FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Masters' Degree in Informatics Engineering
Dissertation

CloudAid

Aggregation of Linked USDL Cloud Services using Multi-Criteria Methods

July 2, 2013

Jorge Araújo
jaraujo@student.dei.uc.pt

Advisors at DEI:
Jorge Cardoso
Catarina Ferreira da Silva
Paulo Melo

Abstract

In the past few years organizations have been turning to cloud as a way to reduce costs. Typically by outsourcing their non-core competencies. This brought a huge development to cloud services, leading to a crescendo amount of new providers, increasing competition, therefore, also increasing the functionality and capabilities of these services. This new paradigm poses a big challenge to organizations willing to adopt such services. *Which service to contract? Which are the best options in the market for my problem? What services can be combined to achieve my goals?* These are some of the questions a decision maker has to answer when searching for cloud solutions to his organization. Sometimes a single service is not capable of providing what the organization needs, in these cases a composition of several services is needed. This adds more complexity to the comparison and decision that has to be made.

However, this is no longer just a search for services that fulfill the functional requirements (supports JAVA, allows SSL, etc...) for a determined problem, and that can work together. It has become a matter of distinguish which of those services provides a better composed solution, having in mind organization goals. Therefore, having a greater importance given to the non-functional requirements (price, security, availability, etc...).

Describing a service and its requirements, namely the non-functional ones, is a challenge. Publishing this description in a way it can be easily discovered is another challenge. The already existing approaches like WSDL do not suffice since they do not describe the service business aspects. Service description languages like USDL or its semantic approach Linked USDL can supress this need for describing the business aspects. The amount of service features relevant for a decision also makes it very difficult to manually compare and choose the best composite solution. It has become much more than *the cheaper is the best* solution. Some organizations start to pay more attention to other details, security or portability for example, even if that has a price cost. This is why we think a decision aid process can help the person in charge by automatically comparing and presenting a recommendation on what are the best solutions, based on pre-determined constraints.

What we propose in this thesis is to provide methodology, methods and tools to help the decision maker during the process of search and choice of services for aggregating a composite solution.

Keywords: Cloud Service Composition, Cloud Service Aggregation, Linked USDL Service Description, Decision Aid, Multi-Criteria Decision Making

Acknowledgements

First, I thank the institutions that provided the means to achieve this result, Instituto Pedro Nunes for the scholarship provided and Faculdade de Ciências e Tecnologia da Universidade de Coimbra for all the infrastructure made available.

Furthermore, I would like to express my sincere gratitude to my advisors, Prof. Catarina Ferreria da Silva, Prof. Jorge Cardoso and Prof. Paulo Melo for their continuos support in this research thesis, for their patience to read all the artifacts, for the precious insights and discussions we had throughout the project and for the experience they provided in all scientific, professional and personal aspects of my final academic year. Certainly without their guidance nothing would have been accomplished.

I would also like to thank Dr. Carlos Pedrinaci and Dr. Torsten Leidig for their experience and fruitful discussions about Linked USDL as it was a fundamental part of the final thesis. But especially for the opportunity of working with them.

To all my fellow researchers and professors involved in the Genssiz: Center for Service Systems Research I also leave a special thank for the constructive feedback and ideas.

Also, a much special thank you to my family who always supported and encouraged me, specially my mother for her tireless patience. I also want to dedicate a special thought to my late father who would have loved to read this work.

Last, but not the least, i would like to apologise to my girlfriend and all my friends that endured my absence and heard so many "working on thesis" related excuses. I want to thank them for the heartfelt support in the tough moments and the motivation to keep going.

Jorge Araújo

Contents

1 Introduction

This chapter gives an introduction to the topics addressed in this thesis and is organized in five sections. The first, Section 1.1, gives the reader the background to the main topics discussed throughout the project, services, cloud services, service composition and decision aid. Since some of the concepts discussed in this thesis are of common use there is the need to assign a description regarding the scope of this thesis. These descriptions are presented in the second section, Section 1.2. The third, Section 1.3, introduces the motivation for the work done. In the fourth, Section 1.4, we have the description of the problem at hand where, besides the initial description, a small example to better understand the reality of the problem is presented. The fifth, Section 1.5, presents the objectives to achieve as well as the major challenges associated with the previous described problem. These challenges are linked to the objectives and are the main concern of our work. The sixth, Section 1.6, explains the chosen approach to achieve the above described goals and the reasons for choosing so. The seventh, Section 1.7, gives an overview of the project planning and explains the changes made during its execution. Finally, Section 1.8 explains the structure of this document.

1.1 Background

In order to contextualize the reader this section introduces the main topics discussed in this thesis.

In a community highly oriented to services, being capable of understanding what are the capabilities of the available services and later choose the best to fit our goals is of crucial importance. Not only to enrich our solutions, but also to allow more focus in the core competences by outsourcing non-core services. Therefore, the work presented in this thesis tries to solve some of the current problems regarding this matter and explained in this section. How to describe a service? Which service to choose? Which services can be aggregated to achieve my needs? These will be some of the questions answered in this thesis.

Section 1.1.1 explains what a service is and why this concept is so important for society and to our work in particular. Section 1.1.2 focuses on a specific kind of services, cloud services, and how they are introducing a new paradigm which

sees software and hardware as utilities. Section 1.1.3 gets us closer to the problem addressed by discussing service composition, which in this thesis scope, is the capability of aggregating several different services in a single solution. The goal is to create a solution to a particular problem not resolved by a single service. Finally, Section 1.1.4 addresses the decision aid topic.

1.1.1 Services

Since the beginning of human communities that services are present in everyday activities, even when goods were the mean for trade instead of money. In fact, everyone can give a simple example of what can be a service: repairing a car, gardening, cleaning a house, teaching, consulting; these are all services, tasks done by someone contracted by someone else. In [56] the author divides services in two categories, *services affecting goods* and *services affecting persons* and defines a service as "...a change in the condition of a person, or of a good belonging to some economic unit, which is brought about as the result of the activity of some other economic unit, with the prior agreement of the former person or economic unit...one economic unit performs some activity for the benefit of another...". Many other authors define service in similar ways (cf. [121],[122], [23]).

With the explosion of web technologies the service concept evolved into the technological spectrum to the so called web services [18]. The consequent demand for web services has radically transformed the way we look nowadays at software. The rise of architectures like SOA (Service Oriented Architecture) [43] or SOC (Service Oriented Computing) [13] caused services to be in the centre of a technological revolution. From the technological perspective, W3C defines in its glossary [1] a service "... is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requester entities. To be used, a service must be realized by a concrete provider agent".

The term Internet of Services (IoS) [115] refers to the infrastructure that enables the provision of universal services to consumers. The IoS describes an infrastructure that uses the Internet as a means for offering and selling services. This kind of provisioned services, through the internet, will be the focus of our work.

Note that the scope of this thesis does not only restrict itself to web services, since web services are only one dimension of the overall service concept.

The importance of the service concept in this thesis is of the upmost importance. A service will be the single element in our solution. This thesis will discuss how to describe and compare services, particularly cloud services, in order to attain the company needs for building its desired composed solution. Therefore understanding what is a service and what are its properties is very important.

1.1.2 Cloud Services

In Section 1.1.1 the concept of service and web service were introduced. However this is not the final step in delivering services. In the past few years a new way of

[1] http://www.w3.org/TR/ws-gloss/#defs

looking into the web and services emerged: cloud computing.

Some argued that Cloud is not a big change, Oracle CEO, Larry Ellison, referred to the cloud as "...we've redefined cloud computing to include everything that we already do.... I don't understand what we would do differently in the light of cloud computing other than change the wording of some of our ads". However, this is no longer the case, the consciousness change has been deeper, cloud advertises computation as an utility, like electricity or gas, and the cost reduction by using cloud services is greatly acknowledged, as it will be explained in Section 1.3.

Cloud has been the most recent strategy of all the major companies like Amazon, Oracle and Microsoft. Even smaller players are moving to the cloud. But what exactly is cloud computing? The National Institute of Standard and Technology (NIST) defines cloud computing as being a "...model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [84].

It is now possible to think in computing, storage or software as an utility that is provided on demand to anyone who wishes to use it. The most visible result is cost reduction since operational expenses with servers, software licenses or specialized personnel for maintenance are now migrated to the provider entity side. But cloud goes even further. By allowing to re-scale resources on demand, the cost reduction can be huge, once resources are no longer needed they can be released. An interesting description of cloud computing can be found in [10].

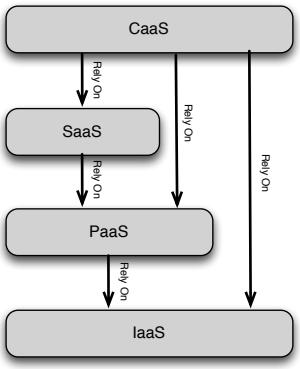


Figure 1.1: XaaS Service Models Dependencies

With this information in mind, cloud service can be defined as being a service that provides computing resources and fulfills the five fundamental characteristics introduced by NIST [84]: *on-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity* and *measured service*. In fact, from the cloud computing point of view, everything is considered a service, because every resource or task is provided by the terms explained in Section 1.1.1. This model known as XaaS (X as a Service, X meaning any kind of concept). In Figure 1.1 we define the dependencies

between cloud service models. All these models are already vastly discussed in the literature [84], [44], [74].

Since the service concept in general is too broad, we decided to focus on this particular kind of services: cloud services. The reason was the emerging growth that cloud services have been experiencing lately, as explained in Section 1.3. Therefore, from now on, when referring to services, unless otherwise explicitly specified, we mean cloud services from the XaaS model in Figure 1.1.

1.1.3 Service Composition

With today's high degree of specialization, companies can no longer be responsible for the entire supply chain. This is one reason why many companies are turning to outsourcing or sub-contracting. After the appearance of web services and, later, cloud services, companies can easily use other companies services to enrich their own offers and focus on their core competences. Standards like the Web Service Description Language (WSDL) [26] and transport protocols like the Simple Object Access Protocol (SOAP) [19] have greatly increased the use of web services by facilitating both their description and consumption.

The service composition concept is strongly connected to SOA and SOC approaches. As stated in [113] implementing a well-shaped SOA has one main benefit, "...the ability to compose new functionality out of existing services into so-called composite services, thus significantly increasing reuseability of existing services". In [81] the authors define that "the underlying principle in service composition is for a service provider to implement a new service reusing existing services as building blocks, and to add value to the sum of the parts". Lets then succinctly define service composition as an aggregation of services in order to obtain an added value resource or product.

During the last few years service composition topic as been subject for several studies and surveys, (e.g. [106], [86], [60], [123]), which explain many different approaches to this problem (e.g static and dynamic composition, automatic and manual, etc...). It is also considered by some authors that service composition is analogous to workflow management [41], [24]. A common concept to all these works however is: web service. Almost all the previous work has been around web service composition, and that is why most authors define it as analogous to a workflow. For example, the execution of one web service will influence the outcome of a second, a temporal constraint exists, a flow must be created to compose this web services. However, opposing to the above literature, the service composition concept in the purpose of our work is linked to different ideas:

- Considering services as single composable elements with their own functionality. This means not only from the technical point of view but also from the business point of view (service price, or security for example).
- Each composable element being a XaaS.
- In opposition to workflow techniques which specify a flow of execution, our service composition is based on constraints from these composable elements

(price cap or minimum of data storage capability for example), and it is not entitled to specify temporal constraints, such as for example, component A must execute before component B, or component A needs components B outputs to work.

Thus our final solution will be the aggregation of several composable elements (cloud services), resulting in a composite service. For that reason note that from now on, unless explicitly specified, composite service refers to this final composition solution which results form the aggregation of several composable elements.

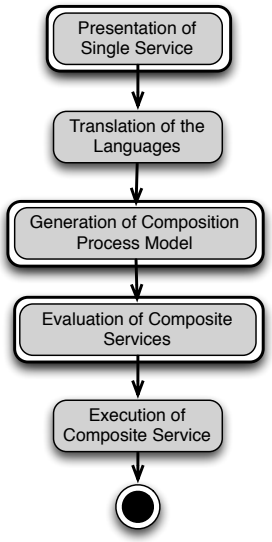


Figure 1.2: Service Composition Life-Cycle: The three stages highlighted are the focus of this thesis. (Adapted from [106]).

In [106] the authors define a service composition life-cycle in five stages: *Presentation of Single Service*, *Translation of the Languages*, *Generation of Composition Process Model*, *Evaluation of Composite Services* and *Execution of Composite Service*. Once again this life-cycle is intended for web services. However in our work we are attending to cloud services. This brings new requirements to the way we address the life-cycle. Take the example of service descriptions. Cloud services cannot be described solely by its technical aspects like a web service with WSDL, since they are usually not invoked in the same way. A web service is typically used by developers for integration purposes, under the SOA principles, usually within a controlled environment (within an organization or between partner organizations). Cloud services on the other hand, provide a wider service level, potentially usable by the entire internet community. Thus, the way they are described cannot be the same. In [48], the authors propose a solution to this particular problem. They try to extend existing open standards, like the DMTF's (Distributed Management Task Force) Open Virtualization Format standard, that addresses the problem of packaging and distributing virtual appliances (e.g. complete software stacks deployed in

one or more virtual machines).

Taking into account these five stages of service composition, adapted from [106] to relate to cloud services, in our work we focus on three: the first stage, *Presentation of Single Services*, where the services are described. For this matter, the service business aspects will have a key role in its description, rather than only focusing in service technical aspects from WSDL [26]. This makes all the composition process much more complex than the processes addressed in the literature [106] [41]; the third stage, where both the requirements and the service attributes are described in order to compile a viable composite service; and the fourth stage, *Evaluation of Composite Service*, where the composition is evaluated and a decision has to be made on which services to choose for the final solution. Figure 1.2 shows the composition life-cycle and the three stages addressed by this thesis.

More recently Composition as a Service (CaaS) has been introduced in [16] as "...a service that mediates communication between multiple clients. It provides composition recommendations to stakeholders and collects feedback from them". We will define CaaS as being a service recommendation system or application, provided by an agent (provider entity), to clients (requester entity), typically through an internet connection. As shown in Figure 1.1 CaaS is the top model that references the models bellow, optimistically a CaaS can reference any kind of service based on the requester entity preferences, however in a simplest way it can also be viewed as a SaaS, as it can be distributed as a software, based on SaaS assumptions.

1.1.4 Decision Aid

As any other business and management topic, services and service composition involve risk assessment and decision making. Some decisions have to be made and usually by a person (manager). Typically the decision is whether or not to use a specific service from other entity, according to the assumption on which is the best choice to the problem in hand. It is easy to identify a big decision problem here. Distinguish the best from all the available options in the market is sometimes very complicated if not impossible by a single person. The amount of constraints and characteristics of today's services is much more than simply the price. A manager needs to have this information in mind to make an adequate decision. This is the focus of this. Help this decision process by properly describing services, allowing them to be computable for comparison, in order to achieve a recommendation on which are the best services to aggregate and fulfill the identified requirements or enterprise goals.

While we can simply make a search in the web for a service which can satisfy our company needs, this is far from an optimal solution. Even if some satisfactory results are indeed found there is still the need to make an assessment on which is the best option. Thus, two problems while choosing a suitable service can be identified: finding all the possible candidates based on our needs and choosing which of the candidates is the most suitable also based on those needs or even enterprises goals.

With all this in mind it is not difficult to think on the benefits that a decision aid solution could bring to the manager who has to decide. Several different ap-

proaches can be used for this particular problem: Portfolio Analysis [118] that in our case could be used through service grouping and subsequent application of an evaluation function; the Matching [88] problem defined in economics and applied in informatics [105], which in our case could be used as a pairing mechanism, allocating services to required needs or tasks; Multi-Criteria Decision Making (MCDM) [46] which has a vast number of different methods for decision aiding and is especially adequate for decisions involving several characteristics or having user preferences in consideration; or simply Mono-Criteria. However, with Mono-Criteria, it would be very difficult to capture the decision maker preferences, or a problem where many different incomparable service characteristics exist, which is often the case in service elicitation.

From all these options MCDM seems the most suitable solution. It allows us to define service characteristics as key values for comparison, either if those characteristics are incomparable or perfectly defined. It also allows an easier mapping of the problem at hand as we will explain in Section 2.4. However, all the alternatives are considered and the problem will be further discussed in Section 1.6.3.

1.2 Concepts Definitions

Many of the concepts used in this thesis are of common use and may have different interpretation depending of the reader's background. Thus, the current section outlines key concepts and describes them accordingly to the thesis use. When needed some examples are also presented to ease their comprehension.

