component framework for easily integrating Ajax capabilities into business applications using JSF [48]. Java Server Faces technology is a server-side user interface component framework for Java technology-based web applications [49].

6.7.2 Implementation Process

Firstly, description documents (XML files) were created using VDDL. VDDL objects used in the XML files need to be processed and provide an abstraction between the low level VDDL logic and the high level cost estimation logic.

![Diagram of Brokering Service Oriented Architecture](image)

**Figure 6.5: Brokering Service Oriented Architecture**

Consequently cost estimation process was implemented as shown in figure. 6.5, following a Service Oriented Architecture (SOA) with enterprise beans, session beans and POJO. Each processing units in cost estimation process were implemented in Enterprise Java Beans (EJB 3.0) container. EJB was used to support scalability by allowing the system components to be deployed in multiple machines, to provide developers a freedom to build application using the cost estimation service, where the service can run on any compliant J2EE server. When the processing units were implemented as an individual enterprise bean, a graphical user interface was needed to make the service request and see the cost estimation result; hence the web portal was implemented. Separating the presentation layer allowed creating the web pages for each processing unit within brokering service. Each unit was implemented in a separate web page. Web pages were implemented to collect user inputs, the system output needed to be rendered to the XHTML and Managed Beans were created because, managed beans are used as model for UI components in JSF. JSF 2.0 uses annotations to register managed beans. With annotations, the bean and its registration are in the same place (Java class), which simplifies code management. Managed bean accesses enterprise data through the enterprise beans and specifies how that data should be presented. It maintains consistency in its presentation when the model changes. Push model was used where the bean registers itself with the model for change notifications and pull model was used for calling the model when it needs to retrieve the most current data. When the user request and system response logic was implemented the enterprise beans were exposed as stateless session beans, because no conversational state was required to be
maintained between browser and the server [50]. For example, cloud users access the portal, enter service requirement and request for cost estimation. When the system invokes the methods of a stateless bean, the bean’s instance variables may contain a state specific to that cloud user but only for the duration of the invocation. When the method is finished, the user-specific state needs not be retained. This decision was also taken because stateless session bean can implement a web service, which is a potential future work for this project [50]. At the point transferring calculated value from the data source the system needed to present the information in a simple class and therefore Plain Old Java Objects were created for each object (e.g. instance, network traffic, storage etc).

6.7.3 Source Code Structure

RichFaces project facilitated rapid development of the system by default structuring the source code and build process artefacts. The source code was structured into separate packages with the idea of separating the conceptual components as presented in figure 6.6.

![Brokering System Source Code Structure](image)

**Figure 6.6: Source Code packages**

Packages were created to separate the classes involved in the pre-calculation processing from the actual calculation process. controller package contains all the classes which executes, XML file uploading, processing, creating inventory. In contrast, the core system components were separated in four distinct packages (Cloud, Product, User Requirements, Cost), to facilitate addition, modification and maintenance of source code in each of these packages. For example making changes to user requirements description processing, VDDL object extraction or data rendering for web portal developer would only need changes in the corresponding package, which is userRequirement in this particular example. These packages contain the classes that read VDDL objects and extracts descriptions from the description inventory which are described in a description document. Description inventory access was implemented by injecting DynamicDescriptionInvEJB enterprise bean using @EJB notation, whereas @Singleton was used to implement DynamicDescriptionInvEJB.java class to provide a single instance of the inventory at the time of cost estimation. Auto addition of the type description documents to the system was implemented using the @PostConstruct, because the process required to be executed after dependency injection was done to perform the initialization of DynamicDescriptionInventory (VDDL object).

6.1.1 Web Portal Implementation

As mentioned in the system design section 6.1, admin requiring file upload, Richface’s “file upload” composite component helped to implement the feature. Admin can add single or multiple files to the component list and “upload” button uploads the XML files into the server. “Clear All” button was added to allow clearing the xml files when not required.
Figure 6.7: Selecting description documents and uploading feature

Once the XML files are uploaded admin required to see all the uploaded files in the system. The feature was implemented using Richface’s `<rich:dataTable>` feature.

Figure 6.8: Uploaded description documents on the server

VDDL objects are represented as *Dynamic Description*. To use any of the objects from the deployment descriptions VDDL inventory requires the calling of an internal method `resolveAll()`. This method matches all the resources with their types and creates the mapping between all the objects. Without resolving all files the system cannot run the calculation algorithm. Therefore this step is a mandatory task before running the cost estimation process.
Having the brokering system ready to be used user requirement form was designed to capture user requirements with minimal requirement knowledge as presented in the figure 6.7. The feature was implemented using Richface’s composite component, which contains two sections: <composite:interface> and <composite:implementation>. The <composite:interface> section defines an interface that can be used to configure the component. The <composite:implementation> section contains the component's implementation. It uses the #{cc.attrs.ATTRIBUTE_NAME} expression to access attributes defined in the component's interface. (The cc, which is a reserved keyword in the JSF 2 expression language, stands for composite component.)

Once cloud user enters the required information in the User Requirement form and click “Estimate Cost” button, then user is forwarded to the page with a list of all software products and their estimated cost. The list contains software products, their deployment environment as Cloud and their individual estimated total cost. This is to give an overall cost comparison between various Software Products and IaaS providers. To provide more granular level of detail cost analysis the system was implemented to expand the total cost viewing. This feature was implemented in rich:collapsibleSubTable and rich:collapsibleSubTableToggler components from Richfaces. Sub-tables were used because they can be collapsed and expanded in different modes. rich:collapsibleSubTable component's expandMode attribute was used to specify ajax mode for the system.

Estimated cost for running an individual software product in a cloud infrastructure is presented in descending order of the low-cost option for a software product being on the top of the list. The list was implemented in collapsible sub table to expand individual row and provide detail cost analysis. The initial row expansion provides three core cost components as (i) total instance cost, (ii) total network traffic cost and (iii) total storage cost. Consequently the elaborated cost calculation for each resource is presented in the next associated rows. The rows include each cost factors used in the estimation process against cloud users requirements. The following resources items highlight the associated cost factors:

- **Instance Cost** includes the required size and number of machines, hourly rate charged by the IaaS providers for the used machines and the duration of the infrastructure resource is considered to be use.
- **Network Traffic Cost** includes the number of connectors that represents the data transfer between components, the geographical location of the infrastructure resources in boundary column and the duration of data transfer from the requested period of service.
- **Storage Cost** includes the amount of storage required for the service as block size, rate applied by IaaS providers pricing policy for the storage as per Block (in TB), and the cost factor of the duration for the storage.
<table>
<thead>
<tr>
<th>Product</th>
<th>Cloud</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://apis.com/studentRegistrationProduct3/studentRegistrationProduct">http://apis.com/studentRegistrationProduct3/studentRegistrationProduct</a></td>
<td>tokyoEC2Cloud</td>
<td>54.0</td>
</tr>
<tr>
<td><a href="http://apis.com/studentRegistrationProduct3/studentRegistrationProduct">http://apis.com/studentRegistrationProduct3/studentRegistrationProduct</a></td>
<td>dublinEC2Cloud</td>
<td>120.0</td>
</tr>
<tr>
<td><a href="http://apis.com/studentRegistrationProduct3/studentRegistrationProduct">http://apis.com/studentRegistrationProduct3/studentRegistrationProduct</a></td>
<td>tokyoEC2Cloud</td>
<td>156.0</td>
</tr>
<tr>
<td><a href="http://apis.com/studentRegistrationProduct1/studentRegistrationProduct">http://apis.com/studentRegistrationProduct1/studentRegistrationProduct</a></td>
<td>tokyoEC2Cloud</td>
<td>1890.5</td>
</tr>
<tr>
<td><a href="http://apis.com/studentRegistrationProduct1/studentRegistrationProduct">http://apis.com/studentRegistrationProduct1/studentRegistrationProduct</a></td>
<td>dublinEC2Cloud</td>
<td>2255.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Instance Cost</th>
<th>Total Network Traffic Cost</th>
<th>Total Storage Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>636.0</td>
<td>1488.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Number of Instance</th>
<th>Hourly Rate</th>
<th>Duration (in Days)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>2</td>
<td>1.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>medium</td>
<td>1</td>
<td>1.5</td>
<td>5.0</td>
<td>7.5</td>
</tr>
<tr>
<td>large</td>
<td>1</td>
<td>1.8</td>
<td>5.0</td>
<td>9.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connector Name</th>
<th>Boundary</th>
<th>Duration (In Days)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>studentDBConnector</td>
<td>studentRegistrationProduct</td>
<td>5.0</td>
<td>90.16</td>
</tr>
<tr>
<td>authenticationConnector</td>
<td>studentRegistrationProduct</td>
<td>5.0</td>
<td>240.16002</td>
</tr>
<tr>
<td>userDBIConnector</td>
<td>zonesC</td>
<td>5.0</td>
<td>44.44</td>
</tr>
<tr>
<td>serviceS3Connector</td>
<td>studentRegistrationProduct</td>
<td>5.0</td>
<td>214.16002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Size (in TB)</th>
<th>Rate per Block(TB)</th>
<th>Duration</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>5.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Figure 6.10:** Estimated cost for individual software products against IaaS provider’s pricing policy.
7. Evaluation

This chapter of the dissertation sets out to evaluate the cost estimation techniques we have implemented against the proposed aims and objectives we set out in chapter 1.3 and also highlights the difficulties that we anticipated during the use of VDDL and its restrictive capability in a few circumstances. Following this we propose feedback, suggesting features that would allow getting more information for the system’s future advancement. We then briefly discuss each stage of the project and evaluate my performance.