CSA

Composite Service Architecture (CSA) is the conceptual system the user is trying to build. This system is composed of different composable elements also called Service Templates, constraints or requirements and preferences. Basically the CSA is the result of the requirements analysis previously performed by the user about the desired cloud service aggregation. We define Composite Service Architecture as the set of Service Templates, Requirements and Criteria used to define the desired Composite Cloud Service to be built over an aggregation process.

Service Template

In this work we adapt the concept of Service Template introduced in [22]:

> "...a Service Template represents a structure or blueprint that the designer uses to indicate the characteristics of the Web service that is needed."

Although with a slight but important modification, in this thesis a Service Template relates to a blueprint of a cloud service instead of a Web Service. Thus the Service Template represents the conceptual idea of a cloud service in the CSA with all its requirements and criteria. For instance: A MySQL database with 1TB storage capacity is a Service Template example for defining a group of services that have MySQL and at least 1TB of storage capacity.

Requirement

As defined in [117] a software requirement is:

1. A condition or capability needed by a user to solve a problem or achieve an objective.

2. A condition or capability that should be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

3. A documented representation of a condition or capability as in 1 or 2.

In this thesis the requirements relate to the CSA, either to the system as a whole or to a specific Service Template and are the means for a user to specify a resource or capability need or a certain constraint regarding a certain resource or capability.

Exclusive Requirement

An exclusive requirement is a special type of requirement. These are used to limit the search spectrum. In other words an exclusive requirement eliminates all the services that do not fulfill the requirement. For instance: I want services with a minimum of 500Gb of Storage Capacity. This means that all the services that provide less that 500Gb of Storage Capacity will be discarded.

Global Requirement

A global requirement is a normal requirement, exclusive or not, with a global scope. This global scope means that instead of being related to a specific Service Template it is related to the entire system. All the Service Templates inherit this requirement. For instance: I want all my services to be hosted in Unix machines. All the Service Templates will have this requirement.

Alternative

The authors in [127] define alternative as "...different choices of action available to the decision maker." . In our specific case, an alternative is a service offering that fulfills all the requirements specified for the specific Service Template to which the alternative relates to. By other words, an alternative is a suitable candidate to fit the Service Template.

Criterion

A criterion is, as the authors in [127] state, a dimension from which the alternatives can be viewed. This means that a criterion is specific alternative parameter or characteristics that will be evaluated in the decision. i.e: When contracting a Cloud Storage Service a customer can define storage capacity, platform and price as criteria for his decision. This means that he will evaluate the system based on these three service features or resources.

Global Criterion

Global Criterion follows the same principle of global requirement. When a global criterion is defined all the Service Templates inherited this criterion. i.e: Price will be a global criterion in my system. This means that all the Service Templates will have a price criterion.

Attribute

Similar to Criterion, the Attribute concept is used in this thesis as the distinguishing point from a conceptual criterion (Price, Storage, CPU, Security,...) and the actual value of an alternative. An attribute is considered part of an alternative rather than the conceptual decision criteria. Its value is then considered the Attribute value. i.e: A customer defines Price as being a criterion. This means that all the alternatives will be evaluated by its price attribute. A particular alternative which costs €100, will have a price attribute value of €100.

Decision Weight

Some MCDM methods require that the criteria be assigned importances or weights. By other words they are the importance assigned to each criterion for the final decision calculation. As the Authors in [127] state, this decision weights are usually normalized to add up to one. i.e: when contracting a SaaS a customer defines its price as being the most important criterion, lets say with a importance value of 0.5. Then, he also defines the availability and the number of allowed users as decision criteria. However, both have a lower importance than the price, 0.3 and 0.2. These values are the decision weights of each criterion to evaluate.

Decision Method

A decision method is the process used to achieve a decision. In this thesis however, when we refer to decision methods we are actually referring to the the group of MCDM methods.

Service Offering

As stated in the Linked USDL web page [100], a Service Offering is "...an offering made by a gr:BusinessEntity of one or more services to the public or specific customers. It usually defines a price and terms and conditions including service level agreements.". It can be considered an instance of a service or a bundle of services. This definition is particular important when looking into Cloud Services, since it often happens that the final offering is composed of several service instances. Each instance has its resources or features chosen by the user before its contracting, but it can also happen that these service parameters (Storage, Processing capacity, Data Transferred,...) are define during the service usage (pay-per-use model).

Aggregated Solution

An aggregated solution is a specific combination of alternatives that fit the CSA specification for its Service Templates.

Typically a Service Template can have multiple alternatives, this means that several combinations of alternatives can exist for the same CSA specification. However, in order for a set of alternatives to be considered an aggregated solution all the Service Template must have a fitting alternative. The term "aggregated" comes from the principle of aggregating one alternative (which by themselves are already solutions for a specific Service Template) for each Service Template. i.e: In our CSA we have two Service Templates defined, ST_1 and ST_2, any combination of alternatives of the type: $[ST_1ALT_i\ ,\ ST_2ALT_i]$ would be an aggregated solution.

Note that an Aggregated solution is a potential candidate to be the final composite service to reccomend to the user.

Admissible Solution

While an aggregated solution ensures that all the Service Templates have a fitting alternative, the admissible solution ensures that any potential restrictions of particular alternatives are not violated. It also ensures that global prices are not violated when the different alternatives are aggregated. i.e: if in an aggregated solution with two Service Templates, ST_1 and ST_2 each of them with one alternative ST_1ALT and ST_2ALT, it may happen that ST_1ALT is not compatible with ST_2ALT because ST_1ALT only interacts with Oracle databases and ST_2ALT is a MySQL database. In this case we have an aggregated solution but not an admissible solution. In another example, we have the global price for the aggregated solution set to a maximum of €100, however, ST_1ALT costs €75 and ST_2ALT costs €50, using this two alternatives would violate the global price, hence, not being an admissible solution.

Alternative's Performance

The alternative's performance is the value calculated by the decision method and assigned to that specific alternative. This value is calculated based on the alternative attributes and the Service Template's criteria to which the alternative relates to.

1.3 Motivation

Companies have always been eager to reduce costs, so everything they can find to achieve this goal is worth a change. Cloud is precisely this change. Besides reducing costs and allowing the outsourcing of non-core competences as stated in Section 1.1.2, it can also improve the companies "green image" by reducing energy footprint.

According to a study made by Cisco in 2012 [27], cloud is growing and is here to stay. The study states that 90% of IT decision makers say that cloud is in their agenda, and 31% consider it as critical for their business, against 7% in 2011. Also the former fifth reason for adopting cloud in 2011, cost reduction, is in 2012, the number one reason elected by decision makers. For those still sceptic about cloud, 70% of the companies say that cloud has met or surpassed their expectations.

Even the European Union is turning to cloud as a way to reduce costs [31], [57] and "save" the environment at the same time. According to a survey conducted by

European Commission in 2011 [31], 80% of organizations reduce costs by 10-20%. Other benefits include enhanced mobile working (46%), productivity (41%), standardization (35%), as well as new business opportunities (33%) and markets (32%). As a final remark the study also states that cloud computing has an high growth expectation. To the environmental impact the study could not be more clear: "The unprecedented increase of data flow and processing of information over the Internet has a significant environmental impact through energy and water consumption, and greenhouse gas emissions. Cloud computing can help mitigate these problems thanks to more efficient use of hardware as well as, more specifically, by building data centers to use low-energy servers and green energy.".

With this data we can see that cloud is without any doubt a big opportunity, and as more and more companies turn to it, more providers and different services will appear.

Moreover, the vertical component integration is no longer the same as it was in the industrial revolution and outsourcing company's offerings non-core components is becoming a key factor for achieving specialization and differentiation. Take the example of producing a car. Nowadays a factory only assembles the components that arrive from different providers, one could be located in Asia and produces tyres, another one in Europe produces the electronic components, and so on. Even this providers can again assemble different components, as for rubber, plastic, etc. This means an enormous effort to identify and find the right provider for the right component. Of course this view also applies to services in general and cloud services in particular.

Therefore, it is important for decision makers to have tools capable of aiding them to choose which is the best options for their company needs.

Our motivation is to help providing these tools and methods as a mean to outsource non-core competences and improve business performance. Our hope is to allow companies to reduce their effort in adopting cloud as a "partner" in their business thus benefiting from its advantages.

1.4 Problem Description

Today a company that wishes to adopt cloud services, for whatever reason they see fit, still has a huge task at hand. Figure 1.3 shows the six steps a manager, or the person responsible, has to undertake in order to implement a composite solution in his company. Moreover, all this process, from identifying the company needs which can be fulfilled by cloud services, to the final integration of all contracted services into the company environment is typically manual, and could take a long time and still, not being able to find the best solutions. Figure 1.3 is a domain specific translation of the rational decision making model discussed in [110].

As stated in Section 1.3, we propose to introduce methods and tools to facilitate some of the manager decisions. Although not all the steps can be automated, most can surely be facilitated even if in the end the final decision is taken by the person in charge.

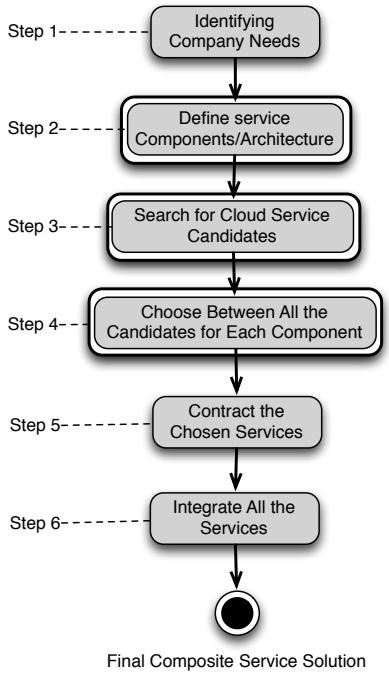


Figure 1.3: Steps for defining a composite service solution. The steps our work addresses are highlighted.

In Figure 1.3 three steps are highlighted, these are the steps addressed in this thesis: Step 2), *Define service Components/Architecture*; Step 3), *Search for Cloud Service Candidates* and Step 4), *Choose Between All the Candidates for Each Component*.

The other steps are not the scope of this thesis, since they do not enter in a service composition as we define it in Section 1.1.3.

Step 1, *Identifying Company Needs*, is by definition a manual process, although it can be aided by some tools to calculate metrics, cost or efficiency for example. Step 5, *Contract the Chosen Services*, in cloud services, it is mostly done by registering and configuring the service data in a web site or directly contacting the provider for further information or contract signing. Finally, step 6, *Integrate All the Services*, is a task usually done by a development team, sub-contracted or not. This integration has to do with the specific configuration to the target environment in the company, and so it is not covered in this thesis objectives as well.

From those covered in this thesis, in step 2, *Define service Components/Architecture*, the company defines which are the components (also called Service Templates, later in the process mapped to concrete cloud services) it wishes to implement and compose as well as its restrictions or special needs. This step is the formalization of the requirements elicitation phase started in step 1, and will produce an architecture of the overall system the enterprise wishes to build (CSA). This architecture is more similar to a requirements list for each composable element (Service Template), than an UML (Unified Modeling Language) component, class or deployment diagram.

For that reason from now on, unless explicitly specified, we will refer to this elicitation of requirements and elements as the CSA (Composite Service Architecture) definition.

In step 3, *Search for Cloud Service Candidates*, takes place a manual search for cloud services capable of fulfilling the requirements stated in step 2. It would be preferable to automate this search. Sometimes it is very difficult for a single person, or even a team, to find the right services or even to understand what the services are capable of doing. For this reason, a clear way of describing services and what they offer is necessary. Also a mechanism capable of searching these services based on list of requirements would also be advisable.

Finally, in step 4, *Choose Between All the Candidates for Each Component*, a comparison has to be made between all the found candidates (Alternatives) for each CSA Service Template. This comparison is quite simple if we have one or two characteristics to evaluate, however, as explained in Section 1.1.4, these kind of comparisons usually have several attributes which turn the decision into a more complex problem that must be facilitated by some automated decision aid solution.

Composite Service Contracting Example

In order to better understand the problem above described, we introduce a small example, with Figure 1.3 in mind and based on a simpler version of the use case defined in Chapter 3.

In our example a company wishes to start monitoring their headquarters energy consumption in order to reduce costs and follow the European Union energy efficiency standards [30]. Since the current "of the shelf" solutions are too expensive, the manager decides to create their own system with the purpose of cost reduction. With this in mind, and as explained in Section 1.3, the manager decides to use cloud services. The system to be implemented has several components each of them can be a separate cloud service (from the XaaS model), that all together will create the final composite service, the company energy monitoring system.

The manager identified some components and requirements that the system must fulfill. However, he does not know if these components can be allocated in the cloud and so he starts to search options for each one, always with the identified requirements in mind. He came to the conclusion that there are several options in the market, unfortunately, these services have too many variables for him alone to consider in order to find a viable solution for the final composite service. He is afraid to choose the wrong options leading to a bad business decision. With this in mind, a team is assembled with the sole purpose of analysing the market options and compile a final solution.

This team has the difficult job to find single services or other composite services in the web able to fulfill the requirements defined by the company. Along with this search task it is also required to do an analysis of pros and cons of each alternative for later comparison. After a few days, the services have all been found and can now be compared to achieve a final composite solution. A few more weeks later, after each of the composite solution elements has been decided, the contracting of

each service can begin for later integration and configuration within the company environment.

This is the normal process of today's cloud composition, what we propose are methods and tools to facilitate this process, hence helping the cloud adoption by smaller companies or startups as well as helping to reduce bad decision from the business point of view.

1.5 Objectives and Challenges

In Section 1.3 we presented the motivation and in Section 1.4 the problem at hand, with a small practical example. In this section we will discuss the top objectives for this thesis.

As mentioned before, our work will focus on three of the six steps for defining a composite service, presented in Section 1.4. In fact the objectives and challenges are a direct mapping of these steps. Bellow in Sections 1.5.1, 1.5.2 and 1.5.3 we discuss the objectives and challenges of the three steps: *Define service Components/Architecture*, *Search for Cloud Service Candidates* and *Choose Between all the Candidates for Each Component*.

1.5.1 Expressing the Composite Service Architecture

Objective 1: Capture all the Composite Service Architecture elements: Service Templates, Requirements, and Criteria. These three elements are fundamental to achieve a CSA definition capable of coping with both the search and decision processes. Service Templates represent the different services necessary for the composite solution. Requirements state the system constraints and needs. Criteria specify alternative parameters of characteristics and are a way to evaluate alternatives.

Challenge A: From Figure 1.3 we see that the second step, after identifying the company needs, is to define the required components to achieve the desired composite service, this process as previously stated is called CSA definition. This CSA is the combination of two or more Service Templates, each of them being a cloud service blueprint (see Section 1.2). As any architecture there are constraints and requirements that have to be fulfilled. These specifications should be provided by the company in order to advance to the next step (finding candidates for each component).

The challenge is therefore to create a data structure capable of storing all this information in both organized and machine readable way. The way this data is asked to the user must also be understood by him for sake of correct data insertion.

To describe the CSA, we should consider not only functional but also non-functional requirements. In fact, in [128] the authors say "non-functional requirements play a crucial part in decision making process for service composition. Consideration should be given on how to represent these non-functional requirements...".

1.5.2 Uniform Remote Access to Service Description

Objective 2: Achieve a uniform publishing and description of the service, capable of enhancing service discovery and comparability. Using semantic web and linked-data principles (see [15] for linked-data reference), it is intended to achieve a richer service description, by including non-functional aspects, as well as making all the underlying information available through remote access (Web), for both humans and software applications. These descriptions must allow the easy extraction of all the service properties (functional and non-functional) facilitating their property based discovery and comparison.

Objective 3: Allow a pricing description that can cope with cloud services pricing plans. This third objective has to do with the cloud service pricing challenge. The service description language, should have or be added a proper price model description, that match the new models introduced by cloud services. Again semantic web approaches can be of great value to address this objective, as it will be explained in Section 1.6.2.

Challenge B: In step 3 of Figure 1.3 a search has to be perform, however in order to search something we need to not only find information but also understand what is being searched. Searching cloud services is no different, we need to have a description of the service in order to know if that particular service fulfills the requirements or not. In our particular case, if the service is an alternative to be compiled in the final composite solution. A description language capable to describe several different service aspects (technical and business) and at the same time be understandable to both human and machine is a big challenge and a crucial point in this thesis.

Languages like WSDL focus on technical interfaces such as operations and data types, and do not give enough information about services, specially non-functional requirements. For a system to be able to aid a decision maker to choose between two services it must look into all service aspects, and for that, legal, pricing models, marketing strategies, and service quality levels need to be included in the descriptions.

Accessibility is another challenge. If a recommendation system wants to be reliable it needs both quality and reliable information. The cloud service industry is a vast territory and many providers exist, however it is not easy to access service descriptions in order to know exactly what they offer, at least not in an automatic way. By using languages built on top of linked-data principles, which can provide easy ways to manage data and publishing it by terms of RDF (Resource Description Framework), one can improve the service discovery, allowing an automated system to retrieve a much larger set of services in a machine understandable language.

As stated in Section 1.1.2, an important advantage of cloud, is the cost benefit, generally guaranteed through a *pay-per-use* model. This means that most cloud services have a much more complex price plan than, for example, a simple monthly subscription. Almost every factor influences the service final price, and these values

easily fluctuate with time. This requires a much more sophisticated price model description.

1.5.3 Complex Decision Making

Objective 4: Provide a mechanism able to compare and recommend the best services based on a list of Criteria. The aim is to provide a decision aid based on the input given in step 2, and extracted from the CSA. The final decision would always be taken by the manager. However, by providing a recommendation we would be improving the chance of success of the final decision. A decision could be made solely by excluding the services which do not fulfill the requirements, however this could not be the best solution specially when we look to the non-functional requirements, the user preferences, or other kinds of input that might prove preponderant. This information should be taken into account.

Challenge C: In Section 1.1.4 it was explained why a decision aid system would be a great advantage to companies willing to adopt cloud services. The amount and variety of offers is growing at a fast pace, and so are the service functionality, but not only. The clients demands are getting higher, today the differentiation is of the up most importance and companies that want to succeed have to carefully decide what they want and what they need from the cloud. All these aspects are difficult to achieve by manually searching cloud services in the hope of finding what we are looking for. Even if we could have all the services in one marketplace we would still need to search them all and decide which to use, the big challenge is how to do it.

Unlike the previous two challenges, the decision challenge is transversal to steps 2, 3 and 4 from Figure 1.3. It is also related to the two previously described objectives. This is because the decision should be based on the CSA defined in step 2 and the challenge in Section 1.5.1, and the service description challenges in Section 1.5.2. The former has all the requirements and criteria while the latter has all the service relevant information to be analysed an evaluated.

In step 2, while defining the CSA, one has to have in mind that the requirements being defined have to be later, in step 3, compared with the service descriptions in order to understand which of the services fulfill those requirements. This could be a trivial comparison, for example, if a service supports SSL[2] communication (binary comparison (y/n)), or it could be a more complex one, for example, the service price, that involves a variety of other factors (see Section 1.5.2). For this to happen, the service description should be understandable by a machine.

Finally in step 4, the recommendation is made based on the collected information of the previous two steps. This step must be performed by some established decision method. The reason for using decision methods is simple, while a company gives primacy to low cost solutions, another would not mind to pay for a costly solution if high security is provided. Thus, not all the requirements defined in the CSA have the same importance and they will surely influence the decision. Therefore, the

[2] http://tools.ietf.org/html/rfc6101

Criteria is an important tool to capture these user preferences and to reach a better solution based on the user needs.

1.6 Approach

This section will be dedicated to the approach taken to achieve the objectives proposed in Section 1.5.

The service aggregation topic we are addressing, as already explained, is different from what has been done in the past. Our approach to this particular service composition is based on cloud services (Service Templates), requirements and decision criteria, and not on web services and workflows.

The process is divided in three major parts: first, the definition of the CSA with all its elements; second, the property-based service discovery, based is service description with not only technical aspects addressed by WSDL, but also in business aspects addressed by USDL (Unified Service Description Language) [20]; finally, the decision aid process for comparing and recommending the best alternatives for each Service Template in the CSA, based on MCDM methods. Needless to say that all these three parts are linked together through the data they use and produce. Note also that these three parts are mapped directly from the objectives and challenges in Section 1.5.

These three parts are described in Sections 1.6.1, 1.6.2 and 1.6.3.

1.6.1 Composite Service Architecture Modeling

As explained in Section 1.5.1 the definition of the CSA is an important task. All the relevant information about the system to be built is contained within the CSA, thus all the process herein described depends on how this data is structured. Our approach does not propose to be state of the art, it rather aims to fulfill the objective stated in Section 1.5.1.

Many different approaches could have been used, patterns [128] or blueprints [93] [92] are only two options. Many others exist based on representation models like UML (Unified Modeling Language) or FMC (Fundamental Modeling Language) [61]. In our work, we will not restrict ourselves to any of these approaches as we will use concepts and methodologies from more than one.

The final approach although not graphical, thus not using most of the representation models theories, is a data structure specially built to hold all this information. However, this structure no only stores data but also its relationships and affectation. A detailed description of this data structure is presented in Section 4.2.4.1.

1.6.2 Linked-USDL Service Description

Linked-USDL as described in [21] is a version of USDL [20], based on semantic web and linked data principles developed to achieve a wider global acceptance than his predecessor (USDL). The authors define the goal of Linked-USDL to "develop an ontology to represent services by establishing explicit ontological links to other

existing ontologies emerging from Linked Data initiatives", and also defines three reasons for this initiative:

- Retain the necessary simplicity for computation as well as for modeling purposes.

- Reuse existing vocabularies to maximize the compatibility of related systems, reusability of previously modeled data, and reduce engineering efforts to build complex models.

- Provide a simple, yet effective, means for publishing and interlinking distributed data for an automatic computer processing.

This language is being developed to describe services according to the USDL specification, however one of the goals is to keep it simple in order to achiever a wider acceptance. It's ability to be extended, since it is based on ontologies, can later be used to cope with different domains if that is the wish. To the moment a few modules have been developed: *Core*, which has main properties, like service name and provider; *SLA*, for describing service level agreements of the service; *Pricing*, which describes the pricing information of the service; *Security*, for describing security features of the service.

If we go back to Section 1.5.2, we can see the link between the challenges and the Linked USDL goals listed above. With the use of this service description it is intended to achieve a easier remote access to a more complete and sophisticated service description with more than just operational and technical interfaces. In fact, it is intended to focus in the business aspect of the service, where the non-functional requirements are fundamental to distinguish two similar services.

The approach will be to use this service description language to model several cloud services and assert if it can be used to define all the criteria needed for both, the decision aid process and the CSA requirements definition. This language will be the bridge between the raw service information and the truly important information in cloud service composition.

As mentioned in 1.5.2, the price models are an important feature of cloud services and so, some extensions to the already existing Linked-USDL modules should be made in order to cope with this requirements. With this in mind this thesis will collaborate with Linked USDL initiative in the development of the pricing description, enhancing the overall service description.

As an example of the capabilities of this service description specification, in Appendix A, a tutorial on how to model a service with Linked-USDL is presented. Based on a business intelligence cloud service, called BIME [29], this example was the first step taken in our work towards Linked-USDL service description. Its purpose was to introduce and experiment with this service description language to gauge its capabilities for later describing other services. This way we also encourage other researchers to use it as a service description language.

1.6.3 Multi-Criteria Decision Making

In Section 1.1.4, some potential approaches were introduced, now it will be explained which one was selected and why.

Multi-Criteria Decision Making (MCDM) is the approach chosen, mainly because it offers a number of different methods to address the multiple criteria kind of decisions. As said before, choosing a cloud service has several decisive factors and not only the price, so we have multiple criteria to analyse and weight before reaching a decision. This makes the mono-criteria approach less useful. The other two approaches, Portfolio Analysis and Matching were both valid, and can be used. The difference would have been in the problem modeling. The easiest and more direct way to model the problem is achieved with MCDM, since the criteria are mapped from the service requirements stated in the CSA almost directly. After defining this criteria, we can apply any kind of MCDM method, allowing for different experiments with different decision methods. We can state that MCDM provides more freedom of choice in controlling the way we want to conduct the decision aid process, allowing us to test different methods and constraints, such has user preferences.

The initial plan is to incrementally increase the method complexity and the different levels of input given by the user to reach a decision. The plan was devised to contain three phases:

1. a simple method will be used to assert the quality of the criteria and how to model the decision problem. The method used is the Simple Additive Weighting (SAW) [46];

2. experiments with a more complex version, allowing us to compare and check if more or different decision criteria are needed. The method used is the Analytic Hierarchy Process (AHP) [114];

3. a more complex approach like the outranking or fuzzy methods, this last phase would allow us to deal with incomplete information and incomparability.

Note that all these methods will be further discussed in Section 2.4.

Figure 1.4 shows a summary of what have been discussed so far, and maps the six step model for defining a composite Service solution to this thesis proposed approach. As shown in the figure the challenges are a direct match of the three steps highlighted, and the solutions correspond to the tools an methods used to address these challenges.

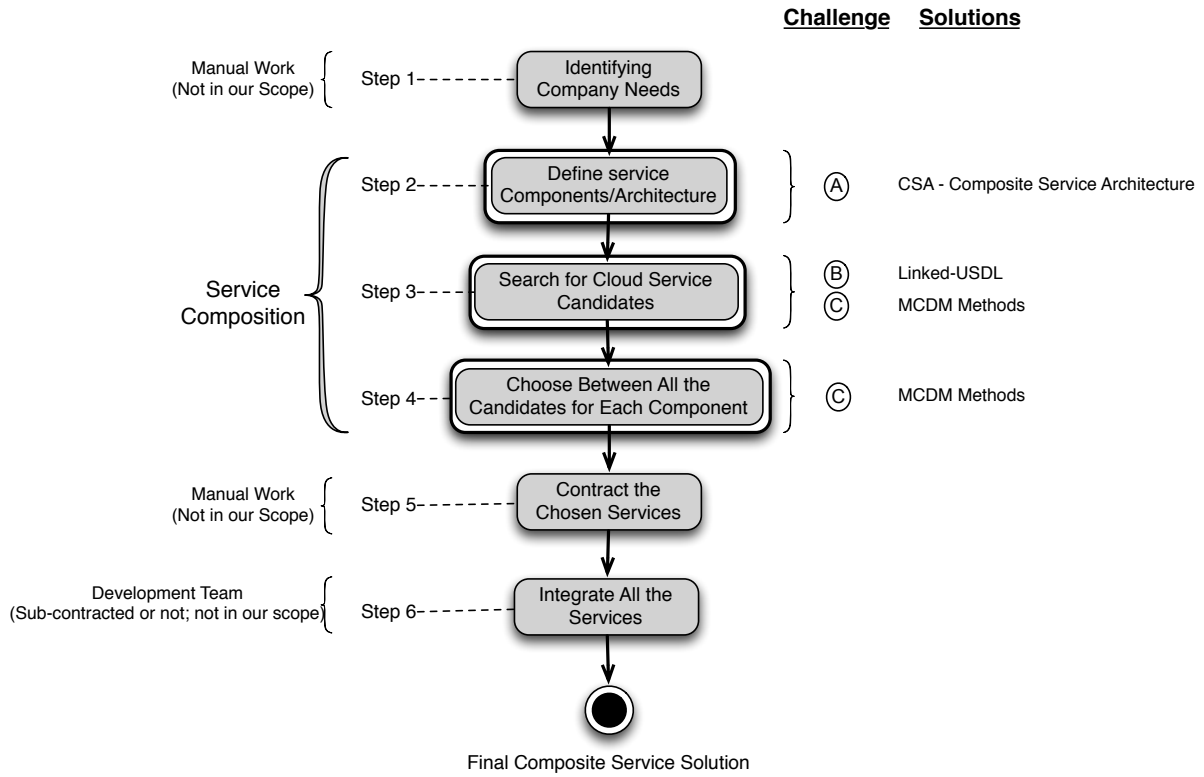


Figure 1.4: Summary of the topics addressed in this thesis mapped to the six steps for defining a composite Service solution.

1.7 Scheduling

In this Section the project planning is discussed. Also some major changes and decisions are pointed out as being crucial to the overall result of the project.

As any project, planning is of crucial importance and for that reason an initial plan was designed taking into account four major steps: Research/Literature, Prototype/Development, Testing/Evaluating and Report/Final Thesis. Figure 1.5 shows exactly these four steps. For obvious reasons, like the lack of knowledge about the study field and being a research/investigation project, this first plan was not entirely accurate. However, it was important to understand where the effort should be focused as well as to measure progress.

As time went by also our view of the problem and the possible solutions changed. This was the reason for the majority of the changes throughout the project.

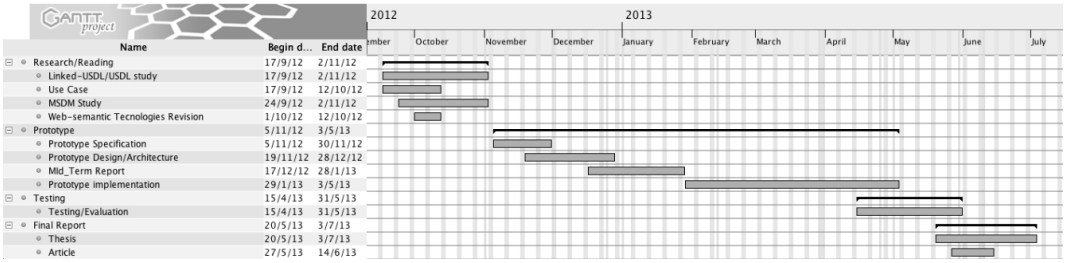
The first step was research, both of the technologies available and the possible paths to be taken for prototyping. With this in mind it is not difficult to understand that this first process brought some changes to the initial plan. One of them was the choice for focusing in a specific scenario, cloud services, instead of the initial broader vision. This decision was made based on the use case developed, also another task included in the intermediate plan in Figure 1.6, as it was believed that this new approach could bring more value to the final result since cloud services are an emerging concept.

Later on with the research about service descriptions we also came to the conclusion that pricing in cloud services was a big issue and decided to address this matter by improving the linked-USDL description for pricing. Again changes to the initial plan had to be made.

Obviously during the research period many more challenges were found, some of them had to be resolved with a time cost (postponing tasks), others were simply abandoned. The point is that research is a iterative process which leads to more tasks appearing ahead. As shown in Figure 1.6, many new tasks have been introduced, and some deadlines have been postponed. Some of the new tasks are specializations of the initial plan, which was a generic plan with a broader view of what should be happening.

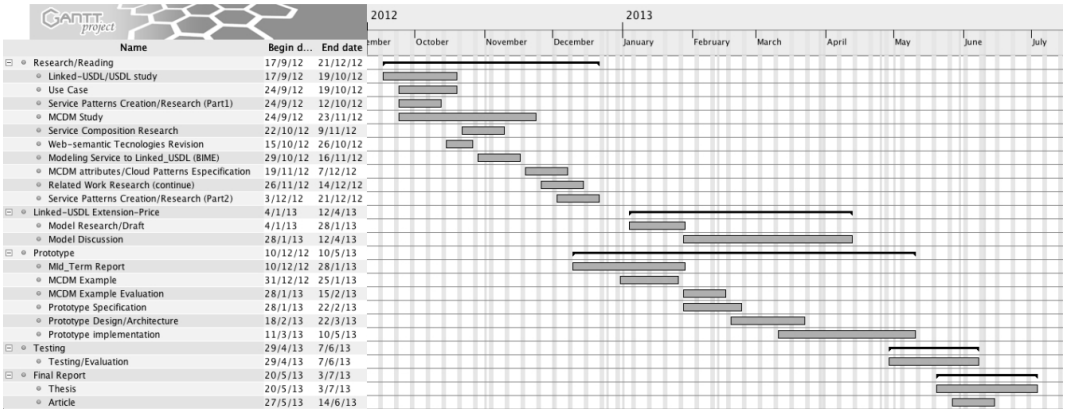
Moreover, after analysing some of the possible problems that could arise, especially in concern of MCDM, it was decided that the overall project would benefit from a simple implementation of an example, using some of the work done until then and applying a simple MCDM method. Also, the change of the initial plan for drafting a price model had to be postponed for time reasons concerning the report writing. This new group of changes and decisions are also reflected on Figure 1.6.

Finally, Figure 1.7 shows the last planning reorganization. This time the restructuring was due to implementation paths taken, as the Cloud Ontology research or the Service Set Generator implementation had to be implicated in the process. It was also postponed the initial plan of writing a paper, this one for time constraints.