7.1 Evaluating Against Aims and Objectives

The main aim of the project was to develop a specification to describe software products and cloud IaaS pricing policies that would be able to describe related cost factors to perform estimation on cloud service, in particular infrastructure and software specific user requirements by mapping defined cost factors from description documents. The project was designed with three phases that would be prerequisites to the brokering system.

The objectives we set out for the first phase was to provide a mechanism to describe a domain in particular software products and cloud not for a specific reason but to have a generic specification where current as well as future cost related information can be communicated for cloud deployment. This was based on being able to describe virtual deployments for cloud infrastructure. To accomplish this the author required to build standard notation to describe the artefacts such as kind of required service, machine, storage etc. which was successfully achieved through designing and implementing type description documents (e.g. MachineType.xml). Consequently, the objective of describing the domains based on standard notation was achieved through product description (for software products), environment description (for cloud) and deployment description (for user requirements) documents using VDDL as described in chapter 4 and highlighted in (Appendix A). The subsequent objective was set to build a base level cost model to provide estimation from the described cost factors in the description documents, which was achieved through the model described in chapter 5.6. Having the prerequisites ready to be applied to a novel cost estimation algorithm, designing the algorithm was set as another objective and successfully meets the criteria as discussed in chapter 6.5 that shows the correct behaviour for the cost model. Finally, the objective set to provide a mechanism to request for pre-deployment cost estimation to the broker was achieved through the web application developed and discussed in chapter 6.1.1.

With all of our objectives met we can claim that the brokering service provides a platform that can enable cloud users to be freed from complex cloud overheads and benefit cloud users to change the balance between capital cost (upfront computing infrastructure cost) and the operating cost (maintaining the infrastructure). This can allow people to be more flexible in how they address challenges in a short period of time. For example, there is a necessity for massive parallel computation needing hundreds of machines only for a few hours. Large scale short time analysis demands availability of computing resources for a short period of time. The challenge is cost estimation against cloud users’ requirement. The cost estimation using our algorithm would facilitate cloud users in budgeting. If accurate cost estimates are obtainable, preforming short term micro deployment has a real possibility of becoming the new dominate pattern of gaining access to computing resources. If people get the ability to do so then this would radically change the way they interact with computing resources. To get the most benefit from the cloud services cloud users would require rapid deployment process, which demands quick cost estimation for the service use and workout different relative benefits from the various IaaS providers which is achieved through our brokering system.

7.2 Quality of Background Research

Constructing the automated techniques for cost estimation service involved considerable amounts of research to design, implement and test the proof-of-concept for the cost estimation algorithm within the brokering system.

In the beginning, there were doubts as to whether or not the objectives of the dissertation could have been met. The reason for this was identifying description of a service deployment and deriving techniques to enhance deployment description to communicate charges to the cloud user was an unfamiliar topic. Investigate the potential scope for enhanced communication between service consumer and software product providers, identifying cost types and factors to construct a generic cost model. However, work done by Mihoob, Molina-Jimenez and Shrivastava [52] and Molina-Jimenez, Cook Shrivastava [6] laid the foundation to take the approach with this project taken by the author. The highly regarded paper by Mihoob, Molina-Jimenez and Shrivastava is one that is at the forefront of this area of research. The quality of research done throughout the
dissertation led to a detailed understanding of the problems faced in estimating cost delivery of service deliver for multiple software providers and IaaS providers in a cloud environment.

7.2.1 Related Work

Although not particular, a combination of research aim and objectives were followed in prior work, cost estimation for cloud service deployment or usage have separately been research interests in cloud computing. In [18] multi method approach was followed to present a Total Cost of Ownership (TCO) for Cloud Computing Services. Infrastructure service charge was considered one of the crucial elements for decision making, although abstract high level model was proposed for IaaS service charge assumption. In the similar area of work a consumer centric resource accounting model [57] is proposed where billing factors, more precisely service charges must be collected by consumers. A third party measurement service for storage, operations and bandwidth was also recommended, which gave platform to this dissertation to introduce cost estimation as a brokering service. On another cost reduction approach [53] for data intensive scientific applications in cloud service usage is suggested through building cost aware application, but the study lacks to propose a base level cost estimation algorithm for deriving raw data for estimation. These arguments support the practical and commercial relevance of the study of charging models.

In [6] authors propose a bilateral accounting model where cloud consumer and provider can independently measure resource consumption, which focuses on deployed services and absences the idea of pre-deployment cost estimation that support the practical relevance of the study of deployment description models. In [54] authors proposed that bandwidth cost for network intensive application is clearly based on the runtime service cost, however does not have any optimisation technique to support his argument. In contrast, in this dissertation the author builds the ground to propose software product description mechanism where in-house run-time simulation gives the application developers an opportunity to describe the bandwidth related cost factors. In [60] the authors propose “smart metering” approach as an information gathering device for cloud utility services, where “true cost” of service can be observed by analysing architectural decision point, different pricing structures. However, the missing link is how the information can be described to be captured through the metering device. In this dissertation the author proposes a specification to describe the cloud utility service information in a cloud description document developed in VDDL.

In another paper [61] the authors also propose meter usage approach to validate customer billing according to their usage following a chargeback model highlighting pay-as-you-go principal. The proposal to the solution is presented with virtual platform architecture (VPA) that claims to be the platform for providing accuracy in cost estimation. In [62] authors propose a billing model where consumer charges, offered tariffs can be variable. The proposal is based on load prevailing on the cloud infrastructure where the estimation is calculated on the basis of historical data and the load on the cloud infrastructure is predicted using the auto-regressive integrated moving average (ARIMA) statistical model. The authors use a monitoring service to capture data for cost estimation, but the question would be which data to capture? A base model solution is therefore provided in this dissertation where the author describes the cost factors that will facilitate “smart metering”. In [62] the resource monitoring algorithm focuses on various computing resource monitoring for example, an instance monitoring observes resource utilisation and stores utilised data. However, this particular instance can be described in different size. The question then would be how does the resource monitoring service in [62] distinguish the different size of an instance? In this dissertation the author provides a mechanism to describe a instance as a resource and describe the size as an attribute for that resource in a type description. Another algorithm for price prediction within the same paper entirely depends on the historical data, whereas in this dissertation the author proposes software developers to provide implementation specific details, which will provide cost factors specific to particular software. Finally, in [52] the authors describe an abstract model for accounting resource consumption, which provided the base platform for this project.

7.3 Arjuna’s VDDL Platform

Despite a steep learning curve encountered when initially investigating the techniques for Arjuna’s legacy technology VDDL, having completed the design and implementation of a complete sets of type description by describing a software product and cloud description, subsequent use of the interfaces, APIs and underlying system architecture soon became familiar and seamlessly intuitive.
VDDL is considered by this author to be an ideal tool to describe cost estimation information for software and IaaS providers. The greater flexibility VDDL offers through the mechanism of describing resources to be deployed within a cloud environment a distinct opportunity for software providers and IaaS providers to build a standard platform of describing their services. However, as I developed the techniques to describe a software implementation for cloud deployment and cloud pricing policy I came across features that could have been provided in VDDL that I felt would have enhanced the ability to augment cost estimation information to the description mechanism. These features would have been necessary should the author have wanted to describe the description documents to allowing implementing a richer estimation tool with the designed cost models against different pricing model. This would have also allowed establishing the rationality and accuracy of the cost model. Below is a feedback I would like to provide for VDDL:

- A specific VDDL implementation feature supports describing a purpose for an entity (resource, connector, port or attribute) for example a resource needs to be used to describe a “Machine”, an attribute of purpose “type” is used to describe notation for the resource. For cost estimation that purpose can be extended to describe “Cost” as an attribute for purpose where creating manual type description would not be necessary.

With describing capabilities VDDL is an often used technology in conjunction with Arjuna’s Agility™ platform. The platform offers that the use of Agility™ will increase resource utilisation and consequently applications can be dynamically deployed across resources. It is, however, clear that the dynamic deployment would demand a pre-deployment cost estimation to use the dynamic deployment facility and here this project fits in seamlessly.

### 7.4 Potential Integration with Arjuna’s Agility™ Platform

Agility™ offers organisations the ability to consolidate their existing IT workload onto fewer computational resources, demonstrating clear and measurable savings in cost. Agility removes the barriers between traditional silo deployments and creates an ‘internal cloud’ of shared IT resource within the enterprise and beyond [57].

- With the integration of cost estimation tool to Agility™ the cloud users can select a single service from the software product to be deployed in a specific cloud environment and order a deployment where a service level agreement (SLA) could be generated with the capabilities available from the other brokering service such as relationship agreements (e.g. no changes in pricing policy for 1 year), service quality agreements (e.g. system down time 99.999), and invocations agreements (e.g. instance of virtual machines will be switched off while not in use).
- Agility™ may allow software providers and IaaS providers to customise and host the user request forms directly onto company’s website which can be connected to various broker’s systems. Where software product description and cloud description is part of the Service Level Agreement (SLA). User requesting for cost estimation through the software product provider’s website can directly reach at broker’s end were this particular estimation system can offer approximation of service renting cost, where specific pricing policy is negotiated with providers and brokers.