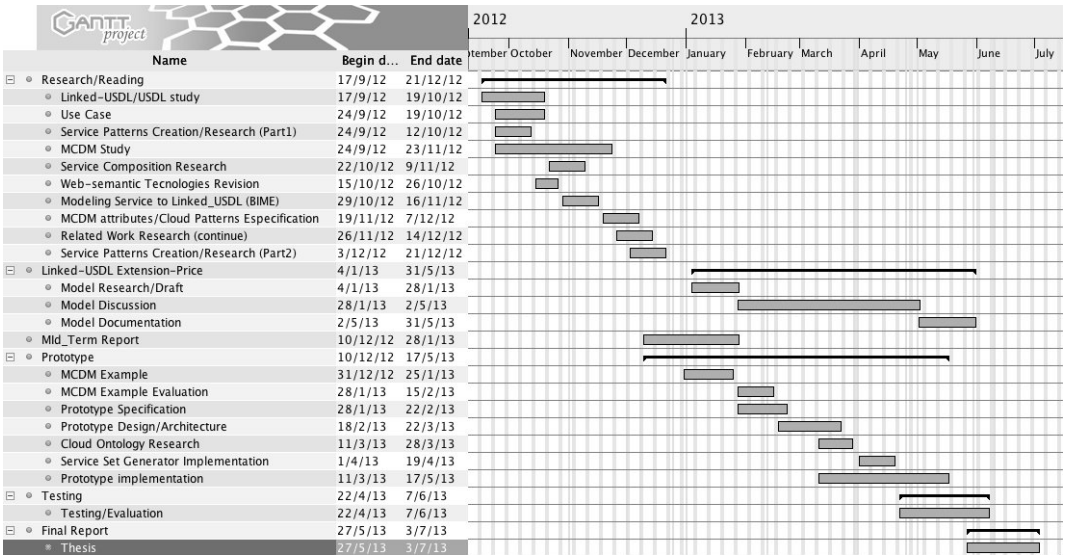
Name	Begin d...	End date
Research/Reading	17/9/12	2/11/12
Linked-USDL/USDL study	17/9/12	2/11/12
Use Case	17/9/12	12/10/12
MSDM Study	24/9/12	2/11/12
Web-semantic Tecnologies Revision	1/10/12	12/10/12
Prototype	5/11/12	3/5/13
Prototype Specification	5/11/12	30/11/12
Prototype Design/Architecture	19/11/12	28/12/12
Mid_Term Report	17/12/12	28/1/13
Prototype implementation	29/1/13	3/5/13
Testing	15/4/13	31/5/13
Testing/Evaluation	15/4/13	31/5/13
Final Report	20/5/13	3/7/13
Thesis	20/5/13	3/7/13
Article	27/5/13	14/6/13

Figure 1.5: Project Initial Planning (September 24th)



Name	Begin d...	End date
Research/Reading	17/9/12	21/12/12
Linked-USDL/USDL study	17/9/12	19/10/12
Use Case	24/9/12	19/10/12
Service Patterns Creation/Research (Part1)	24/9/12	12/10/12
MCDM Study	24/9/12	23/11/12
Service Composition Research	22/10/12	9/11/12
Web-semantic Tecnologies Revision	15/10/12	26/10/12
Modeling Service to Linked_USDL (BIME)	29/10/12	16/11/12
MCDM attributes/Cloud Patterns Especification	19/11/12	7/12/12
Related Work Research (continue)	26/11/12	14/12/12
Service Patterns Creation/Research (Part2)	3/12/12	21/12/12
Linked-USDL Extension-Price	4/1/13	12/4/13
Model Research/Draft	4/1/13	28/1/13
Model Discussion	28/1/13	12/4/13
Prototype	10/12/12	10/5/13
Mid_Term Report	10/12/12	28/1/13
MCDM Example	31/12/12	25/1/13
MCDM Example Evaluation	28/1/13	15/2/13
Prototype Specification	28/1/13	22/2/13
Prototype Design/Architecture	18/2/13	22/3/13
Prototype implementation	11/3/13	10/5/13
Testing	29/4/13	7/6/13
Testing/Evaluation	29/4/13	7/6/13
Final Report	20/5/13	3/7/13
Thesis	20/5/13	3/7/13
Article	27/5/13	14/6/13

Figure 1.6: Project Intermediate Planning (December 20th)



Name	Begin d...	End date
Research/Reading	17/9/12	21/12/12
Linked-USDL/USDL study	17/9/12	19/10/12
Use Case	24/9/12	19/10/12
Service Patterns Creation/Research (Part1)	24/9/12	12/10/12
MCDM Study	24/9/12	23/11/12
Service Composition Research	22/10/12	9/11/12
Web-semantic Tecnologies Revision	15/10/12	26/10/12
Modeling Service to Linked_USDL (BIME)	29/10/12	16/11/12
MCDM attributes/Cloud Patterns Especification	19/11/12	7/12/12
Related Work Research (continue)	26/11/12	14/12/12
Service Patterns Creation/Research (Part2)	3/12/12	21/12/12
Linked-USDL Extension-Price	4/1/13	31/5/13
Model Research/Draft	4/1/13	28/1/13
Model Discussion	28/1/13	2/5/13
Model Documentation	2/5/13	31/5/13
Mid_Term Report	10/12/12	28/1/13
Prototype	10/12/12	17/5/13
MCDM Example	31/12/12	25/1/13
MCDM Example Evaluation	28/1/13	15/2/13
Prototype Specification	28/1/13	22/2/13
Prototype Design/Architecture	18/2/13	22/3/13
Cloud Ontology Research	11/3/13	28/3/13
Service Set Generator Implementation	1/4/13	19/4/13
Prototype implementation	11/3/13	17/5/13
Testing	22/4/13	7/6/13
Testing/Evaluation	22/4/13	7/6/13
Final Report	27/5/13	3/7/13
Thesis	27/5/13	3/7/13

Figure 1.7: Project Final Planning (March 15th)

1.8 Document Structure

This document is organized in the following manner: Chapter 2 makes an overview of the related literature, comparing and contrasting it to our work. It also clearly states our scope regarding the three main topics discussed in this thesis, *Service Composition*, *Service Description* and *Decision Aid*; Chapter 3 presents the use case that will work as both, guidance and example of the work done in this thesis; Chapter 4 all the CloudAid prototype analysis and specifications is discussed, including requirement elicitation, prototype architecture and test plan; Chapter 5, presents the two semantic models produced in this work: the CloudTaxonomy, an ontology for cloud concepts; and the Linked USDL Pricing Module, an extension to the Linked USDL service description to cope with modern service price models; Chapter 6 presents all the CloudAid prototype related topics focusing in an high level description of its modules and describing the main decision taken during the development phase; Chapter 7 presents and discusses the results collected in the prototype test and evaluation phase drawing some conclusions; finally, Chapter 8 makes a summary of the thesis work and discusses the overall findings and its main contributions as well as the future developments and research potential.

Note that some chapters come from the intermediate thesis report. Among these Chapters are the Related Work, Chapter 2 and the Use Case, Chapter 3. Although they could have been removed from the main document, they are kept for coherence reasons and for those that did not had access to the previous document. However, a new section was introduced in the Related Work chapter. This new section, copes with the new project scope of Service Aggregation and was necessary to make the bridge between the Service Composition and the Service Aggregation concepts. The latter is now the focus of these thesis as a means to achieve Service Composition. Nevertheless, the former keeps most of its importance and for that reason there is no point in removing its documented research.

2
Related Work

This chapter reviews the literature, comparing and contrasting different approaches to the problems of service description, composition, aggregation and decision aid identified in Chapter 1. The objective is to identify the similar works and how our work can distinguish itself. Three main topics will be addressed in this chapter, each one represents a concrete challenge defined in Section 1.5. In Section 2.1, we analyse the work that has been done at describing a service. In Section 2.2, service composition is addressed and in Section 2.3 the Service Aggregation as a means to achieve the composition is discussed. Finally, Section 2.4 discusses decision aid.

2.1 Service Description

In this section the service description topic is addressed and some of the current solutions and their problems to describe services. We start by presenting the scope of service description to our work and then discuss languages and frameworks used to describe services. Finally, we present our perspective on what has been done concerning a particular important problematic challenge for this thesis: price description.

2.1.1 Our Scope Regarding Service Description

A service description can be explained as a mean to capture service semantics, i.e, all the relevant information about a service. We can argue if it is possible or not to achieve a complete service description. However, that is not the scope of this thesis. Instead, we will focus on what has been done to achieve a service description capable of compiling a broader vision of services than simply its technical specification and extend it with the necessary information about business and other aspects, to achieve our objectives.

In [20] and [23], the authors define three service perspectives:

1. **Technical** - Allows specifying technical information of services exposed by an organization. This perspective is divided into two main sections: invocation and execution.

2. **Business** - Usually describes non-functional properties that are fundamental for the characterization of a service, for example, price and security.

3. **Operational** - Describes the operations executed by services. It provides an understanding of what a service is providing from an operational perspective and, thus, what a consumer can expect from a service.

Of these three perspectives, the technical is the one who has known more developments. Languages like WSDL [26] and SOAP [19] aim at describing and invoking these technical aspects and are seen as technological entities or interfaces. However, we still miss the business perspective and its aspects: service quality, service level, economic and legal aspects, among others that will be needed in order to distinguish two similar services. This information is crucial when deciding which services to use to construct a CSA. For example, should the cheaper service or the one who provides more data security be selected.

2.1.2 Web Service Description Language

The Web Service Description Language (WSDL) was developed with the purpose to describe the technical details of accessing a Web service. Mainly it provides information about the technical requirements to invoke a service from the developers point of view (data types, method names, etc,...). It is therefore easy to understand the limitations of this description for the IoS and cloud services in particular. No information about pricing or other non-functional requirements is included. Therefore, we have a limited, yet powerful tool for describing a particular perspective of a particular kind of services, web services.

IoS has a list of requirements much broader than the ones described by WSDL. The business and operational perspectives have a key role in a service description, aspects like price or security have to be described.

2.1.3 Unified Service Description Language

The Unified Service Description Language (USDL) [20] initial version was ready in 2009 and it was later renamed to α-USDL. A second version was added to a W3C incubator group and was finalized at the end of 2011.

As stated in Section 2.1.2, the IoS has a broader description requirements list, for that reason USDL bridges three service perspectives, business, operational and technical. The language aims to model service concepts and properties such as service level, pricing, legal aspects, participants, marketing material, distribution channels, bundling, operations, interfaces, resources, etc, with the ultimate purpose of providing a comprehensive view on services.

This integrated service description view calls for a paradigm shift from the Web Services and SOA service descriptions which only focus on development and technical aspects.

The technical perspective was influenced by WSDL, the Web Service Modeling Ontology (WSMO) and Semantic Markup for Web Services (OWL-S) [79].

2.1.4 Semantic Approaches

As the author states in [71], "Web services extend the Web from a distributed source of information to a distributed source of services. Semantic Web has added machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information. In a similar way, Semantic Web concepts are used to define intelligent web services, i.e., services supporting automatic discovery, composition, invocation and interoperation. This joint application of Semantic Web concepts and web services in order to realize intelligent web services is usually referred as Semantic Web Services". The overall goal is to allow a better capacity for reasoning with services capabilities. This reasoning and easiness of publishing and discovery of services is supported by linked-data principles [15].

Some frameworks for semantic web services exist, and some work has been done to describe services with the aid of semantics like the Semantic Annotation for WSDL (SAWSDL) or Semantic Markup for Web Services (OWL-S).

SAWSDL

The Semantic Annotation for WSDL (SAWSDL) [69], tries to introduce semantic annotation into the WSDL specification. As already explained WSDL is a mere syntactic description of the service technical perspective. This approach tries to address the lack of reasoning capacity using logical inference. The specification became a W3C recommendation in 2007, however the authors in [64] state some limitations, among them the limited software support compared to other specifications like OWL-S and WSMO and as the author says, being "a mere syntactic extension of WSDL, without any formal semantics".

OWL-S

The Semantic Markup for Web Services (OWL-S)[79] is based on the W3C standard ontology OWL [82].

This ontology which imports WSDL concepts, goes a little further as it describes more than simply the technical aspects from the WSDL in a semantic point of view. In fact, only one of the three parts of the specification focus on WSDL concepts. These three parts as shown in Figure 2.1 are:

- the *Service Profile* is meant for describing what the service does, with the purpose for advertising and discovering services. It includes service functional parameters like, inputs, outputs, preconditions and effects, as well as non-functional requirements, for example, service name, quality rating and information about the provider. However with the release of OWL-S 1.1 the functional parameters started to be defined in the *Process Model* and referenced through the *Service Profile*. This allowed the *Service Profile* to focus almost exclusively in the non-functional requirements, therefore more related to the business perspective;
- the *Process Model* that gives a detailed description of a service's operation and how to orchestrate one or more services;

- the *Grounding* that provides the logic and XML-based service definitions in order to execute the service. This particular part is where WSDL was an inspiration, in fact it is used as an example in order to connect OWL-S to an existing standard.

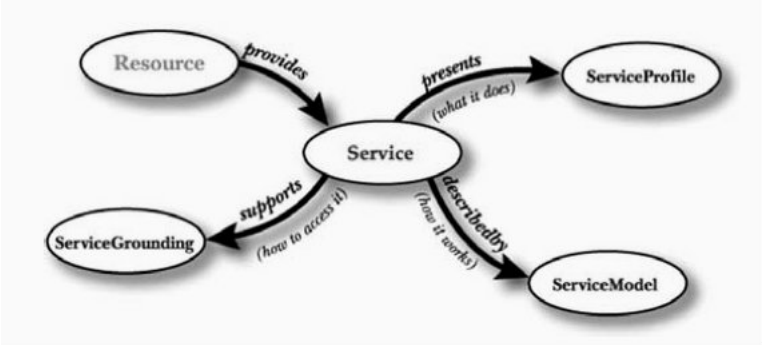


Figure 2.1: OWL-S service description elements. From [64]

Being an ontology it allows the specification to be extendable. Some work as been done in this matter, for example in [131], the author proposes a process model supporting Artificial Intelligence Planning techniques for automatic web service composition called SHOP2. The author in [83] uses GOLOG language to improve the service composition aspect. In [91], the author gives an extensive usage of the DAML-S (former version of the OWL-S specification) in the literature, and points some of the topics extended.

However, the OWL-S has some limitations. As the author states in [64], the limited expressiveness of the service description that is based in OWL-DL logic [82], since it does not cover the notion of time and change it allows only to describe static and deterministic aspects of the world. Also, being built around web services narrows the possibility of application to the IoS broader scope of the service without extra extension to the specification. In spite these limitations, OWL-S is a well founded base for a more complete service description capable of attaining the three perspectives for IoS.

WSMO

The Web Service Modeling Ontology (WSMO) [111] is a conceptual model that offers a Web Service Modeling Language (WSML) for describing semantic web services. This specification evolved from the Web Service Modeling Framework (WSMF) [45] after several European funded research projects in the semantic web services domain.

This conceptual model offers four components, all of them described in WSML, as stated in [32]:

- Ontologies - provide formal and explicit specification of the vocabularies used by the other modeling elements. Such a formal specification enables the automated processing of WSMO descriptions and provides background knowledge for Goals and Web Service descriptions;

- Goals - describe the functionality and interaction style from the requester perspective;
- Web Services - specify the functionality and the means of interaction provided by the Web Service;
- Mediators - connect different WSMO elements and resolve heterogeneity in data representations, interaction styles and business processes.

WSML was designed for describing the service capabilities as in pre-conditions, post-conditions, effects, assumptions (what has to be the world state before service execution) that consist of logical expressions.

Once again, this specification has some problems, one is the lack of use for OWL standards such as RDF in ontology specification as stated in [64]. Another problem is the lack of connection to other web service standard, WSDL. However, an attempt to resolve this issue has been developed more recently with WSMO-lite that proposes extensions to the SAWSDL specification [129]. Finally, the scope for IoS is still not achieved as of the WSMO focus is, once again, intended to web service.

SWSF

Other initiatives have been followed Semantic Web Service Framework (SWSF) [11], for example, defines another semantic web service language (SWSL) and an ontology (SWSO) [12]. It uses FLOWS, a first-order logic ontology, to describe web services, and ROWS, an ontology of rules. The advantage of SWSF is the use of a first-order ontology which allows more expressiveness in the service description compared to OWL-DL, used by OWL-S for example. However, the problem of first-order logic is that is not decidable thus it is difficult to use with a current inference engine. Therefore, this approach is not yet sufficient for the general IoS use.

2.1.5 Linked Services

As we saw the semantic web services approach has some limitations and was not able to properly disseminate the use of services in the web as expected. Recent research as pointed to a new approach in service descriptions, the author in [98] presents it as "Linked-Services". According to the author "Linked Services are services described as Linked Data. Therefore, these are service descriptions whereby their inputs and outputs, their functionality, and their non-functional properties are described in terms of (reused) light weight RDFS vocabularies and exposed following Linked Data principles.". This is a different approach on how to describe services which could not only allow a more complete service description by using domain specific vocabularies, but also facilitate their publishing and discovery process making use of the Linked Open Dada.

Based in these principles the author in [21] presents a service description language focused in attaining the above characteristics, Linked-USDL.

The term Linked-USDL presents exactly the connection it has to the linked data and linked services, and is a direct evolution of the USDL, despite of being a simpler and less complex version. In fact it is an attempt to overcome some of the original

USDL problems as well as to achieve a wider use than its predecessor. Extending it to accept linked data concepts the Linked-USDL tries to resolve the extensibility problem of the original model. Although there is not much literature about Linked-USDL, since it is still under development, the overall idea as the author say is "to develop an ontology to represent services by establishing explicit ontological links to other existing ontologies emerging from Linked Data initiatives". Using this methodology virtually any service perspective could be described. As the author uses it to model service networks, in our work we intend to use it for cloud service business descriptions.

As an effort to show the potential and the advantages of linked data for publishing service descriptions and potentiate its discovery and use, in [103] the authors proposes a platform to achieve this goals, called iServe. As the authors define, iServe "is a novel and open platform for publishing semantic annotations of services based on a direct application of linked data principles to publish service annotations expressed in terms of a simple vocabulary for describing services...". Other works follow the same line and try to apply Linked Data principles to the service annotation, discovery, search and invocation, [99], [40] and [135].

2.1.6 Service Description Pricing Component

As presented in 1.5.2, one of the objectives is to capture the pricing model of cloud services, which is one of its big advantages. According to the author in [96] the price gets into the non-functional property models, and not many work has been done to capture it in the service descriptions mentioned above. The exception is USDL who in fact has this property in mind while describing the business perspective. Being built on USDL concepts, Linked-USDL also has the price in consideration, however this description language as one more advantage, since it is based on linked data principles it can easily adopt other ontologies in its specification and some price ontologies already exist. *GoodRelations* [55] is an example of an e-commerce ontology for describing service offerings that among other properties already defines some price components.

Therefore in this thesis will be presented a extension to linked-USDL in order to contemplate the service price for the IoS as stated in the Objective 3 in Section 1.5.2.

2.2 Service Composition

In this section we analyse some of the work done in the past concerning Service Composition. However, since the area is vast we start by introducing the scope for our work. Then some of the most complete surveys about the topic are presented and discussed.

2.2.1 Our Scope Regarding Service Composition

First of all, it is important to outline the difference between two related yet different concepts: Mashup and Service Composition. As the authors in [76] mention, "Service composition is usually performed on the business logic layer, and results in executable workflows or plans that fulfill certain requirements of the new Web application. Mashup is an innovative way to develop Web applications by syndicating contents, data and functionalities from distributed sources on the Web. Different from service composition, mashup can be carried out on layers ranging from data to presentation. Rather than enacting a predefined workflow,...". In fact, several other authors describe Service Composition as very similar to a workflow [106] [41]. However, we are not interested in this view of service composition but rather in the composability of cloud services. The actual execution flow, or workflow is not as much important as the added value extracted from the composition of several service. Meaning that we are not interesting in how will the services actually interact but rather in finding suitable candidates for the building blocks of the composed service.

The other important remark is that, as already stated in Chapter 1, we focus in attaining a composite service solution based on a recommendation of services for each of the CSA Service Template. Then, through an aggregation process we check the feasibility of the final composite solution. The actual composition in not executed but rather described as a conceptual solution. Therefore, the tools and methods proposed in our work do not aim at composing services beyond a recommendation of a possible composition able to fulfill the decision maker requirements. Later on, this proposed solution could be invoked and executed, or not, by the decision maker.

2.2.2 Automatic Composition

The Service Composition topic has been subject of several surveys explaining methodologies, approaches and composition languages. Although almost all are about web service composition [86], [123], some information can still be relevant to IoS or cloud services. In [41] a detailed view on the subject is given. The authors divide composition in *Static* that takes place at design time when the system is planned and is used when services or partners rarely change; and *Dynamic*, allowing dynamic change of services in the composition, usually at runtime. The authors compare several models against different criteria as we can see in Figure 2.2.

A good example of a dynamic composition platform is eFlow [24] which models composite services by a graph of service invocation. It is similar to UML activity diagrams, and is translated to XML. The system allows the user to select services at runtime if new services are also discovered at runtime. Another example is the Star Web Services Composition Platform (StarWSCoP) [124]. However these two approaches focus on web services, which, as we already discussed in Section 2.1, do not fulfill our work requirements.

In Figure 2.2 we can see the comparison of some composition models analysed on the survey in [41].

Framework	*Composition strategy*	*Execution monitor*	*Dynamic conversation selection*	*QoS Modelling*	*WSDL Language extension*	*Transaction support*	*Graph support*
E-flow	Dynamic composition	Yes	Yes	No	No	No	Yes
MAIS	Context based	No	No	Yes	Yes	No	No
MOEM	Context based	No	No	No	No	No	Yes
SELF-SERV	Declarative composition	No	No	No	No	No	Yes
OntoMat-Service	Semantic composition	No	No	No	Yes	No	No
SHOP2	Semantic composition	No	No	No	Yes	No	No
WebTransact	Dynamic composition	No	No	No	Yes	Yes	No
StarWSCoP	Dynamic composition	Yes	No	Yes	Yes	No	No

Figure 2.2: Web services composition models [41].

In [106] the authors make a survey on the automatic service composition, where two different approaches are defined: *WorkFlow Composition* and *Artificial Intelligence Planning*. As the author explains: "The workflow methods are mostly used in the situation where the request has already defined the process model, but automatic program is required to find the atomic services to fulfill the requirement. The AI planning methods are used when the requester has no process model but has a set of constraints and preferences. Hence the process model can be generated automatically by the program". As mentioned before, in our work the workflow techniques do not apply. We are not interested in temporal constrains, like execution flows. On opposition our CSA will work as a list of requirements and constraints based on the services features and defined by the company.

2.2.3 Quality of Service and Model Based Composition

Many other approaches relate to Quality of Service (QoS), as in [113], [112] and [51]. These approaches try to capture the QoS aspects of a service and propose a composition based on some constraints. This could relate to the our work in the constraint based composition concept, where the single services are proposed for each CSA component based on requirements that must be fulfilled. However, in our case, these requirements might not be only QoS aspects.

In [95] the authors propose a new approach, *Model Driven Composition* and according to [41] this approach is classified as a *Dynamic* composition. They propose a series of business rules where the composition will be based. It defines an information model to describe the service inter-relationships as well as the components for the composition. Although an interesting approach is yet another web service composition approach, though having its limitations for IoS.

In [128] yet another approach is proposed this time based on patterns. The authors propose the use of patterns in the design of the service composition. It is also discussed their uses in non-functional requirements, however as the author say: "A

lot of work still needs to be done on bridging the gap between user requirements and patterns as well as the gap between patterns and service descriptions". This approach although interesting has some limitation. These patterns are used as development tools for achieving some predefined compositions. For example, payment patterns, which will create a composition based on the stored pattern. However, our work tries to create a composition based on an architecture, the CSA, specified by the user, which might not be extracted from a pre-determined pattern.

2.2.4 Semantic Service Composition

Semantic service composition as the name suggests is the use of semantic technology, like Linked Data or the Web Ontology Language (OWL) [82] for example, to achieve a service composition. This concept is of course tightly linked to Semantic Service Descriptions and Linked Services (see Sections 2.1.4 and 2.1.5). In [120] the authors propose a system for matching services in every step of the composition process. This is achieved by using semantic descriptions (Section 2.1.4) in this case OWL-S. Although this approach is not yet based on Linked Services, and therefore, has none of its capacities for discovering and publishing services (Section 2.1.5).

The work done in [76], although related to mashup of services, already includes Linked Data principles applied to services and is based on a platform called iServe [103].

Being a semantic description and following Linked Data principles, Linked USDL can be a perfect addition to Linked Services in order to facilitate service discovery and matching. Therefore, in this thesis we apply this service description specification as a mean to achieve the composition solution to our particular problem.

2.2.5 Software as a Service Blueprinting Composition

An interesting approach is presented in [93] where the authors developed a blueprint approach to automatically combine cloud service offerings. This blueprint is based on the work done in [92], where a blueprint template of a cloud service offering is presented. This blueprint in [92] works as a description of the service offering and can define some QoS constrains for later processing in the composition. This template can be an interesting step in defining a cloud offering and since it is described in XML it can easily be extended. However, it lacks the linked data approaches publishing capability.

In [93] the previous work is extended and a composition mechanism is built on top of this blueprint specification. As the authors outline the advantages of the blueprint approach are: "blueprints can be flexibly (re-)assembled in different configurations to form a complete SBCA (Service-Based Cloud Applications). This facilitates the composition of offerings across blueprints to express end-to-end offerings from various providers". The author defines offerings and requirements as the two core concepts in this approach. An offering is the service itself with its blueprint template; the requirements are the constraints that a certain offering needs to be executed. For example, if a database offering needs a linux server to run, this linux server will be the requirement for this particular offering. The presented mechanism

then searches this offering and creates a tree of execution, every time an offering with a requirement is found the search continues until all offerings have resolved its requirement issues. The authors call this process *Resolving a Target Offering.*

Another interesting part of this mechanism is that it uses RDF instead the original XML in the blueprint specification. According to the author "Using RDF to formally describe blueprints provides a powerful and flexible way to manipulate, aggregate and normalize data and metadata about cloud delivery models", this will also facilitate the querying process since SPARQL [104] can be used to query on the offerings.

With this being said this approach is the closest to our work, however, several differences arise: First, the service description itself is a limitation, since it only describes basic information of the service and its requirements to work. Despite these requirements could be used, in our case, as simple constraints or user preferences, the overall blueprint contains less information about the service than other semantic based approaches. The use of RDF already a good effort for facilitating the publishing and querying of specific cloud services, however the querying mechanism developed is based on service name rather than in its specification or characteristics, this greatly interferes with the whole purpose of the problem addressed in our work. If the decision maker already knows the name of the service he is going to contract, the use for a decision aid mechanism is limited since he already reached a decision.

Therefore, we will propose a different approach, merging some of the concepts of the blueprinting presented in [93] and the semantic approaches for service description and composition.

2.3 Service Aggregation

This section presents some of the work and definitions of service aggregation in the literature as well as what is the interest for the work proposed in this thesis.

2.3.1 Our Scope Regarding Service Aggregation

According to Gartner [1] in a 2009 press release [49] "An aggregation brokerage service combines multiple services into one or more new services. It will ensure that data is modelled across all component services and integrated as well as ensuring the movement and security of data between the service consumer and multiple providers.", In most of the literature as in [75], [72] and [108] to service aggregation as a component of a wider concept, Cloud Service Broker. As the authors state in [90] "...a Cloud service broker creates a governed and secure cloud management platform to simplify the delivery of complex cloud services to cloud service customers.".

As defined by [108] "The Service Aggregation role supports domain specialists and third-parties in aggregating services for new and unforeseen opportunities and needs. Services may be integrated into corporate solutions, creating greater value for its users not otherwise available in their applications, or they could be aggregated

[1] `http://www.gartner.com/technology/home.jsp`

as value-added, reusable services made available for wider use by business network users."

In this thesis however, it is not our intention to propose a full aggregation broker. It is of this thesis interest the aggregation of services as a recommendation basis and not in the service delivery and management itself. Thus, the focus is on the service discovery based on customer requirements and the recommendation of possible aggregation scenarios.

The service aggregation concept is related to the service compositionone . Both try to create added value from smaller parts (services). However, we present them in different contexts. We try to aggregate several cloud services into a feasible aggregation. This aggregation can then be converted into a composed service or as we call a Composite Service Solution. Therefore, we use aggregation to achieve a composed service. Although it is not this thesis intent to materialize the actual service composition.

2.3.2 Cloud Service Aggregation

Not many work as been done regarding cloud service aggregation in the terms proposed in this thesis. However, the authors in [130] present an interesting mechanism based on constraints to discover and aggregate cloud resources from multi-providers. They use a similar approach to the one in this thesis, using user constraints, similar to requirements, to discover the most suitable candidates to the user, and also follow the same provider agnostic approach. However, they use a two-phase discovery system: alternatives pruning with hard constraints and soft constraints optimization techniques. In this thesis we present a semantic search engine that queries a service set for services that fulfill a determined set of user requirements (exclusive requirements, similar to hard constraints as defined in [130]) and finally the alternatives are ranked according to their decision value. This decision value is controlled by the user by means of criteria and preferences. Therefore, in our approach we introduce the user preferences into the equation and not only constraints optimization like the work in [130]. It may happen that a user prefers a system that performs a little worst in some requirement but delivers as desired in another. This type of aggregation would never be possible by just optimizing constraints.

Other less related works have been developed as in [73] where an IaaS service aggregation tool is proposed. This tool however, is focused in allowing IT managers to manage their contracted cloud services in a common interface.

2.4 Decision Aid

In this section we start by introducing the scope of decision aid in our problem and then explaining in more detail the other two analysed approaches besides Multi-Criteria Decision Making: Portfolio Analysis and Matching Theory. Then, we present some of the work done in the literature to address the problem at hand with MCDM techniques.

2.4.1 Our Scope Regarding Decision Aid

The decision aid problem has been vastly discussed in the past, and many different approaches exist. However, this section is not intended to give a literature review on decision mechanisms but rather present what have been done in order to help our concrete decision problem. This translates in multi-criteria mechanisms used for service selection and ranking. In Section 1.6.3 we introduced our approach to the service decision problem, which has several specific constraints, among others, the amount of criteria to be taken into account, the complex preferences of the decision maker, some possible incomparability of criteria and how these problems could be resolved using a Multi-Criteria Decision Making mechanism. This is the scope of our work concerning decision aid, the need for analysing various constraints in order to achieve a desired decision recommendation. This thesis will not develop its own decision aid techniques but rather build on and apply already existing methods to achieve its goals.

2.4.2 Portfolio Analysis and Matching

Besides Multi-Criteria Decision Making other two approaches were considered both from the economics field, Portfolio Analysis and Matching Theory. Although the MCDM approach was already a certainty due to its capabilities and the almost perfect match to our problem, we wanted to keep every possibility in mind and so these two approaches were analysed as potential candidates.

Portfolio Analysis

Coming from the finance field, Portfolio Analysis was initially presented in 1952 [78] and it attempts to maximize portfolio expected return for a given amount of portfolio risk. It has been widely used in financial industry. In our particular case a possible use would be to have portfolios of services, each portfolio containing a category of services with a specific set of characteristics or compatible with each other. Then the concept of value dispersion from the theory would have to be mapped in some of the service characteristics allowing to choose the best service based on a specific characteristic.

The problem here is evident, the amount of relevant service characteristics to analyse would make this approach way to complex. This does not mean this approach is invalid. However, given the constraints, MCDM was a more straightforward solution.

Matching Theory

This theory, presented in [88], attempts to describe the formation of mutually beneficial relationships over time. Probably more related to our problem the predecessor of this theory, Search Theory, also applied in economics for finding partners in a transaction. The idea is to create stable pairings, in our case services and consumers, in order for both to have benefits. The Search Theory could help in situations when a change in the service provider is needed or a new provider gets

in the market, and an analysis of the best action to take is needed, if change to the new provider or stay with the old one.

Although our problem could benefit from such a solution it is not our primary target because of the difficulty to model our problem to fit the Search Theory.

2.4.3 Multi-Criteria Decision Making

With the advent of information and business intelligence today's decision makers have at their reach huge amount of information. In spite of being able to make better decisions, at least more informed ones, it is sometimes very complex to make a decision based in so many variables. This is where MCDM can be of crucial importance. MCDM consists generally on the application of mathematical models to decision making using explicitly several criteria. In recent years many approaches and algorithms have been developed, which can be used as a framework to aid the decision. Another name to MCDM can be Multi-Criteria Decision Aiding, MCDA, which better transpires the aiding component of this mechanism. An extensive review of MCDM and its methods can be found in [46].

Analytic Hierarchy Process

Another approach for decision aid, also included in MCDM is the Analytic Hierarchy Process [114] which has been developed in 1970s and widely used in different domains such as government, business, industry, healthcare, and education. This model gives an overview of the decision process, weighting the criteria and alternative solutions it does not give an automatic solution to a particular decision problem.

The AHP, as the name suggests, is an hierarchic process where the decision maker is invited to decompose his problem in an hierarchy of independent smaller problems. After the hierarchy has been set an evaluation of each element is made, by means of comparison with each other, usually by questioning the decision maker what he prefers, A over B, or B over A, A or B being decision criteria. The method then converts this information to numerical values, computable by a machine, in order to give its assessment. An interesting point of the AHP is the capability of gauge the coherence level of the decision maker preferences.

Decision Deck

The Decision Deck appeared as an European effort to collaboratively develop Open Source software tools for MCDM techniques [36]. As a result this project generated two main contributions: the *Diviz*, a software for designing, executing and sharing MCDA methods, algorithms and experiments [85] and a standardized XML recommendation to represent objects and data from the MCDA field, called XMCDA [14].