### 7.5 Brokering Service Design and Implementation Process

As outlined in the next chapter discussing possible direction of future work, there are techniques and features that could have been added that would have produced a more refined cost estimation algorithm. However, considering the time constraints of this dissertation we feel that within the design and implementation section of description documents, cost estimation model and brokering service we have met all the major goals we set out to achieve, but certain limitations still exist.

Known limitations of the brokering system include:

- Forcing a broker to manually design the description documents within an XML file instead of using an automatic description documents generation feature.
- Forcing to deploy the system on the server after a certain amount of time when the system stays idle, this is because due to time constraints no persistent storage was implemented to hold the description documents within the system, where the process of storing dynamic description could have been in a relational database instead of temporarily storing in the system memory.
Forcing the cloud user to write the name of the required service instead of selecting from a list of available services to rent through the broker’s web portal. I tried to solve this by adding a feature of reading all the available software product descriptions in the inventory and showing the result through a drop down list for service name on the user requirements page described in figure. 6.9: however, due to time constraints this could not be implemented.

Displaying textual output of a low-priced software product to be rented as a service instead of graphical representation of an analysis. Ideally I would have liked to flag the estimated cost by adding a justification of the output result on the estimated cost table described in figure 6.10, allowing cloud users to understand why they must select certain options despite being expensive. The reasons could be analysis of system performance, system backup on the cloud infrastructure, replication dependencies, suggestions of potential risk of large client base offering of scalable system, suggesting the consistency trade-offs for various IaaS providers.

Missing integration with a federated cloud computing platform. Agility™ is considered by this author to be an ideal platform upon which software product provider and IaaS provider may build their system from the development perspective, nevertheless in order to assess its overall suitability, business considerations must be studied.

Currently all software product deployments in the system stand in isolation to each other, where the deployments are assumed that previously existing underutilisation resources cannot be made a use of it. For example if pre-deployed database is currently underutilisation, where the algorithm in this dissertation does not take into an account, because the designed was based on an assumption that every deployment was a fresh, new, isolated and non-overlapping deployment. Therefore, the current model only supports static deployment in regards to resource utilisation. This can be implemented where the system is aware of the usage, where underutilised resources can be turned off to reduce cost. The cost factors in this case can be the time frame monitoring of service utilisation, a time dependant factor.

Currently the system does not take into an account of various possible cost factors that can be defined in cloud description, such as software product licensing cost. This immediate next step would be to analyse cloud service provider’s charging models for “cloud software” in cloud marketplace that can be launched quickly and paid by hour [58].

Due to the time constraints the author performed what was feasible to a high standard. Ideally, if time permitted the author would have liked to have:

- Extended the brokering system by adding auto collection feature of cloud and software product descriptions documents from the location specified by from providers instead of uploading them by the system administrator.
- Designed more forms, which can generate more sophisticated service requirement information by monitoring service pattern where the framework can be developed to capture more user requirement after a service is deployed in the cloud infrastructure, because if a framework is available then the user requirement can be added to the system easily rather than re-creating in every instance of client request session.
- Built a sophisticated user requirement form generation framework, which will facilitate creating service specific forms to the portal.
- Advanced the network traffic cost calculation process by building a sophisticated simulation model.
- Integrated the brokering system with Arjuna’s dynamic service agreement management framework Agility to directly provision the requested service in available cloud provider’s infrastructure.
- Advanced the cost model by sophisticated the values assigned to the cost factors by describing network traffic elements in the software description document in more granular and sophisticated level. The advance may follow two distinct direction (i) sophisticating the assumption values within the cost model and (ii) sophisticating the numbers used in the model to be more accurate. This can be developed by monitoring what the characteristics of the real system are?
- Advanced accuracy to the estimation model by refining the estimated value for descriptions.
- Integration with a runtime monitoring system that can be attached to a deployed service and collect feedback from real system running in the cloud infrastructure.
- Advanced the system where estimation is not only provided in pure financial terms but in two category (i) amount of energy consumption and (ii) amount of money cost for a particular service.
- Developed a software product filter with the security requirements to provide cloud users to have a consolidate cost estimation for a particular service that also requires security software.
The design phases of the dissertation was the hardest to comprehend, as this is where all my research had to be accumulated to enable me to produce the cost estimation algorithm and a brokering system that allowed me to satisfy the aims and objectives of the project. The system was implemented in a way that would allow further work to be added by refining the cost factors for software product, cloud and user requirements description documents.

7.6 Testing

The testing strategy involved throughout the project was constantly designing and implementing several description documents with variety of assumed values, particularly for a software product description to establish the rationality and accuracy of the designed cost model. No major problems in terms of the logic of the algorithm occurred during the testing phase of the dissertation. This does not mean that the proof-of-concept provide a bug free solution. The tests aimed to check the cost model to be generic and as accurate as possible. When it came to cost model and brokering system validation testing, I designed and implement a web application with certain workflows to ensure that the brokering system was able to perform the features I set out to achieve in the aims and objectives and was able to estimate the cost for requested service for a cloud user.

7.7 Learning Outcomes

This project provided a broad scope to recognise and articulate the foundational assumptions, central ideas and dominant criticism of the cost estimation techniques in the cloud environment. Aiding the tooling support for the research involved exploring relevant development language to facilitate rapid development and advancing programming skills and techniques.

The research involved in this project extended the developers knowledge by designing the web based brokering system from lower level components to the high-level functionality that has enhanced the developer's design skills of web development with RichFaces, Java EE application design and development and software support for custom languages. The process of selecting which technology to use to express the business logic developed their original skills, improving their analytical and reviewing techniques.

This project was undertaken as an industrial placement with Arjuna Technologies Ltd which has given an opportunity to solve a problem that lies in the state-of-the art cloud computing. The experience also provided them with a chance to use new technology from industry. However, writing a dissertation while working on an industrial project surely demanded a great deal of organising skills, it also provided the cloud computing community to explore and prepare for the future, where cloud brokering for cost estimation is believed to be a consolidated one point of contact.
8 Conclusion and Future Work

In this dissertation I have described an investigation into the notion of describing how the cost factors of cloud computing services that play the major part in potential cost estimation. The author’s approach provides a specification to describe a software product deployment configuration and a cloud IaaS pricing policy for generic cost estimation, a cost model used by the estimation algorithm executable by the broker engine. The author also provides a facility to compare the estimated cost for different combinations of software product sand cloud providers, in particular IaaS provider through a brokering web portal. The brokering system presented in this dissertation is able to estimate cost where:

- Set of standard notations (type descriptions) are agreed between software product and IaaS providers
- Software developer implements the product and IaaS provider describe the cloud pricing policy on the basis of pre-designed standard notations (type descriptions)
- Multiple software providers provide multiple descriptions of single software product, considering no non-defined-type was used in description which the cost estimation algorithm cannot recognise.
- Cloud User is aware of the service required for a certain duration and approximate amount of data will be entered through the service

Interesting future work would be to use the cost model as a basis to describe computing services, apply the cost estimation algorithm as a basis to estimate cost for already running services in the cloud infrastructure and allow advance utilisation for paid virtual machines to reduce cost. To achieve accurate estimation a Resource Consumption model [51] can be developed where allocation based model for machines would allocate virtual machines in possible places in the cloud which would be more economic. Also a usage-based model can be developed for network traffic to capture network traffic transfer pattern and amount to allow accuracy. For instance, instead of assuming data transfer pattern, the monitoring service actually can capture how much data transferred in a specific event and feed this information back to the cost estimation tool for predicting more accurate service cost. This can facilitate describing the cost factors that requires to be taken into account while describing a software product for cloud deployment. Similarly, utilisation over time can be captured by simulation of many deployments, where simulation facilitates validating the deployment process. The more deployment occurs, the many cloud users would require providing information. Consequently, this information can be annotated to a software product description to have more dynamic deployment information. In another case cost factors can be captured through asking users what their needs are and augmenting that information to the software product description to predict run time cost estimation. For example if cloud user requires day time access for 100 students with 500 request on the system per 5 minutes, the technique of describing a software product would incorporate all these cost factors and their values captured from user’s given information. With this information multiple applications can be slot for deployment together when user provides run time environment specific statistics. Following this when software product description is well understood then cost estimation can be reduced by switching off certain virtual instances by integrating data capture and data analysis phase, where for a specific software product the machines that need to be switched off can be annotated and hence more cost factors can be captured for more dynamic deployment environment. In such environment machine can be switched on and off which will increase and reduce estimated cost. In the case of allowing cloud users to spend more when they have budget for the system would require to understand what changes could affect on cost estimation. That would require specific type of information in the model, which can be presented as performance characteristics. Adding such characteristics can analyse network traffic performance over a connector (VDDL feature). Specific performance details for a particular software product can be defined in associated description therefore estimation algorithm will pick a software product for deployment based on the performance.

Finally the verdict is that the cost model needs to be extensible to consider constant change of the cloud pricing models. This is because during the course of this project two new changes were announced by one of the largest cloud service provider: (i) Introducing AWS marketplace in May 2012 and (ii) Amazon Glacier (storage solution) [59] in 21st August 2012.
9 Glossary

B

**Broker:** a company. That plays an intermediary role between cloud service consumer and provider.

C

**Cloud:** “Cloud computing allows computer users to conveniently rent access to fully featured applications, to software development and deployment environments, and to computing infrastructure assets such as network-accessible data storage and processing” [8].