With regard to XMCDA, in our work we expect to make use of this XML-based standard for manipulating MCDM related data, such as criteria, methods and alternatives in order to maintain interoperability with the *Diviz* project. The expectation is that later research can be conducted based on our work bridging it to other MCDM methods.

2.4.4 MCDM Approaches

For the purpose of this thesis, we will analyse two service related decision problems which MCDM can resolve: *Service Selection* and *Service Ranking*. *Service Selection* allows us to decide from the entire service set the ones which can fulfill the decision maker requirements. This problem is intimately linked to the service description problem. Depending on the level of service description a different degree of criteria has to be considered. *Service Ranking* is the process of comparing the selected services in order to gauge which one is the best solution taking into account the decision maker preferences.

In the past few years several MCDM methods have been used for service ranking, for example, the authors in [125] propose a multi-criteria mechanism based on non-functional requirements described using the WSMO specification. This multi-criteria approach is a simple weighting mechanism, the simplest of the MCDM methods, where an importance value is assigned to each criteria and then computing the highest results (Simple Additive Weighting) [46]. Despite simple and easily manipulated, since is it quite easy to tamper results by changing weighting parameters, this approach is a good example of how MCDM methods can be used to address the service ranking problem. However this approach allows us no tolerance to imperfect knowledge, or user preferences besides manually altering the method parameters.

In [126] we find a survey on a different kind of MCDM methods used for service ranking with regard to Quality of Service, Fuzzy Methods. As the author says "Different Quality of Service properties may have different important levels to specific service consumers. In many cases, the values and the important weights of Quality of Service criteria are not easy to be precisely defined", for this reason research has been conducted to apply fuzzy theory to this problematic, allowing the definition of vague concepts. This kind of approach despite being complex allows us to deal with this imprecise knowledge in some domains, which is often the case for decision makers preferences.

The work done in [137] is quite similar to our work and so a good starting point, in a way that the author tries to compose web services ranking them based on Quality of Service aspects. However once again the ranking method used is a simple additive weighting (SAW) [46], which, as already explained, has several limitations. This thesis distinguishes himself from the work done in [137] in three different ways: first the generalization to Internet of Services instead of web services; second, the criteria for ranking services focus in more than just quality of service aspects; finally the methods used for ranking these services.

In [133] the author proposes the AHP as a method to evaluate the service composition. The difference to the work we propose is that in [133], the AHP is used to evaluate the composition as a whole and not to evaluate to what extent a determined service can fulfill a requirement or a specific user preference. It is nevertheless one more good example on how different MCDM methods can be used to address the composition problem encompassing service selection and ranking.

The work done in [136] is another application of the SAW method to solve the service selection problem. As the author says "The goal of service selection algorithms is to select individual services that meet QoS constraints and also provide the best value for the user-defined utility function". In fact this gives us a perspective on what has to be done on service selection level, first the matching of services that fulfill the user defined constraints and then the calculus of the best solution, where the service ranking as particular critical role. Other example is proposed in [25], where a fuzzy approach is used for selecting a service provider.

3
Use Case

With the arrival of SOA [43], and all the service oriented products, such as IaaS, Paas, SaaS, etc (see Section 1.1.2, more and more enterprises tend to outsource some of their competences or business processes by contracting other enterprises specialized in those services.

Alongside with this increasing demand for service oriented products, there has been obviously an increase of service providers, where the cloud based services play a major role. For a view on Cloud refer to Section 1.1.2. Section 1.3 shown the problem that current companies face when trying to adopt cloud services and that the Cloud is growing and a vast field of new opportunities is waiting for those willing to take the next step.

In this section we present an extended version of the summary given in Section 1.4, with a use case of a problem that a company could face. We start by describing the problem in more detail in Section 3.1. In Section 3.2 we present how the use case would be used and by whom. Finally in Section 3.3 we map this concrete problem to the diagram presented in Figure 1.3 with the required steps for contracting a composite service solution. A full version of this Use Case, with requirements and architecture specification can be found in Appendix B.

3.1 Problem Description

We start by introducing our IT company, with 200 employees, most of them working in the only building the company owns. Taking into account the number of electronic devices constantly running in the building, the company knows the energy consumption is high, and is concerned about reducing it to a minimum, reducing their operational costs. In addition to the above concerns, the European Union directives point to a 20% cut on the energy consumption as well as encourage the use of energy meters to monitor consumption and efficiency [1].

Moreover, the constant increase in social awareness for green energy and environmental concerns, press the company to adopt these measures in order to maintain a "green image" and transparency standards to their consumers.

[1] http://ec.europa.eu/energy/efficiency/index_en.htm

Since the building is full of electronic components there will be a huge amount of sensors to retrieve the data for energy consumption and management. There are many different approaches stated in [63], [50], [1], [119], [62]. With this many devices the amount of data to collect, process and predict is huge, which leads to a fair capability of storage and data processing. The company currently has none of this capabilities to dedicate to their Energy Monitoring and Efficiency System (EMES), and so it is willing to outsource these capabilities. Using the same arguments from Section 1.3, choosing the cloud seems a good option.

The cloud will allow the company to surpass the storage and processing of all the data, without building their own infrastructure. However, they also want to make this data public through a website, that can be accessed by anyone. Also, due to the European Union above mentioned concerns, a monthly report should be delivered, for this matter the EMES needs to automatically create and send reports.

To facilitate the usability and ease of use, an app that allows employees and managers to monitor and control the system should be developed. For eliminating the "in loco" controlling approach the company wishes to adopt an SMS system to control the EMES, as well as to alert the right person if something of relevance happens. The goal is to make the system as controllable as possible without the need of a permanent manager in the building.

3.2 System Usage

In this section we present the actors and their interactions with the EMES, The purpose is to better understand the system concept and the reason for some of the choices for potential cloud services solutions.

Interacting with the EMES we can define 4 major actors, each of them with their own functionality, and different access to information generated by the EMES. The first identified actor is the sensor that will collect data from the correspondent device and transmit it to a wireless receiver. Note that hundreds of sensors will be collecting data at the same time and transmitting it to the main receiver.

The second actor is the building manager, or the person in charge of maintenance or controlling the energy system. The three main interactions with the system are:

- View data (readings, predictions, reports, efficiency,...)
- Create patterns (prediction patterns, SMS alerts, report patterns,...)
- Control the system (SMS commands, App Control, resolve issues,...)

The third actor is the common employee, that only has access to some of the features available to the building manager. They can view most of the information available (average comsumption, ratios, etc), and can even be allowed to make some extra simple interactions through the mobile app or website, such has turn on or off their equipment, or communicate a malfunction to the building manager.

At last the general population. This actor has access to public information, allowed by the company, such as overall readings, efficiency, public reports, overall statistics, etc. This information will be available through the website.

Note that multiple levels of interaction can be specified for each of this actors. For example, in the website, while the general public can only view public information, an employee can see other kinds of information, alerts for example, or energy consumption by department.

3.3 Contracting a Composite Service Solution

For convenience reasons we bring back the image from Section 1.4. Our company wishes to contract several cloud services that can be integrated to fulfill their EMES requirements. To achieve such a system several steps have to be preform as shown in Figure 3.1.

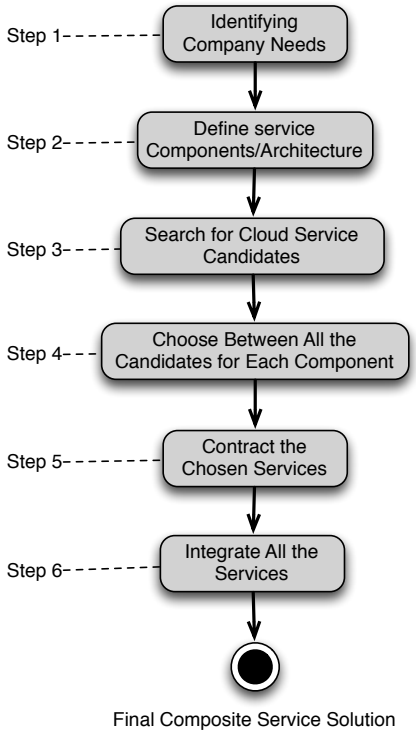


Figure 3.1: Steps for contracting a composite service solution.

The first step corresponds to the identification of the company needs. Those needs are the requirements for the system captured by analysing the company needs as we did in Section 3.1 and 3.2. Other specification could have been collected such as non-functional requirements, dependencies between components or metrics. However, they are not important for the scope of this use case.

The second step, is to compile all the information collected in the previous step and create an architecture view. It could be for example an UML component diagram or any other tool the company wishes. Something that formalizes the requirements information. In this step everything that could influence a decision towards a specific service must be included. The purpose of this process is to gather and

maintain the important information for later steps, this way the company never looses track of what they need, however the architecture can change with time, if no good solutions are found. This process is no different than the one used in software development.

In the third step the search for potential cloud services to fulfill the documented requirements starts. The company knows the importance of the EMES they want to implement and so it decides to allocate a team solely dedicated to this task. They search for all potential candidates without excluding any, except for those who do not fulfill the basic functional requirements. All those services who comply with these basic requirements are cataloged and all the relevant service information is collected and stored for later use in the decision process. This information is mainly extracted manually from the service provider website, making this entire task long and painful.

After collecting the information from enough sources, the fourth step begins. It is now time to reach a decision on which cloud service to use for each EMES component specified in step 2. Several meetings are scheduled with the team responsible for finding the services and the team responsible for implementing the EMES, even members of the administration are present for approval. All the cloud services are discussed and pros and cons are debated. Due to the complexity of the system to be implemented one meeting is not enough, several factors are critical, besides the price, availability and security are of the up most importance. The system must also comply with the amount of data collected from the sensors, which is not an easy task.

After reaching a decision on which services to use, the board starts the contacts to contract the chosen services, this is the step 5.

Once the cloud services are contracted and running the final step belongs to the development team who has to integrate all the contracted services, and implement some parts of the system such has the website, or the mobile app. This step can be outsourced, however since the company is an IT specialized company they decide to do it themselves.

After a few months the EMES is finally up and running and the first report with the energy consumption and efficiency is sent to both the administration and the EU. The next months the company starts to reduce energy usage, and begins a marketing campaign in their new website for customers to prove their ”green image”.

4

Analysis and Specification

This Chapter presents all the CloudAid Prototype analysis and specification phase. Three elements are presented: Section 4.1 presents the requirement analysis and its elicitation process; Section 4.2 presents the prototype architecture and all its relevant components; finally, Section 4.3 presents the test plan for the prototype testing phase.

4.1 Requirements Analysis

This section outlines the analysis process followed regarding the CloudAid Prototype requirements. It also points out the main objectives that drive all the other requirements listed in Appendix C.

4.1.1 Requirements Elicitation

The purpose of requirement elicitation is to discuss with customers and end-users what are their needs and which the required application functionalities. However, our application is a research prototype. The main goal is to prove and validate the research proposal of providing a decision based aggregation of cloud services mechanism. Nevertheless, there is the compromise to simulate as much as possible what would an end-user be interested in.

This way, the main stakeholder interested is the Decision Maker, who is charged with the task of finding cloud solutions for a specific problem. Since, as already explained, we had no contact with a real customer, the stakeholder requirements were discussed within the research group in order to simulate what would be the key points for the application.

Every time a new path of research was followed or another was abandoned the requirements also changed. This made some of the requirements to change their priority or to disappear from the list since they were no longer suitable for our purpose.

4.1.2 Requirements Prioritization and Categorization

All the requirements listed in Appendix C are prioritized using the MoSCoW method [28] which has four levels of priority:

- **MUST** have the requirement
- **SHOULD** have the requirement if at all possible
- **COULD** have the requirement if there are enough resources or if it does not affect higher priority requirements
- **WON'T** have the requirement in this version or WOULD like to have in future developments

Also the requirements were divided according to the FURPS+ Methodology Document [42] which is a methodology to group requirements in categories. FURPS means Functionality, Usability, Reliability, Performance and Suportability. However, the FURPS+ add a few more categories to describe system constraints: Design, Implementation, Interface and Physical.

We make use of all but the physical category. The reason for not using it is because our prototype simply does not have any physical constraints. There are no especial needs for physical resources, a simple personal computer suffices for the prototype needs. Further explanation of the FURPS+ categories is given in each category section in Appendix C.

4.1.3 Top Level Objectives

The top level objectives were defined as directives for the requirements definition and had the thesis objectives in mind at all times.

The most important objective is the "proof of concept". This is the true purpose of this prototype application hence, all the requirements must comply to this purpose and help to validate or prove the concepts discussed in this thesis. With this in mind several other objectives arise:

- **Usability Concerns** - the application does not need to follow usability standards or guidelines. Nevertheless, we wanted that both the requested and the presented information to be understandable by the user.
- **Performance Concerns** - questions such has scalability, throughput, response time among others were not a concern. It was not the purpose of this prototype to show high performance capabilities. However, it should be noted that some performance requirements are used to serve as metrics for system testing purposes. The rational for this decision was that a real decision maker would not mind to wait for the computation of the aggregated solution. However, it would matter if the user has to wait before interacting with the system, this is, wait to insert data. Therefore the final decision computation is not a performance concern, but the search mechanism should perform in reasonable time since extra information is asked to the user afterwords.

- **Development Freedom** - Although there are a few standards we decided to follow (MVC model, or the XMCDA standard), there are no other development constraints such as programming language or frameworks.

- **Modular architecture to facilitate extensibility** - the current prototype should permit extensibility and modularity, hence to allow further developments. As so, the prototype capability to be extended and modified is a requirement. It is our purpose to compose the building blocks for an environment that can be used by other researchers to test their own methods, not only decision methods but also search capabilities or other new functionalities. This lead to some architecture decisions such as the MVC model [70].

The full list of requirements identified in the elicitation process can be consulted in Appendix C.

4.2 CloudAid Architecture

This section explains in some detail the prototype architecture with all of its components, the execution flow and discusses some decisions made in the design process. In Section 4.2.1 is discussed the notation used for the diagrams produced during the design phase. In Section 4.2.2 the actual architecture is presented and a description of its modules and components is given. An use case to show the stakeholders interaction with the system is presented in Section 4.2.3. In Section 4.2.4 are presented the two data models used for the prototypes required data. Finally, the languages, frameworks, tools and standards used are listed and their rational explained in Section 4.2.5.

4.2.1 Diagram Notation

The first problem that emerged when trying to describe the system architecture was to use a notation that could easily be understandable by all interested parties.

Although UML (Unified Modeling Language) [17] is the most well spread and widely used tool in software projects the scope of its use is too software-based for our primary objective. We wanted to easily describe and explain how the prototype works in a way that even non-software oriented people could understand the meaning.

To achieve the above objective we chose the Fundamental Modeling Concepts (FMC) [132] [65]. The goal is to provide the concepts to create and visualize models enabling the different stakeholders to share a common understanding of a system's structure and its purpose. "FMC is based on strong theoretical foundations, it has successfully been applied to real-life systems in practice (at SAP, Siemens, Alcatel etc.) and is also being taught in software engineering courses at the Hasso Plattner Institute for Software Systems Engineering.". [132]

It should be stressed that FMC and UML are not different tools for the same problem, they are rather complementary tools for different phases in the software development lifecycle as shown in Figure 4.1.

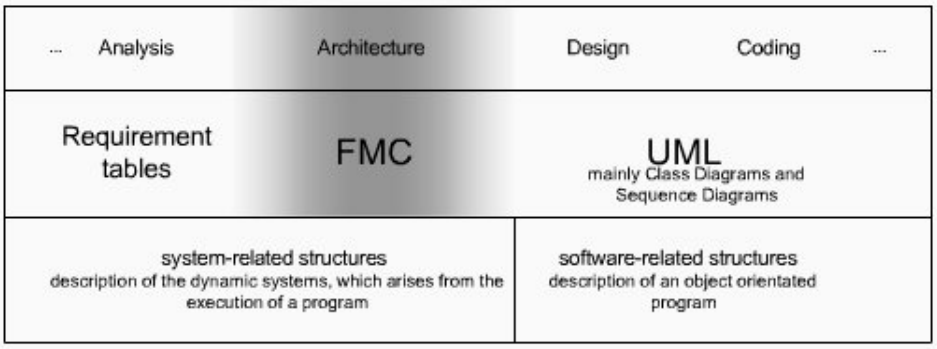


Figure 4.1: Software Lifecycle [3]

While UML is software-related and allows a complete description of the software system with diagrams such as Class, Sequence, Use Case, Component Diagrams, it fails at providing a standard description for system architectures. FMC on the other side is system-related and fills in the void left by UML. "FMC and UML are not mutually exclusive since they are applicable in different phases in the software development lifecycle" [3].

For the prototype application architecture we followed this approach and used FMC for the general system architecture and UML for the software related specifications (Data Models, Sequence and Use Case diagrams).

Note that a more informal representation was also used for several diagrams. These diagrams, however, are only used to facilitate the system comprehension and are variations of the FMC notation with extra information. They should be viewed as an intermediate step between the system architecture (FMC) and the software specification (UML).

4.2.2 Overall Architecture

As stated in Appendix D the system application was entirely build with the MVC (Model-View-Controller) Model in mind. From Figure 4.2 we can see the overall system architecture described with the FMC notation. Two ideas are easily extracted from the diagram: first, that the MVC Model was indeed implemented as specified by IMPR1 requirement. The second is the modular approach, as requested by IMPR2 requirement and needed for the correct implementation of SR3 Requirement. All these requirements can be consulted in Appendix C.

The system is composed of five modules decoupled from each other. These five modules are coordinated by a sixth module (Controller) which only purpose is to keep the system flow of execution and drive the data to the correct destination. The six modules are:

- **UI** - the View in the MVC Model, is responsible for the user interaction, it holds all the user interface for both the information retrieval and presentation.
- **Controller** - the Controller in the MVC Model, mediates the information transactions between the View and the Models. It is responsible for the system boot and consequent execution.

- **CSAEvaluator** - the Model in the MVC Model. Is responsible for evaluating and preparing the CSA data. This operations consist in data validation or verification for example (i.e: there is no point in aggregate solutions composed of only one service.)
- **SearchEngine** - the Model in the MVC Model, is responsible for service search mechanism.
- **DecisionEngine** - the Model in MVC Model, is responsible for the decision process for a determined ServiceTemplate.
- **AggregationEngine** - the Model in MVC Model, is responsible for the aggregation process to find the final aggregated service solution.

Note that the Controller, SearchEngine, DecisionEngine and AggregationEngine will be further discussed in Sections 4.2.2.2, 4.2.2.3 and 4.2.2.4.

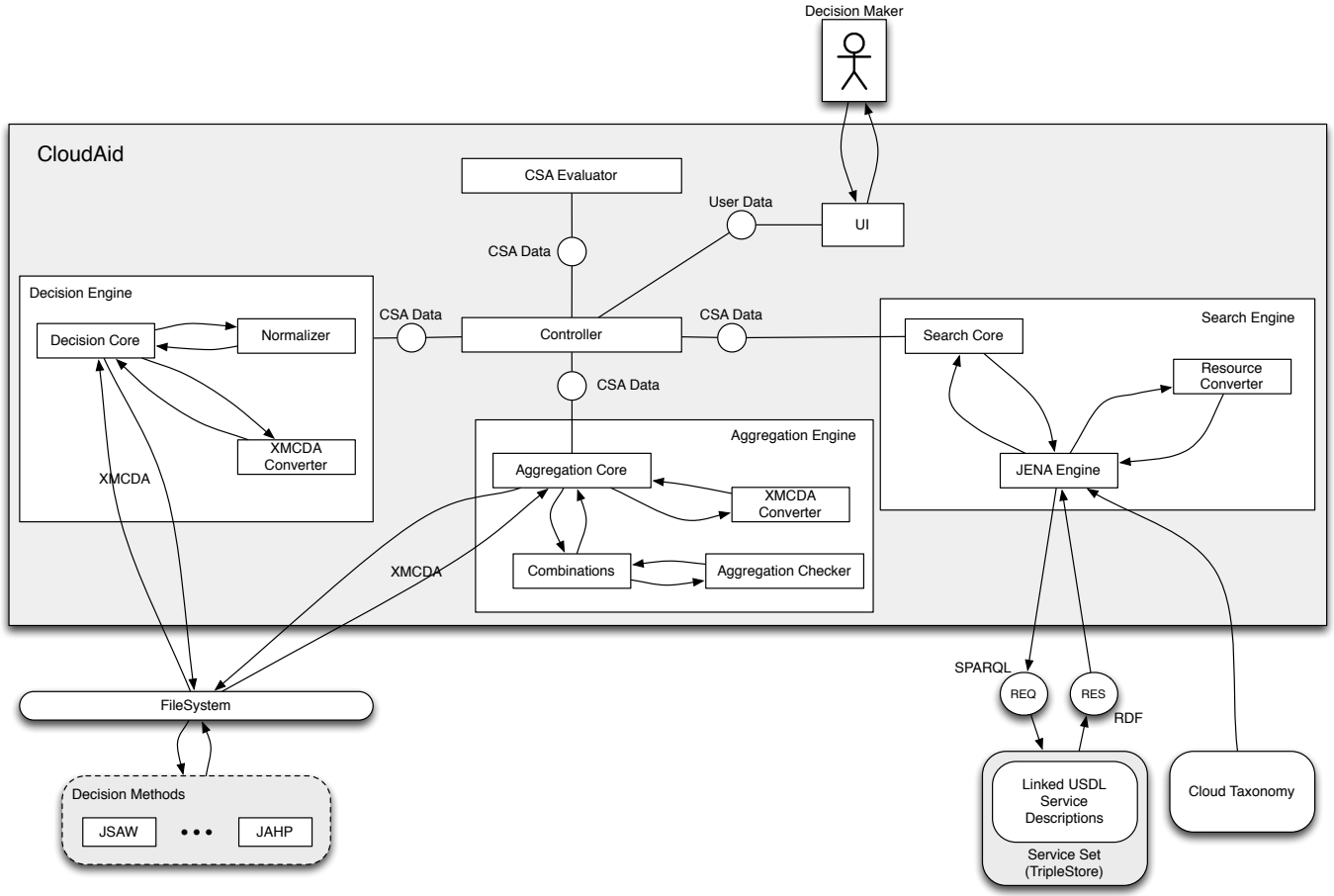


Figure 4.2: CloudAid High Level Architecture

From Figure 4.2 is also possible to see the use of three external components:

- **Decision Methods** are used by the system to decide upon the alternatives found for a particular ServiceTemplate. These methods can be simple or more complex applications, each of them with their own logic. The important thing to retains is the external component, this means that these methods can be developed by third parties and used by the CloudAid Application. The linking point is the Filesystem, that works as communication bridge between the two entities: CloudAid Application and the Decision Method. It should be stressed out that all the methods should comply with the XMCDA standard for describing and publishing decision problems and results.

- **Service Set** is the service description repository. This repository holds all the Linked USDL service descriptions and is where the service search is going to be performed. Ultimately, this would be a triple store, as requirement SR2 states, that would be populated from services crawled from the IoS, as requirements FR4 indicates.

- **Cloud Taxonomy** has it has all the relevant cloud characteristics supported by the CloudAid application. As requirement DR3 states, the use of an external ontology to describe cloud concepts would benefit the search mechanism. These concepts are therefore used in the SearchEngine module.

All the external communication to and from the CloudAid application can be seen in Figure 4.3. This informal diagram tries to add some extra information to the FMC notation by clearly stating which type of information is being communicated and the frameworks and tools used.

An important remark is that besides the previous described external components, a new one is introduced in Figure 4.3, the CloudGen. This component is a cloud service description generator created with the sole purpose of testing the prototype application. The rational behind the decision to include this new component is simple, there are still not enough Linked USDL service description in order to properly test the system. The fact that Linked USDL is still under development explains the lack of service descriptions.

Since this new component is not a key element for the CloudAid Application functioning but rather for its testing, we only briefly explain how it works and the advantages of such an approach.

CloudGen

In order to test the prototype a huge amount of service descriptions is needed. Although some manual descriptions were done, specially during the Linked USDL initiative collaboration for the Pricing Model, they are far from enough for the need of a suitable Service Set.

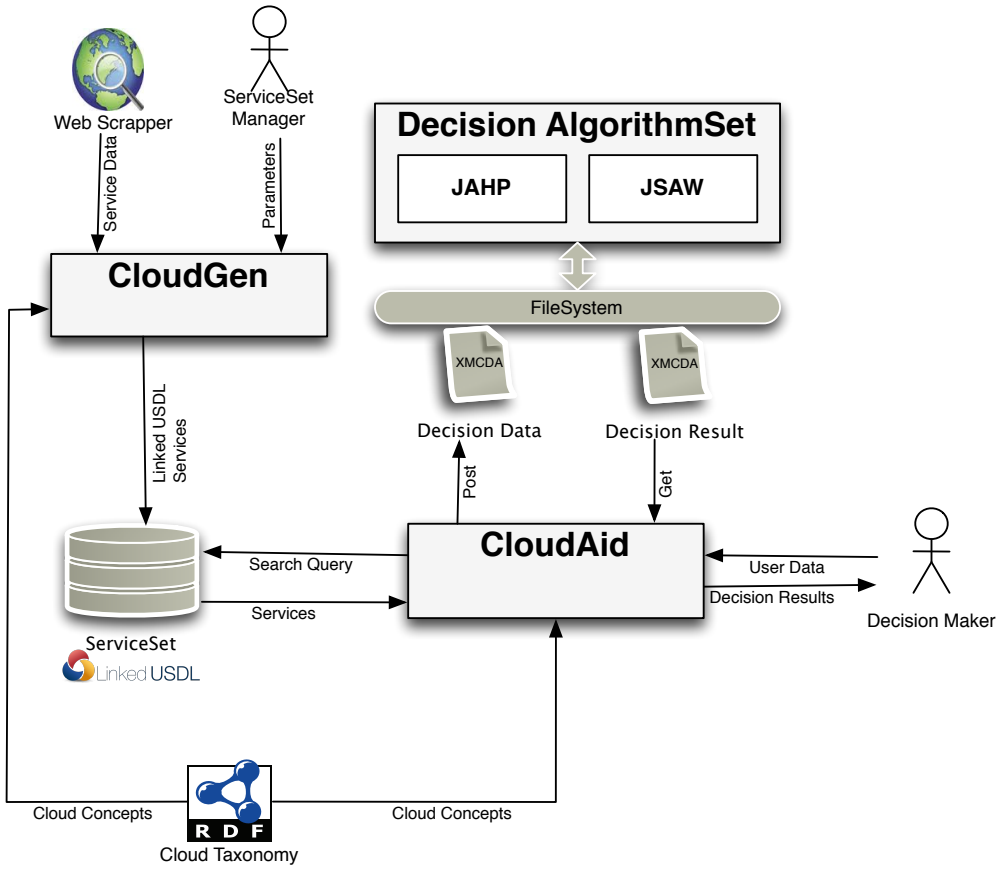


Figure 4.3: CloudAid Project Architecture

Hence, the need for a generator capable of creating fair amounts of Linked USDL service descriptions was evident. However, the generated services should be as similar as possible to real world services and not merely random generated services that makes no sense for a potential buyer. With this constraint in mind the first concern was to allow the generator to receive a series of characteristics extracted from real service providers websites. These characteristics would be relevant for the generation process by giving some "real world" values for common service characteristics.

The second mechanism is the inclusion of a Service Set manager in the generation process. This manager is responsible for providing service parameters to the generator. Then all this information is joined together and all possible service descriptions are generated based on this input data.

Going back to Figure 4.3 we see the two sources of information: the Web Scrappers, that extract information about real service characteristics and the Service Set Manager that provides the desired service parameters. The result of the generation is then populated into the Service Set Triple Store.

Note also the link between the CloudGen Application and the Cloud Taxonomy. This has to do with the usage of the cloud concepts described in the CloudTaxonomy ontology. This way we ensure that the newly generated service descriptions are

already annotated with these concepts, thereby facilitating the search mechanism.

4.2.2.1 Controller

The controller is the most important component of the CloudAid Application. It is responsible to boot the required resources and to start the application execution.

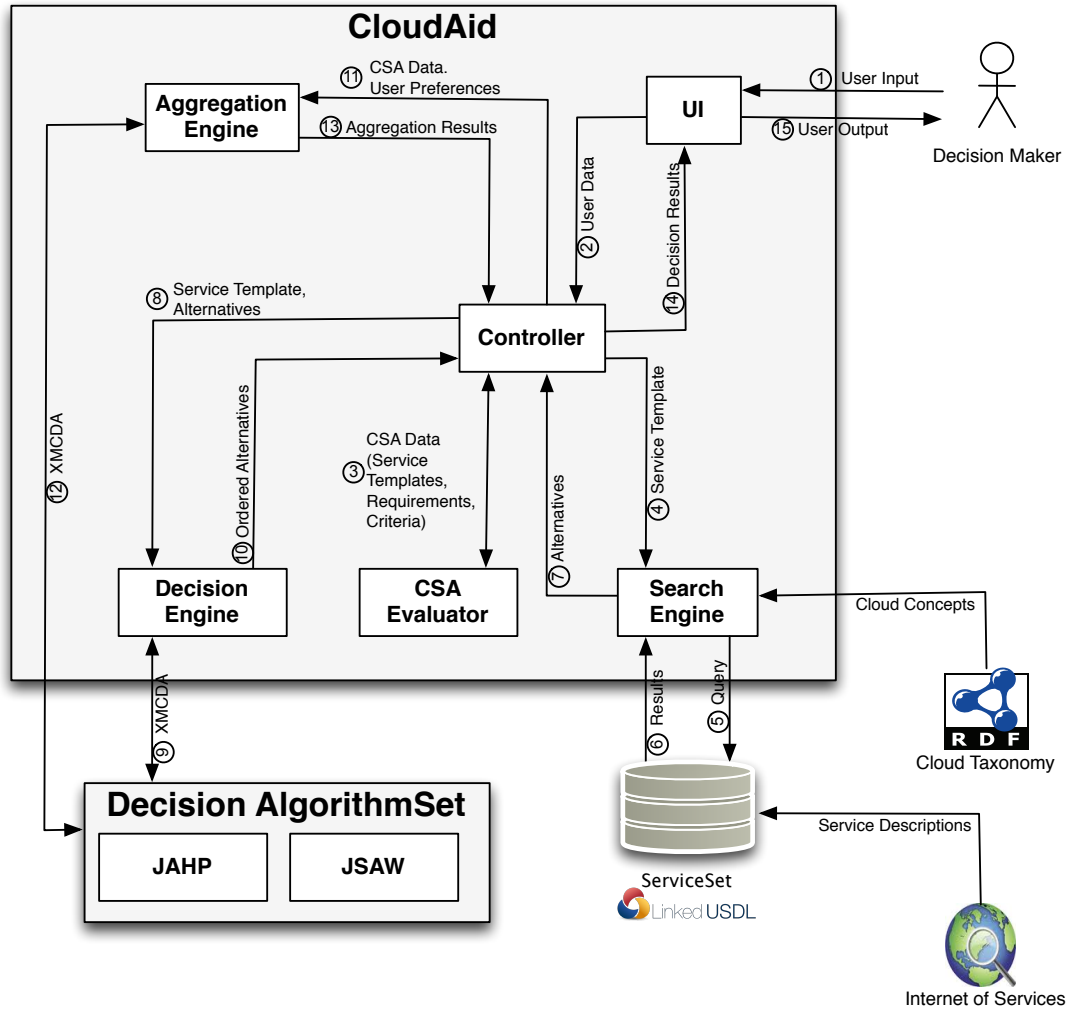


Figure 4.4: CloudAid Architecture

The concern about modularity and extensibility lead to this module that can easily be modified if any major change to the normal functioning of the application is to be performed. Figure 4.4 helps us to follow what exactly is the goal of this component. It works as a dispatcher of data to the module responsible for the next step in execution. If we take a deeper look into the flow we see the similarities with the Figure 4.5 with the steps for defining an aggregated service solution.

First, in Step 2 we have the definition of the CSA components, which is also the first step in the application, where the decision maker inserts the required data

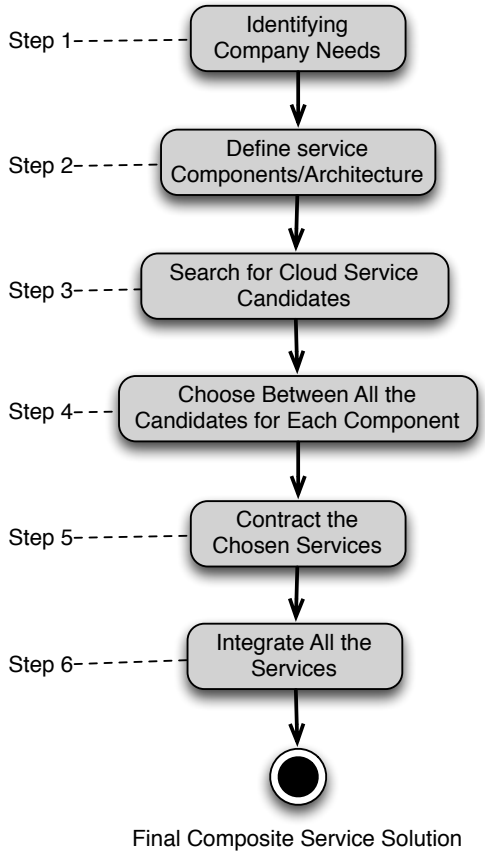


Figure 4.5: Steps for defining a composite service solution

into the UI module (Step 1 and 2 in Figure 4.4). Then this input is evaluated and analysed for incorrect data in step 3 of Figure 4.4.