**Cloud Provider:** a company. Who offers computing services over a network generally called Internet.

G

**GUI:** Graphical User Interface. “A [program interface] that takes advantage of the [computer’s graphics] capabilities to make the program easier to use.”

I

**IaaS:** Infrastructure as a Service. “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 (e.g., host firewalls)” [8].

**IaaS Provider:** Infrastructure as a Service provider. A company, that offers computing resources (such as compute power, network bandwidth and storage) in a virtualised environment.

N

**Notation:** Notations specify what operations the service provides and what charges the service incurs.

S

**SLA:** Service Level Agreement. “SLA is a legal document specifying the rules of the legal contract between a consumer and provider” [8].
10 References


Appendix A

1 Sample Description Documents

1.1 Machine Type

The following VDDL description describes a machine type, which details the parameters to collect by both cloud and software product providers. The parameters are used when a standard notation is described by a standard body for a “Machine” and imposes constraints over what constitutes valid cost factors. For example, the constraint is “size of a machine” and the valid cost factor in this case is “smallMachine” or “largeMachine”.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://ieee.org/machineTypes" xmlns:d="http://schema.arjuna.com/vddl">
  <d:attributeType name="machineSizeType"/>
  <d:resourceType name="machineType">
    <d:attributeType name="machineSize" ref="#machineSizeType"/>
  </d:resourceType>
</d:typesDescription>
```

1.2 Use Model Type

The following VDDL description describes a use model type, which details the parameters to collect by the broker for the cost estimation algorithm. These parameters will be specified when a standard notation is described by a standard body for a “User Requirements”. For example, a parameter is that type of service represented as “serviceoperation”, duration of the service required represented as “serviceduration”.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://ieee.org/useModelTypes" xmlns:d="http://schema.arjuna.com/vddl">
  <d:attributeType name="serviceDurationType"/>
  <d:attributeType name="serviceOperationType"/>
  <d:attributeType name="dataInputAmountType"/>
  <d:resourceType name="costConstraintType">
    <d:attributeType name="serviceDuration" ref="#serviceDurationType"/>
    <d:attributeType name="serviceOperation" ref="#serviceOperationType"/>
    <d:attributeType name="dataInputAmount" ref="#dataInputAmountType"/>
  </d:resourceType>
</d:typesDescription>
```
1.3 Network Boundary Type

The following VDDL description describes a network boundary type, which details the parameters to collect when a standard notation is described by a standard body for a “Network Boundary”, which is used to calculate data transfer between virtual machines. The parameter “networkBoundary” will allow software product provider to specify where the software components will be deployed.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://ieee.org/networkTrafficBoundaryTypes" xmlns:d="http://schema.arjuna.com/vddl">  
  <d:attributeType name="networkBoundaryType"/>  
  <d:resourceType name="boundaryType">  
    <d:attributeType name="networkBoundary" ref="#networkBoundaryType"/>  
  </d:resourceType>  
</d:typesDescription>
```

1.4 Student Registration Application Type

The following VDDL description describes an application type, which details the parameters to collect by software product and cloud providers, when a standard notation is described by a standard body for a “Type of Service” that a software product provides.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://sap.com/studentRegistrationApplTypes" xmlns:d="http://schema.arjuna.com/vddl">  
  <d:portType name="studentRegistrationServiceType"/>  
  <d:resourceType name="studentRegistrationApplType">  
    <d:portType name="service" ref="#studentRegistrationServiceType"/>  
  </d:resourceType>  
</d:typesDescription>
```
1.5 Software Product (Student Registration) Implementation Type

The following VDDL description describes an implementation of a software product, which details the parameters to collect by software product and cloud providers when a standard notation is described by a standard body for a "Software Product" that a software product provides.

```xml
<?xml version="1.0" encoding="UTF-8"?><d:typesDescription name="http://sap.com/studentRegistrationImplTypes" xmlns:d="http://schema.arjuna.com/vddl">  <d:portType name="studentDBNeedType"/>  <d:portType name="studentDBServiceType"/>  <d:portType name="authenticationNeedType"/>  <d:portType name="authenticationServiceType"/>  <d:portType name="userDBNeedType"/>  <d:portType name="userDBServiceType"/>   <d:portType name="serviceS3NeedType"/>  <d:portType name="serviceS3ServiceType"/>  
  <d:connectorType name="studentDBConnectorType">    <d:portType name="studentDBNeed" ref="#studentDBNeedType"/>    <d:portType name="studentDBService" ref="#studentDBServiceType"/>  </d:connectorType>  
  <d:connectorType name="authenticationConnectorType">    <d:portType name="authenticationNeed" ref="#authenticationNeedType"/>    <d:portType name="authenticationService" ref="#authenticationServiceType"/>  </d:connectorType>  
  <d:connectorType name="userDBConnectorType">    <d:portType name="studentDBNeed" ref="#userDBNeedType"/>    <d:portType name="studentDBService" ref="#userDBServiceType"/>  </d:connectorType>  
  <d:resourceType name="studentWebApplType">    <d:portType name="studentRegistrationService" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationServiceType"/>    <d:portType name="authenticationNeed" ref="#authenticationNeedType"/>  </d:resourceType>  
  <d:resourceType name="studentDBType">```
<d:portType name="studentDBService" ref="#studentDBServiceType"/>
</d:resourceType>

<d:resourceType name="authenticationWebApp1Type">
  <d:portType name="authenticationService" ref="#authenticationServiceType"/>
  <d:portType name="studentDBNeed" ref="#studentDBNeedType"/>
</d:resourceType>

<d:resourceType name="userDBType">
  <d:portType name="userDBService" ref="#userDBServiceType"/>
</d:resourceType>

<!-- New resourceTypes are created for Blobs -->
<d:resourceType name="studentServiceBlobType">
  <d:portType name="serviceS3Service" ref="#serviceS3ServiceType"/>
</d:resourceType> <!-- New connectors are created for Resource: S3Storage <resource: blob> -->
<d:connectorType name="serviceS3ConnectorType">
  <d:portType name="serviceS3Need" ref="#serviceS3NeedType"/>
  <d:portType name="serviceS3Service" ref="#serviceS3ServiceType"/>
</d:connectorType>
</d:resourceType> <!-- connecting components to blobs(storage components) -->
<d:connectorType name="authenticationS3ConnectorType">
  <d:portType name="authenticationS3Need" ref="#authenticationS3NeedType"/>
  <d:portType name="authenticationS3Service" ref="#authenticationS3ServiceType"/>
</d:connectorType>

<d:connectorType name="studentDBS3ConnectorType">
  <d:portType name="studentDBS3Need" ref="#studentDBS3NeedType"/>
  <d:portType name="studentDBS3Service" ref="#studentDBS3ServiceType"/>
</d:connectorType>

<d:connectorType name="userDBS3ConnectorType">
  <d:portType name="userDBS3Need" ref="#userDBS3NeedType"/>
  <d:portType name="userDBS3Service" ref="#userDBS3ServiceType"/>
</d:connectorType>
</d:typesDescription>
1.6 EC2 Cloud Type

The following VDDL description describes an EC2 Cloud type, which details the parameters to collect for cloud and software product provider, when a standard notation is described by a standard body for a “Type of Service” and their pricing policy that a IaaS provider (here Amazon’s EC2 [9] was used as a sample cloud IaaS service) provides.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<typesDescription name="http://amazon.com/ec2CloudTypes" xmlns:d="http://schema.arjuna.com/vddl">
  <attributeType name="perHourRentalPriceType"/>
  <attributeType name="perMBPriceType"/>
  <attributeType name="storageAmountPriceType"/>
  <attributeType name="perGBPriceType"/>
  <resourceType name="ec2CloudPricingType">
    <attributeType name="smallInstancePrice" ref="#perHourRentalPriceType"/>
    <attributeType name="mediumInstancePrice" ref="#perHourRentalPriceType"/>
    <attributeType name="largeInstancePrice" ref="#perHourRentalPriceType"/>
    <attributeType name="extraLargeInstancePrice" ref="#perHourRentalPriceType"/>
    <attributeType name="storagePrice" ref="#storageAmountPriceType"/>
    <attributeType name="interS3StorageIn" ref="#perGBPriceType"/>
    <attributeType name="interS3StorageOut" ref="#perGBPriceType"/>
    <attributeType name="intraS3Storage" ref="#perGBPriceType"/>
    <attributeType name="interComponentIn" ref="#perMBPriceType"/>
    <attributeType name="interComponentOut" ref="#perMBPriceType"/>
    <attributeType name="interMachineIn" ref="#perMBPriceType"/>
    <attributeType name="interMachineOut" ref="#perMBPriceType"/>
    <attributeType name="intraMachine" ref="#perMBPriceType"/>
    <attributeType name="interZoneIn" ref="#perMBPriceType"/>
    <attributeType name="interZoneOut" ref="#perMBPriceType"/>
    <attributeType name="intraZone" ref="#perMBPriceType"/>
    <attributeType name="interRegionIn" ref="#perMBPriceType"/>
  </resourceType>
</typesDescription>
```
1.7 S3 Cloud Type

The following VDDL description describes an S3 Cloud type, which details the parameters to collect for software product providers and cloud providers, when a standard notation is described by a standard body, for a particular “Type of Service” and their pricing policy that a IaaS provider (here Amazon’s S3 [41] was used as a sample cloud IaaS service) provides.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://amazon.com/s3CloudTypes" xmlns:d="http://schema.arjuna.com/vddl">
    <d:attributeType name="storageAmountPriceType"/>
    <d:attributeType name="perGBPriceType"/>

    <d:resourceType name="s3CloudPricingType">
        <d:attributeType name="storagePrice" ref="#storageAmountPriceType"/>
        <d:attributeType name="interS3StorageIn" ref="#perGBPriceType"/>
        <d:attributeType name="interS3StorageOut" ref="#perGBPriceType"/>
        <d:attributeType name="intraS3Storage" ref="#perGBPriceType"/>
        <d:attributeType name="interZoneIn" ref="#perGBPriceType"/>
        <d:attributeType name="interZoneOut" ref="#perGBPriceType"/>
        <d:attributeType name="intraZone" ref="#perGBPriceType"/>
    </d:resourceType>
</d:typesDescription>
```
1.8 Storage Type

The following VDDL description describes a storage type, which details the parameters to collect for software product and cloud providers, when a standard notation is described by a standard body for particular infrastructure storage such as “Storage” and imposes constraints over what constitutes valid cost factors. For example, the constraint is “amount of storage required by the service”, represented as “storageAmount”.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://ieee.org/storageTypes" xmlns:d="http://schema.arjuna.com/vddl">
    <d:attributeType name="storageAmountType"/>
    <d:resourceType name="storageType">
        <d:attributeType name="storageAmount" ref="#storageAmountType"/>
    </d:resourceType>
</d:typesDescription>
```

1.9 EC2 Cloud Description (resource and pricing) for Europe

The following VDDL description describes a particular IaaS provider’s offerings such as EC2Cloud for a particular region as “Europe”, which details the parameters to collect for software product and cloud providers, when a standard notation is described by a standard body for particular infrastructure resource and imposes constraints over what constitutes valid cost factors. For example, charge for storage usage shows as “storagePrice”, price for a machine for a specific size presented as “smallInstancePrice”.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:typesDescription name="http://ieee.org/cloudTypes" xmlns:d="http://schema.arjuna.com/vddl">
    <d:attributeType name="storageAmountType"/>
    <d:resourceType name="storageType">
        <d:attributeType name="storageAmount" ref="#storageAmountType"/>
    </d:resourceType>
</d:typesDescription>
```
<xml version="1.0" encoding="UTF-8"?>
<environmentDescription name="http://amazon.com/europeEC2Clouds" xmlns:d="http://schema.arjuna.com/vddl">
  <resource name="dublinEC2Cloud">
    <type name="type" ref="http://amazon.com/ec2CloudTypes#ec2CloudPricingType">
      <attributeMapping name="smallInstancePrice" ref="#dublinEC2Cloud/smallInstancePrice"/>
      <attributeMapping name="mediumInstancePrice" ref="#dublinEC2Cloud/mediumInstancePrice"/>
      <attributeMapping name="largeInstancePrice" ref="#dublinEC2Cloud/largeInstancePrice"/>
      <attributeMapping name="extraLargeInstancePrice" ref="#dublinEC2Cloud/extraLargeInstancePrice"/>
      <attributeMapping name="storagePrice" ref="#dublinEC2Cloud/storagePrice"/>
      <attributeMapping name="interS3StorageIn" ref="#dublinEC2Cloud/interS3StorageIn"/>
      <attributeMapping name="interS3StorageOut" ref="#dublinEC2Cloud/interS3StorageOut"/>
      <attributeMapping name="intraS3Storage" ref="#dublinEC2Cloud/intraS3Storage"/>
      <attributeMapping name="interComponentIn" ref="#dublinEC2Cloud/interComponentIn"/>
      <attributeMapping name="interComponentOut" ref="#dublinEC2Cloud/interComponentOut"/>
      <attributeMapping name="interMachineIn" ref="#dublinEC2Cloud/interMachineIn"/>
      <attributeMapping name="interMachineOut" ref="#dublinEC2Cloud/interMachineOut"/>
      <attributeMapping name="intraMachine" ref="#dublinEC2Cloud/intraMachine"/>
      <attributeMapping name="interZoneIn" ref="#dublinEC2Cloud/interZoneIn"/>
      <attributeMapping name="interZoneOut" ref="#dublinEC2Cloud/interZoneOut"/>
      <attributeMapping name="intraZone" ref="#dublinEC2Cloud/intraZone"/>
      <attributeMapping name="interRegionIn" ref="#dublinEC2Cloud/interRegionIn"/>
      <attributeMapping name="interRegionOut" ref="#dublinEC2Cloud/interRegionOut"/>
      <attributeMapping name="intraRegion" ref="#dublinEC2Cloud/intraRegion"/>
      <attributeMapping name="interEC2CloudDTPricingTypeIn" ref="#dublinEC2Cloud/interEC2CloudDTPricingTypeIn"/>
      <attributeMapping name="interEC2CloudDTPricingTypeOut" ref="#dublinEC2Cloud/interEC2CloudDTPricingTypeOut"/>
      <attributeMapping name="intraEC2CloudDTPricingType" ref="#dublinEC2Cloud/intraEC2CloudDTPricingType"/>
      <attributeMapping name="interProviderCloudDTPricingTypeIn" ref="#dublinEC2Cloud/interProviderCloudDTPricingTypeIn"/>
      <attributeMapping name="interProviderCloudDTPricingTypeOut" ref="#dublinEC2Cloud/interProviderCloudDTPricingTypeOut"/>
      <attributeMapping name="intraProviderCloudDTPricingType" ref="#dublinEC2Cloud/intraProviderCloudDTPricingType"/>
    </type>
  </resource>
</environmentDescription>
<!-- New price for amount of storage -->
<d:attribute name="storagePrice" value="0.5">
<d:type name="storageAmountPriceType" ref="http://amazon.com/ec2CloudTypes#storageAmountPriceType"/>
</d:attribute>

<!-- New price for storage traffic -->
<d:attribute name="interS3StorageIn" value="2.0">
<d:type name="perGBPriceType" ref="http://amazon.com/ec2CloudTypes#perGBPriceType"/>
</d:attribute>
<d:attribute name="interS3StorageOut" value="5.0">
<d:type name="perGBPriceType" ref="http://amazon.com/ec2CloudTypes#perGBPriceType"/>
</d:attribute>
<d:attribute name="intraS3Storage" value="1.2">
<d:type name="perGBPriceType" ref="http://amazon.com/ec2CloudTypes#perGBPriceType"/>
</d:attribute>

<d:attribute name="smallInstancePrice" value="1.0">
<d:type name="perHourRentalPriceType" ref="http://amazon.com/ec2CloudTypes#perHourRentalPriceType"/>
</d:attribute>
<d:attribute name="mediumInstancePrice" value="1.5">
<d:type name="perHourRentalPriceType" ref="http://amazon.com/ec2CloudTypes#perHourRentalPriceType"/>
</d:attribute>
<d:attribute name="largeInstancePrice" value="1.8">
<d:type name="perHourRentalPriceType" ref="http://amazon.com/ec2CloudTypes#perHourRentalPriceType"/>
</d:attribute>
<d:attribute name="extraLargeInstancePrice" value="1.9">
<d:type name="perHourRentalPriceType" ref="http://amazon.com/ec2CloudTypes#perHourRentalPriceType"/>
</d:attribute>

<d:attribute name="interComponentIn" value="0.0">
<d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>
<d:attribute name="interComponentOut" value="0.0">
<d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interMachineIn" value="0.0">
<d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>
<d:attribute name="interMachineOut" value="0.0">
<d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

<d:attribute name="intraMachine" value="0.0">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interZoneIn" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interZoneOut" value="0.03">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="intraZone" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interRegionIn" value="0.01">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interRegionOut" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="intraRegion" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interEC2CloudDTPricingTypeIn" value="0.01">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interEC2CloudDTPricingTypeOut" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="intraEC2CloudDTPricingType" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interProviderCloudDTPricingTypeIn" value="0.01">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>

<d:attribute name="interProviderCloudDTPricingTypeOut" value="0.02">
  <d:type name="perMBPriceType" ref="http://amazon.com/ec2CloudTypes#perMBPriceType"/>
</d:attribute>
1.10A Software Product Description

The following VDDL description specifies a software product, which details the parameters for software product providers to collect for software product and cloud provider when a standard notation is described by a standard body, for a “Software Product” that a software product provides. The following description however refers to software Student Registration, which contains four deployable components of one authentication server, one application server and two databases. The sample is one particular implementation of a software product description (studentRegistrationProduct1).

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <d:product name="studentRegistrationProduct">
    <!-- Type of Service this product provides -->
    <d:type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationApplType">
      <d:portMapping name="service" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/type/studentRegistrationService"/>
    </d:type>
    <!-- Region A as a Resource -->
    <d:resource name="regionA">
      <d:type name="type" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
        <d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionA/networkBoundary"/>
      </d:type>
      <d:attribute name="networkBoundary" value="region">
        <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
      </d:attribute>
    </d:resource>
    <!-- Zone A as a Resource -->
    <d:resource name="zoneA">
      <d:type name="type" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
        <d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionA/zoneA/networkBoundary"/>
      </d:type>
      <d:attribute name="networkBoundary" value="zone">
        <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
      </d:attribute>
    </d:resource>
    <!-- Type of Resource as Machine -->
    <d:resource name="machine1">
      <d:type name="type" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
        <d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionA/zoneA/networkBoundary"/>
      </d:type>
      <d:attribute name="networkBoundary" value="machine1">
        <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
      </d:attribute>
    </d:resource>
  </d:product>
</d:productDescription>
```
<d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionA/zoneA/machine1/networkBoundary"/>
</d:attributeMapping>
<d:type name="type" ref="http://ieee.org/machineTypes#machineType">
<d:attributeMapping name="machineSize" ref="#studentRegistrationProduct/regionA/zoneA/machine1/machineSize"/>
</d:type>

<!-- Attribute description for the type of resource -->
<d:attribute name="networkBoundary" value="machine">
<d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
</d:attribute>
<d:attribute name="machineSize" value="small">
<d:type name="machineSizeType" ref="http://ieee.org/machineTypes#machineSizeType"/>
</d:attribute>

<!-- Component as a Resource 1 -->
<d:resource name="webapp1">
<!-- Type description for the type of resource -->
<d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentWebApplType">
<d:portMapping name="studentRegistrationService" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/studentRegistrationService"/>
<d:portMapping name="studentDBNeed" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/studentDBNeed"/>
<d:portMapping name="authenticationNeed" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/authenticationNeed"/>
<d:portMapping name="serviceS3Need" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/serviceS3Need"/>
</d:type>

<!-- New port creation for the component -->
<!-- Port 1 -->
<d:port name="studentRegistrationService">
<!-- Functional Type for the port -->
<d:type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationServiceType"/>
<!-- Non functional Type for the port -->
<d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
<d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/studentRegistrationService/dataProduced"/>
</d:type>
<d:attribute name="dataProduced" value="200">
<d:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
</d:attribute>
</d:port>

<!-- Port 2 -->
<d:port name="studentDBNeed">
<!-- Functional Type for the port -->
<d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBNeedType"/>
<!-- Non functional Type for the port -->
<d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
<d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/studentDBNeed/dataProduced"/>
</d:type>
</d:port>
<d:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
</d:attribute>
</d:resource>
</d:resource>
</d:resource>

<!-- Region C as a Resource -->
<d:resource name="regionC">
    <d:type name="type" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
        <d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionC/networkBoundary"/>
    </d:type>
    <d:attribute name="networkBoundary" value="region">
        <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
    </d:attribute>
</d:resource>

<!-- Zone C as a Resource -->
<d:resource name="zoneC">
    <d:type name="type" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
        <d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionC/zoneC/networkBoundary"/>
    </d:type>
    <d:attribute name="networkBoundary" value="zone">
        <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
    </d:attribute>
</d:resource>

<!-- Machine as a Resource 2 -->
<d:resource name="machine2">
    <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
        <d:attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionC/zoneC/machine2/networkBoundary"/>
    </d:type>
    <d:type name="type" ref="http://ieee.org/machineTypes#machineType">
        <d:attributeMapping name="machineSize" ref="#studentRegistrationProduct/regionC/zoneC/machine2/machineSize"/>
    </d:type>
    <d:attribute name="networkBoundary" value="machine">
        <d:type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#networkBoundaryType"/>
    </d:attribute>
    <d:attribute name="machineSize" value="small">
        <d:type name="machineSizeType" ref="http://ieee.org/machineTypes#machineSizeType"/>
    </d:attribute>
</d:resource>
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

<!-- Component as a Resource 2 -->
<resource name="webapp2">
  <type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationWebApplType">
    <portMapping name="authenticationService" ref="#studentRegistrationProduct/regionC/zoneC/machine2/webapp2/authenticationService"/>
    <portMapping name="userDBNeed" ref="#studentRegistrationProduct/regionC/zoneC/machine2/webapp2/userDBNeed"/>
  </type>
  <!-- Port 1 -->
  <port name="authenticationService">
    <type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationServiceType">
      <!-- Non functional Type for the port -->
      <attributeMapping name="dataProduced" ref="#studentRegistrationProduct/regionC/zoneC/machine2/webapp2/authenticationService/dataProduced"/>
    </type>
    <attribute name="dataProduced" value="20"/>
  </port>
  <!-- Port 2 -->
  <port name="userDBNeed">
    <type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBNeedType">
      <!-- Non functional Type for the port -->
      <attributeMapping name="dataProduced" ref="#studentRegistrationProduct/regionC/zoneC/machine2/webapp2/userDBNeed/dataProduced"/>
    </type>
    <attribute name="dataProduced" value="20"/>
  </port>
</resource>

<resource name="machine4">
  <type name="networkBoundaryType" ref="http://ieee.org/networkTrafficBoundryTypes#boundryType">
    <attributeMapping name="networkBoundary" ref="#studentRegistrationProduct/regionC/zoneC/machine4/networkBoundary"/>
  </type>
</resource>
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

<d:type name="meteredPortType"
ref="http://ieee.org/meteredPortTypes#meteredPortType">
  <d:attributeMapping name="dataProduced"
ref="#studentRegistrationProduct/regionD/zoneD/s3Storage/blob1/serviceS3Service/dataProduced"/>
</d:type>

<d:attribute name="dataProduced" value="25">
  <d:type name="dataProduced"
ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
</d:attribute>

<d:resource>
  <d:connector name="studentDBConnector">
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBConnectorType">
      <d:portMapping name="studentDBNeed" ref="#studentRegistrationProduct/studentDBConnector/studentDBNeed"/>
      <d:portMapping name="studentDBService" ref="#studentRegistrationProduct/studentDBConnector/studentDBService"/>
    </d:type>
    <d:port name="studentDBNeed" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/type/studentDBNeed"/>
    <d:port name="studentDBService" ref="#studentRegistrationProduct/regionB/zoneB/machine3/db1/type/studentDBService"/>
  </d:connector>

  <d:connector name="authenticationConnector">
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationConnectorType">
      <d:portMapping name="authenticationNeed" ref="#studentRegistrationProduct/authenticationConnector/authenticationNeed"/>
      <d:portMapping name="authenticationService" ref="#studentRegistrationProduct/authenticationConnector/authenticationService"/>
    </d:type>
    <d:port name="authenticationNeed" ref="#studentRegistrationProduct/regionA/zoneA/machine1/webapp1/type/authenticationNeed"/>
    <d:port name="authenticationService" ref="#studentRegistrationProduct/regionC/zoneC/machine2/webapp2/type/authenticationService"/>
  </d:connector>

  <d:connector name="userDBConnector">
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#userDBConnectorType">
      <d:portMapping name="userDBNeed" ref="#studentRegistrationProduct/userDBConnector/userDBNeed"/>
      <d:portMapping name="userDBService" ref="#studentRegistrationProduct/userDBConnector/userDBService"/>
    </d:type>
    <d:port name="userDBNeed" ref="#studentRegistrationProduct/regionC/zoneC/machine2/webapp2/type/userDBNeed"/>
  </d:connector>
</d:resource>
1.11A Software Product Description

The following VDDL description specifies a software product, which details the parameters for software product providers to collect for software product and cloud provider when a standard notation is described by a standard body for a “Software Product” that a software product provides. The following description however refers to software Student Registration, which contains four deployable components of one authentication server, one application server and two databases. The sample is a different implementation of a software product description named as studentRegistrationProduct2.

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <d:product name="studentRegistrationProduct">
    <!-- Type of Service this product provides -->
    <d:type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationApplType">
      <d:portMapping name="service" ref="#studentRegistrationProduct/machine1/webapp1/type/studentRegistrationService"/>
    </d:type>
    <!-- Machine as a Resource 1 -->
    <d:resource name="machine1">
      <!-- Type of Resource -->
      <d:type name="type" ref="http://ieee.org/machineTypes#machineType">
        <d:attributeMapping name="machineSize" ref="#studentRegistrationProduct/machine1/machineSize"/>
      </d:type>
      <!-- Attribute description for the type of resource -->
      <d:attribute name="machineSize" value="small"/>
    </d:resource>
    <!-- Machine as a Resource 2 -->
    <d:resource name="machine2">
      <!-- Type of Resource -->
      <d:type name="type" ref="http://ieee.org/machineTypes#machineType">
        <d:attributeMapping name="machineSize" ref="#studentRegistrationProduct/machine2/machineSize"/>
      </d:type>
      <!-- Attribute description for the type of resource -->
      <d:attribute name="machineSize" value="small"/>
    </d:resource>
    <!-- Machine as a Resource 3 -->
    <d:resource name="machine3">
      <!-- Type of Resource -->
      <d:type name="type" ref="http://ieee.org/machineTypes#machineType">
        <d:attributeMapping name="machineSize" ref="#studentRegistrationProduct/machine3/machineSize"/>
      </d:type>
      <!-- Attribute description for the type of resource -->
      <d:attribute name="machineSize" value="small"/>
    </d:resource>
    <!-- Machine as a Resource 4 -->
    <d:resource name="machine4">
      <!-- Type of Resource -->
      <d:type name="type" ref="http://ieee.org/machineTypes#machineType">
        <d:attributeMapping name="machineSize" ref="#studentRegistrationProduct/machine4/machineSize"/>
      </d:type>
      <!-- Attribute description for the type of resource -->
      <d:attribute name="machineSize" value="small"/>
    </d:resource>
  </d:product>
</d:productDescription>
```
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

<d:attribute name="machineSizeType" ref="http://ieee.org/machineTypes#machineSizeType"/>
</d:attribute>

<d:resource name="webapp1">
  <!-- Type description for the type of resource -->
  <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentWebApp1Type">
    <d:portMapping name="studentRegistrationService" ref="#studentRegistrationProduct/machine1/webapp1/studentRegistrationService"/>
    <d:portMapping name="studentDBNeed" ref="#studentRegistrationProduct/machine1/webapp1/studentDBNeed"/>
    <d:portMapping name="authenticationNeed" ref="#studentRegistrationProduct/machine1/webapp1/authenticationNeed"/>
  </d:type>

  <!-- New port creation for the component -->
  <!-- Port 1 -->
  <d:port name="studentRegistrationService">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationServiceType"/>
    <!-- Non functional Type for the port -->
    <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
      <d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/machine1/webapp1/studentRegistrationService/dataProduced"/>
    </d:type>
    <d:attribute name="dataProduced" value="2" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
  </d:port>

  <!-- Port 2 -->
  <d:port name="studentDBNeed">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBNeedType"/>
    <!-- Non functional Type for the port -->
    <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
      <d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/machine1/webapp1/studentDBNeed/dataProduced"/>
    </d:type>
    <d:attribute name="dataProduced" value="2" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
  </d:port>

  <!-- Port 3 -->
  <d:port name="authenticationNeed">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationNeedType"/>
    <!-- Non functional Type for the port -->
  </d:port>
</d:resource>
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

```xml
<d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
    <d:attributeMapping ref="#studentRegistrationProduct/machine1/webapp1/authenticationNeed/dataProduced" name="dataProduced"/>
    <d:type name="dataProduced" value="2" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
</d:attribute>
</d:port>
</d:resource>

<!-- Component as a Resource 2 -->
<d:resource name="db1">
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBType">
        <d:portMapping ref="#studentRegistrationProduct/machine1/db1/studentDBService" name="studentDBService"/>
    </d:type>
    <d:port name="studentDBService">
        <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
            <d:attributeMapping ref="#studentRegistrationProduct/machine1/db1/studentDBService/dataProduced" name="dataProduced"/>
            <d:attribute name="dataProduced" value="2" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
        </d:attribute>
    </d:port>
</d:resource>

<!-- Machine as a Resource 2 -->
<d:resource name="machine2">
    <d:type name="type" ref="http://ieee.org/machineTypes#machineType">
        <d:attributeMapping ref="#studentRegistrationProduct/machine2/machineSize" name="machineSize"/>
    </d:type>
    <d:attribute name="machineSize" value="small" ref="http://ieee.org/machineTypes#machineSizeType"/>
</d:attribute>
</d:resource>

<!-- Component as a Resource 3 -->
<d:resource name="webapp2">
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationWebApp1Type">
        <d:portMapping ref="#studentRegistrationProduct/machine2/webapp2/authenticationService" name="authenticationService"/>
        <d:portMapping ref="#studentRegistrationProduct/machine2/webapp2/userDBNeed" name="userDBNeed"/>
</d:resource>
```
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

<!-- Port 1 -->
<dp:port name="authenticationService">
  <dp:type ref="http://sap.com/studentRegistrationImplTypes#authenticationServiceType" name="type">
    <!-- Non functional Type for the port -->
    <dp:meteredPortType ref="http://ieee.org/meteredPortTypes#meteredPortType">
      <dp:attributeMapping ref="#studentRegistrationProduct/machine2/webapp2/authenticationService/dataProduced" name="dataProduced"/>
    </dp:meteredPortType>
  </dp:type>
  <dp:attribute name="dataProduced" value="2">
    <dp:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
  </dp:attribute>
</dp:port>

<!-- Port 2 -->
<dp:port name="userDBNeed">
  <dp:type ref="http://sap.com/studentRegistrationImplTypes#studentDBNeedType" name="type">
    <!-- Non functional Type for the port -->
    <dp:meteredPortType ref="http://ieee.org/meteredPortTypes#meteredPortType">
      <dp:attributeMapping ref="#studentRegistrationProduct/machine2/webapp2/userDBNeed/dataProduced" name="dataProduced"/>
    </dp:meteredPortType>
  </dp:type>
  <dp:attribute name="dataProduced" value="2">
    <dp:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
  </dp:attribute>
</dp:port>

<!-- Component as a Resource 4 -->
<dp:resource name="db2">
  <dp:type ref="http://sap.com/studentRegistrationImplTypes#userDBType" name="type">
    <dp:portMapping ref="#studentRegistrationProduct/machine2/db2/userDBService" name="userDBService"/>
  </dp:type>
  <dp:meteredPortType ref="http://ieee.org/meteredPortTypes#meteredPortType">
    <dp:attributeMapping ref="#studentRegistrationProduct/machine2/db2/userDBService/dataProduced" name="dataProduced"/>
  </dp:meteredPortType>
  <dp:attribute name="dataProduced" value="2">
    <dp:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
  </dp:attribute>
</dp:resource>
1.12A Software Product Description

The following VDDL description specifies a software product, which details the parameters for software product providers to collect for software product and cloud provider when a standard notation is described by a standard body for a “Software Product” that a software product provides. The following description however refers to software Student Registration, which contains four deployable components of one authentication server, one application server and two databases. The sample is a different implementation of a software product description named as studentRegistrationProduct3.
<xml version="1.0" encoding="UTF-8"?>

  <product name="studentRegistrationProduct">
    <!-- Type of Service this product provides -->
    <type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationApplType">
      <portMapping name="service" ref="#studentRegistrationProduct/machine1/webapp1/type/studentRegistrationService"/>
    </type>
    <resource name="machine1">
      <!-- Type of Resource -->
      <type name="type" ref="http://ieee.org/machineTypes#machineType">
        <attributeMapping name="machineSize" ref="#studentRegistrationProduct/machine1/machineSize"/>
      </type>
      <!-- Attribute description for the type of resource -->
      <attribute name="machineSize" value="small">
        <type name="machineSizeType" ref="http://ieee.org/machineTypes#machineSizeType"/>
      </attribute>
      <!-- Component as a Resource 1 -->
      <resource name="webapp1">
        <!-- Type description for the type of resource -->
        <type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentWebApplType">
          <portMapping name="studentRegistrationService" ref="#studentRegistrationProduct/machine1/webapp1/studentRegistrationService"/>
          <portMapping name="studentDBNeed" ref="#studentRegistrationProduct/machine1/webapp1/studentDBNeed"/>
          <portMapping name="authenticationNeed" ref="#studentRegistrationProduct/machine1/webapp1/authenticationNeed"/>
        </type>
        <!-- New port creation for the component -->
        <!-- Port 1 -->
        <port name="studentRegistrationService">
          <!-- Functional Type for the port -->
          <type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationServiceType"/>
          <type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
            <attributeMapping name="dataProduced" ref="#studentRegistrationProduct/machine1/webapp1/studentRegistrationService/dataProduced"/>
          </type>
          <attribute name="dataProduced" value="100">
            <type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
          </attribute>
        </port>
      </resource>
      <!-- Port 2 -->
      <!-- Port 3 -->
      <!-- Port 4 -->
      <!-- Port 5 -->
      <!-- Port 6 -->
      <!-- Port 7 -->
      <!-- Port 8 -->
      <!-- Port 9 -->
      <!-- Port 10 -->
      <!-- Port 11 -->
      <!-- Port 12 -->
      <!-- Port 13 -->
      <!-- Port 14 -->
      <!-- Port 15 -->
      <!-- Port 16 -->
      <!-- Port 17 -->
      <!-- Port 18 -->
      <!-- Port 19 -->
      <!-- Port 20 -->
      <!-- Port 21 -->
      <!-- Port 22 -->
      <!-- Port 23 -->
      <!-- Port 24 -->
      <!-- Port 25 -->
      <!-- Port 26 -->
      <!-- Port 27 -->
      <!-- Port 28 -->
      <!-- Port 29 -->
      <!-- Port 30 -->
      <!-- Port 31 -->
      <!-- Port 32 -->
      <!-- Port 33 -->
      <!-- Port 34 -->
      <!-- Port 35 -->
      <!-- Port 36 -->
      <!-- Port 37 -->
      <!-- Port 38 -->
      <!-- Port 39 -->
      <!-- Port 40 -->
      <!-- Port 41 -->
      <!-- Port 42 -->
      <!-- Port 43 -->
      <!-- Port 44 -->
      <!-- Port 45 -->
      <!-- Port 46 -->
      <!-- Port 47 -->
      <!-- Port 48 -->
      <!-- Port 49 -->
      <!-- Port 50 -->
      <!-- Port 51 -->
      <!-- Port 52 -->
      <!-- Port 53 -->
      <!-- Port 54 -->
      <!-- Port 55 -->
      <!-- Port 56 -->
      <!-- Port 57 -->
      <!-- Port 58 -->
      <!-- Port 59 -->
      <!-- Port 60 -->
      <!-- Port 61 -->
      <!-- Port 62 -->
      <!-- Port 63 -->
      <!-- Port 64 -->
      <!-- Port 65 -->
      <!-- Port 66 -->
      <!-- Port 67 -->
      <!-- Port 68 -->
      <!-- Port 69 -->
      <!-- Port 70 -->
      <!-- Port 71 -->
      <!-- Port 72 -->
      <!-- Port 73 -->
      <!-- Port 74 -->
      <!-- Port 75 -->
      <!-- Port 76 -->
      <!-- Port 77 -->
      <!-- Port 78 -->
      <!-- Port 79 -->
      <!-- Port 80 -->
      <!-- Port 81 -->
      <!-- Port 82 -->
      <!-- Port 83 -->
      <!-- Port 84 -->
      <!-- Port 85 -->
      <!-- Port 86 -->
      <!-- Port 87 -->
      <!-- Port 88 -->
      <!-- Port 89 -->
      <!-- Port 90 -->
      <!-- Port 91 -->
      <!-- Port 92 -->
      <!-- Port 93 -->
      <!-- Port 94 -->
      <!-- Port 95 -->
      <!-- Port 96 -->
      <!-- Port 97 -->
      <!-- Port 98 -->
      <!-- Port 99 -->
      <!-- Port 100 -->
    </resource>
  </product>
</productDescription>
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

</d:port>
<!-- Port 2 -->
<d:port name="studentDBNeed">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBNeedType"/>
    <!-- Non functional Type for the port -->
    <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
        <d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/machine1/webapp1/studentDBNeed/dataProduced"/>
        <d:attribute name="dataProduced" value="3" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
    </d:type>
</d:port>

<!-- Port 3 -->
<d:port name="authenticationNeed">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationNeedType"/>
    <!-- Non functional Type for the port -->
    <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
        <d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/machine1/webapp1/authenticationNeed/dataProduced"/>
        <d:attribute name="dataProduced" value="4" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
    </d:type>
</d:port>

</d:resource>

<!-- Component as a Resource 2 -->
<d:resource name="db1">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBType"/>
    <!-- Non functional Type for the port -->
    <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
        <d:attributeMapping name="studentDBService" ref="#studentRegistrationProduct/machine1/db1/studentDBService"/>
    </d:type>
    <!-- Port 1 -->
    <d:port name="studentDBService">
        <!-- Functional Type for the port -->
        <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBServiceType"/>
    </d:port>
</d:resource>
<!-- Non functional Type for the port -->
<#attributeMapping name="dataProduced">
    <#type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
        <#attributeMapping name="dataProduced">
            <#type name="dataProduced" value="2">
        </#attributeMapping>
    </#type>
</#attribute>
</#resource>
<!-- Component as a Resource 3 -->
<#resource name="webapp2">
    <#type name="type">
        <#portMapping name="authenticationService">
            <#type name="type">
                <#attributeMapping name="dataProduced">
                    <#type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
                        <#attributeMapping name="dataProduced">
                            <#type name="dataProduced" value="2">
                        </#attributeMapping>
                    </#type>
                </#attributeMapping>
            </#type>
        </#portMapping>
        <#portMapping name="userDBNeed">
            <#type name="type">
                <#attributeMapping name="dataProduced">
                    <#type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
                        <#attributeMapping name="dataProduced">
                            <#type name="dataProduced" value="2">
                        </#attributeMapping>
                    </#type>
                </#attributeMapping>
            </#type>
        </#portMapping>
    </#type>
    <!-- Port 1 -->
    <#port name="authenticationService">
        <#type name="type">
            <#attributeMapping name="dataProduced">
                <#type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
                    <#attributeMapping name="dataProduced">
                        <#type name="dataProduced" value="2">
                    </#attributeMapping>
                </#type>
            </#attributeMapping>
        </#type>
    </#port>
    <!-- Port 2 -->
    <#port name="userDBNeed">
        <#type name="type">
            <#attributeMapping name="dataProduced">
                <#type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
                    <#attributeMapping name="dataProduced">
                        <#type name="dataProduced" value="2">
                    </#attributeMapping>
                </#type>
            </#attributeMapping>
        </#type>
    </#port>
</#resource>
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.

<d:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
</d:attribute>
</d:resource>

<!-- Component as a Resource -->
<d:resource name="db2">
  <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#userDBType">
    <d:portMapping name="userDBService" ref="#studentRegistrationProduct/machine1/db2/userDBService"/>
  </d:type>
  <d:port name="userDBService">
    <!-- Functional Type for the port -->
    <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#userDBServiceType"/>
    <!-- Non functional Type for the port -->
    <d:type name="meteredPortType" ref="http://ieee.org/meteredPortTypes#meteredPortType">
      <d:attributeMapping name="dataProduced" ref="#studentRegistrationProduct/machine1/db2/userDBService/dataProduced"/>
    </d:type>
    <d:attribute name="dataProduced" value="2">
      <d:type name="dataProduced" ref="http://ieee.org/meteredPortTypes#dataProducedType"/>
    </d:attribute>
  </d:port>
</d:resource>

<d:connector name="studentDBConnector">
  <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#studentDBConnectorType">
    <d:portMapping name="studentDBNeed" ref="#studentRegistrationProduct/studentDBConnector/studentDBNeed"/>
    <d:portMapping name="studentDBService" ref="#studentRegistrationProduct/studentDBConnector/studentDBService"/>
  </d:type>
  <d:port name="studentDBNeed" ref="#studentRegistrationProduct/machine1/webapp1/type/studentDBNeed"/>
  <d:port name="studentDBService" ref="#studentRegistrationProduct/machine1/db1/type/studentDBService"/>
</d:connector>

<d:connector name="authenticationConnector">
  <d:type name="type" ref="http://sap.com/studentRegistrationImplTypes#authenticationConnectorType">
    <d:portMapping name="authenticationNeed" ref="#studentRegistrationProduct/authenticationConnector/authenticationNeed"/>
    <d:portMapping name="authenticationService" ref="#studentRegistrationProduct/authenticationConnector/authenticationService"/>
  </d:type>
  <d:port name="authenticationNeed" ref="#studentRegistrationProduct/machine1/webapp1/type/authenticationNeed"/>
</d:connector>
1.13 User Requirements Description

The following VDDL description specifies a cloud user’s requirements, which details the parameters for broker to collect when a standard notation is described for a “User Requirements Descriptions”. The following description however refers to the specific Student Registration software specifying user’s need, such as duration, type of service and amount of data to be input through the service.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<d:deploymentDescription name="http://ncl.edu/my/studentRegistrationAppl" xmlns:d="http://schema.arjuna.com/vddl">
  <d:resource name="studentRegistrationAppl">
    <d:type name="type" ref="http://sap.com/studentRegistrationApplTypes#studentRegistrationApplType"/>
    <d:attribute name="myServiceDuration" value="5"/>
    <d:attribute name="serviceDurationType" ref="http://ieee.org/useModelTypes#serviceDurationType"/>
    <d:attribute name="serviceOperation" value="http://sap.com/studentRegistrationApplTypes#studentRegistrationApplType\service">
      <d:type name="serviceOperationType" ref="http://ieee.org/useModelTypes#serviceOperationType"/>
    </d:attribute>
    <d:attribute name="dataInputAmount" value="100">
      <d:type name="serviceInputAmountType" ref="http://ncl.edu/my/studentRegistrationAppl">
        <d:attribute Mapping name="serviceDuration" ref="http://ncl.edu/my/studentRegistrationAppl/myServiceDuration"/>
        <d:attribute Mapping name="serviceOperation" ref="http://ncl.edu/my/studentRegistrationAppl/serviceOperation"/>
        <d:attribute Mapping name="dataInputAmount" ref="http://ncl.edu/my/studentRegistrationAppl/dataInputAmount"/>
      </d:type>
    </d:attribute>
  </d:attribute>
</d:resource>
</d:deploymentDescription>
```
Cost Estimation of Service Delivery in Cloud Computing,
at Arjuna Technologies Ltd.
Appendix B

1 Configuration, Deployment and Execution Manual

1.1 Pre-requisites

For further development of the brokering service application, the following prerequisites are necessary:

- A Java 6 SDK
- JRE System Library [JavaSE-1.6]
- JBoss Tools 3.3.0
- JBoss Application Server 7.1 (version: jboss-as-7.1.1.Final)
- Maven 2.0 build tool

1.2 Configuration

- Install JBoss Tools
- Create an empty RichFaces project
- Copy the src folder from “the brokering service web application project” and paste into the new project. Refreshing all src folder therefore, will create the packages and their classes accordingly.
- Confirm all dependencies are installed as mentioned in pom.xml maven based build file.

1.3 Deployment

- Run JBoss 7.1 Runtime Server
- Deploy the project from Package Explorer in Eclipse, on server by project folder -> Run As -> Run On Server

1.4 Execution

- Open a browser
- Access the web application through the URL of deployed location, for example http://localhost:8080/richfaces-webapp/
Appendix C

1 UML design

1.1 Cost Estimation Classes

The following classes are part of the cost estimation workflow. Only class operations and attributes are displayed in the diagram.