After the input data, starts step 3 (Figure 4.5), the search for suitable alternatives. This step corresponds to steps 4 to 7 in Figure 4.4, where the search engine is called and executed.

Finally, we have step 4 (Figure 4.5), where a decision is made on which alternatives to contract. This step corresponds to the execution of two modules, first the DecisionEngine and then the AggregationEngine which correspond to steps 8 to 13 in Figure 4.4.

In all this process the Controller work is to call the responsible module, send the required data, and then receive the results.

Since the FMC overall architecture diagram we have been going down in the specification level, first in Figure 4.2 an high level system-related description was presented using FMC notation, then in Figure 4.4 an informal notation was used to serve as intermediate step, with extra information about data types and flow of execution. In the CloudAid Public Repository all the Architecture Documents [6] can be found, including code-related sequence diagram using UML notation for all the modules herein presented. This exercise also demonstrates the process used during the design phase of the CloudAid prototype application (from high abstraction to

software-related).

4.2.2.2 Search Engine

This module has the purpose to wrap all the necessary search related mechanisms in the application. Once again from SR3 and IMPR2 requirements (Appendix C), we extract the need for modularity and extensibility, this way we ensure that any extensions or modifications to the search mechanisms will only affect this module. We also facilitate its comprehension since no knowledge of the remaining modules is needed.

Although it is the second module to be invoked by the Controller, the Search Engine is also responsible for an important task during the application boot process: Initialize the Service Set. This way the application deals with this long task only once.

After the initialization of the Service Set, as we saw in Figure 4.4, the Search Engine is invoked after the CSA data has been collected, by the UI module, and analysed by the CSAEvaluator module. The Search process is invoked for each ServiceTemplate in the CSA. In other words, we want to search for alternatives suitable for each ServiceTemplate based on their requirements.

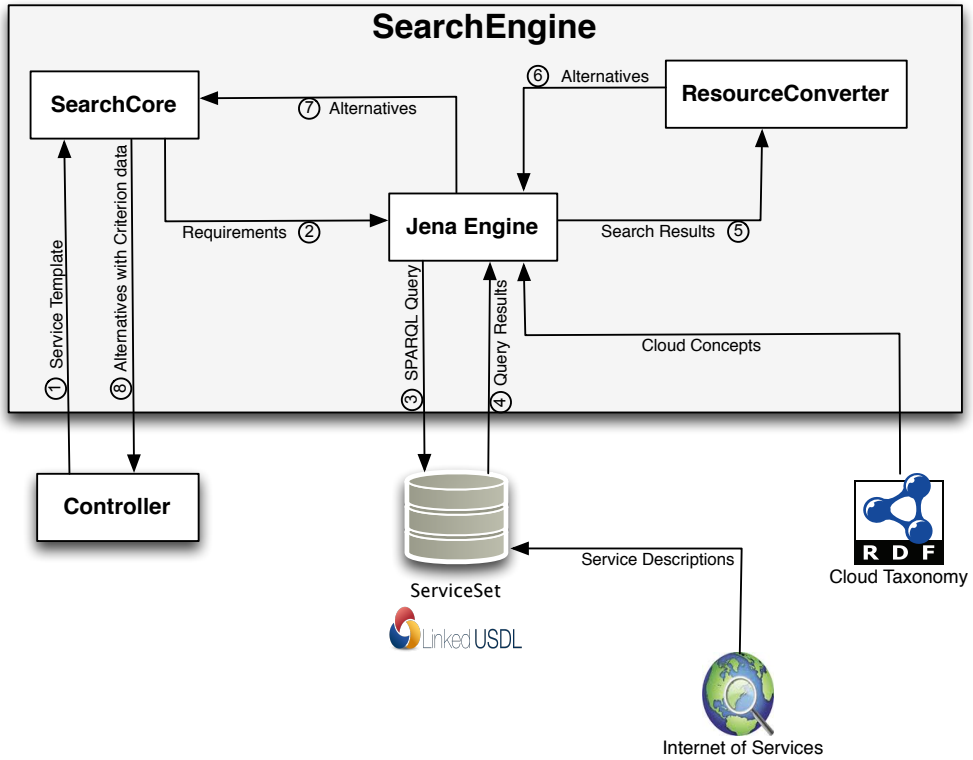


Figure 4.6: CloudAid Architecture: Search Engine Module

From Figure 4.6 we can see the entire search process and what kind of information passes from one component to the other. The main steps are:

1. **Extract exclusive requirements**: (Step 2 in Figure 4.6) The first step, after receiving the ServiceTemplate data is to extract the exclusive requirements (see Section 4.2.4.2). These exclusive requirements are then sent to the Jena Engine. The Jena Engine is the component responsible for creating and executing queries based on the needed requirements. Please refer to Section 4.2.5 for an explanation of the frameworks and tools used in the Jena Engine component. This is the key component in the search mechanism because it is where the actual search happens.

2. **Query the Service Set**: (Steps 3 and 4 in Figure 4.6) The Jena Engine job, after receiving the exclusive requirements, is to build upon these the specific query. This query is then submitted to the triple store and the search results are retrieved. These results are the suitable alternatives for the particular Service Template being analysed.

3. **Query the Service Set**: (Steps 5 to 7 in Figure 4.6) After building the query and get the search results, the Jena Engine must pass these results to the ResourceConverter component. As previously stated our Service Set is composed of Linked USDL Service Descriptions which are RDF triples. These triples must be analysed and the relevant information extracted and converted to a suitable data model that can be easily accessed and processed. Therefore, the purpose of the ResourceConverter component is to receive a set of RDF search results and convert them into the Service Data Model (see Section 4.2.4.2). Only then we can call the search results as alternatives.

4. **Extract the Criteria values for each alternative**: (Step 8 in Figure 4.6) The final step is to take the newly converted alternatives and extract the relevant feature values. These relevant features are those which are linked to cloud characteristics that were defined by the decision maker as Criteria.

It should be stressed that the cloud concepts defined in the CloudTaxonomy are used by the Jena Engine, as shown in Figure 4.6.

4.2.2.3 Decision Engine

The decision core responsibility is to receive a group of alternatives for a specific Service Template and process those alternatives data in order to rank them accordingly to their decision value.

Although it seems a simple process it is quite complex and involves several different components to achieve its final goal: return a ranked list of alternatives based on the decision maker preferences. Again requirements SR3 and IMPR2 were the focus in order to achieve the desired modularity and extensibility in this module.

The decision process depicted in Figure 4.7 has the following steps:

1. **Receive a ServiceTemplate and their alternatives and criteria**: (Step 1 in Figure 4.7) From the Controller module the Decision Engine receives the ServiceTemplate data and the set of alternatives found by the Search Engine. Note that as the Search Engine, also the Decision Engine is invoked once for

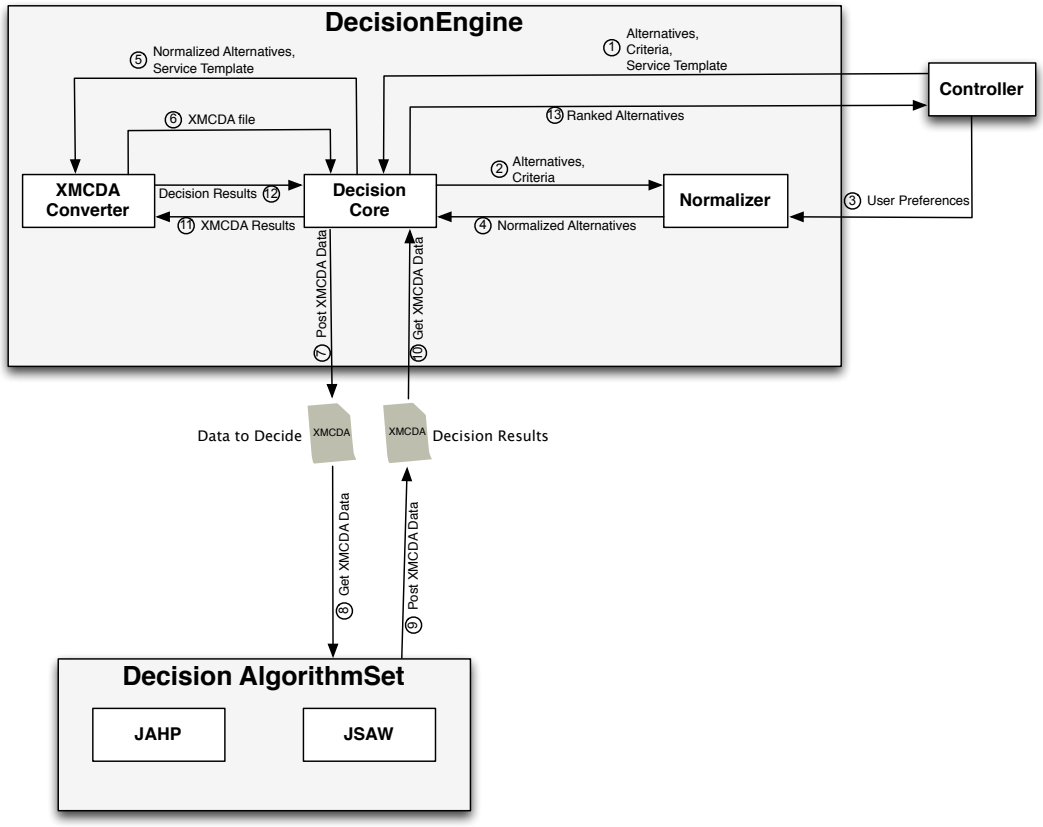


Figure 4.7: CloudAid Architecture: Decision Engine Module

each ServiceTemplate. We are deciding which of the found alternatives are the best having in mind the user defined preferences.

2. **Normalize its alternatives data**: (Steps 2 to 4 in Figure 4.7) The first thing to do with the alternatives data is to normalize their values. We want to compare multiple different criteria, this means that the data must be uniformed. This normalization process is performed by the *Normalizer* component and may require extra user information, as for example the preferable value for a specific criterion (e.g.: the user says: "The best Storage capacity value for me is 4Gb", this means that the closer to 4Gb the Storage Capacity characteristic value of the alternative is, the better). Step 3 in Figure 4.7 shows this information required from the decision maker.

3. **Convert the decision problem into XMCDA**: (Steps 5 and 6 in Figure 4.7) After the normalization of the alternatives data, the decision problem is ready. However, in order to cope with requirement SR2 it was decided to use a standard language for describing decision problems to be used with MCDM methods, XMCDA (see Section 4.2.5). A conversion has to be performed, from the alternatives normalized data to the XMCDA format. This is were the XMCDA Converter component is important, it receives data and returns a XMCDA file with the decision problem.

4. **Publish the decision problem**: (Steps 7 and 8 in Figure 4.7) Once the XMCDA file is created it is published in the FileSystem for the MCDM method to analyse and decide. The method to use, as explained in Section 4.2.3, is chosen by the system based on a series of question asked to the decision maker. By using the XMCDA standard we ensure that, if needed, the decision method can be changed in different executions. It is also ensured the extension to other decision method in future developments. After publishing the decision problem the decision process is halted until the external Decision Method reaches a result and publishes it.

5. **Extract the decision results from XMCDA**: (Steps 10 to 12 in Figure 4.7) When the external Decision Method publishes the result it also come in the XMCDA format, therefore, a new conversion is needed. However, this time we need to convert from XMCDA to a list of alternatives with their decision results. The results are stored as alternative performances and must be stored in the CSA Data Model as Results (see Section 4.10).

6. **Rank the alternatives according to their decision value**: (Step 13 in Figure 4.7) The final step is to sort the list of alternatives according to their decision value (performance) and return them back to the Controller.

4.2.2.4 Aggregation Engine

The Aggregation Engine is the final step in the execution flow of the CloudAid application and unlike the Search and Decision Engines is only invoked once. The reason is simple, this module is responsible to group together all the information produced so far and compute it to find a final aggregated solution composed of one alternative for each ServiceTemplate. Hence, it works with all the information from the CSA and the Service Data Models.

From Figure 4.8 we can see the following steps:

1. **Get ServiceTemplates weights**: (Step 2 to 7 from Figure 4.8) The first task, after receiving the data from the Controller, is to get the weight values for each ServiceTemplate. These weights will be key values to compare aggregated solutions. However, there are as many different ways to collect them as Decision Methods. They can either be defined directly by the user when inserting the CSA Data in the UI module or inside a Decision Method. For the latter case we need to, once again, convert to XMCDA, publish the results for the Decision Method to use, retrieve the results from the method and finally convert them again to the CSA Data Model. This process is the same used in the Decision Engine for evaluating the alternatives. In this case, however, we are evaluating Service Templates according to their importance for the final aggregated solution. For instance: A user can decide that a better database (a database alternative with highest performance) is more important for his goals than the other Service Templates in the CSA. In this case he gives an higer weight to the database Service Template.

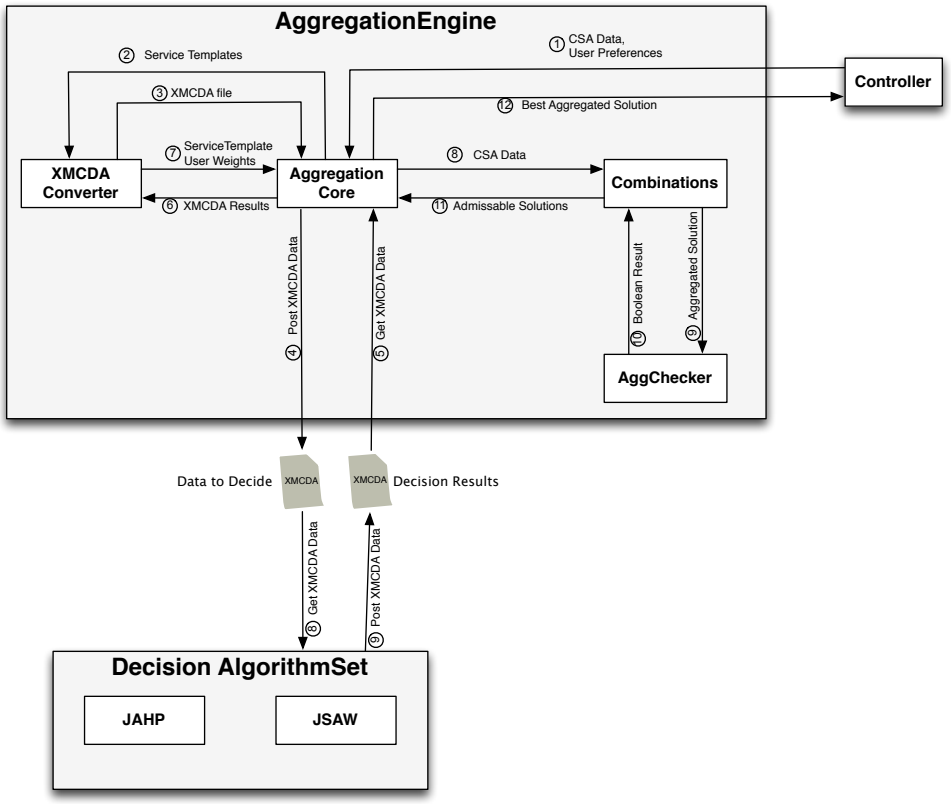


Figure 4.8: CloudAid Architecture: Aggregation Engine Module

2. **Compute the admissible aggregated solutions**: (Step 8 to 11 from Figure 4.8) This is the Key step in this module. The Combinations component is where the CSA Data is computed and aggregated combinations are generated. Each aggregated combination is composed of one alternative for each ServiceTemplate. Everytime a new aggregated combination is generated the system must check if it is admissible or not. This check mechanism is performed by the AggChecker component (Steps 9 and 10 in Figure 4.8). An aggregated solution is considered admissible if it meets all the CSA requirements and if there are no restrictions of a particular ServiceTemplate alternative which makes impossible to combine it with other ServiceTemplate alternatives.

3. **Find the best admissible solution**: (Step 12 from Figure 4.8) After all the admissible aggregated solutions have been found they must be compared to find the best solution based on the decision maker preferences. This is where the ServiceTemplates weights extracted in step 1 come in. The final aggregated solution is then returned to the Controller.

4.2.3 Application Use Case

The Use Case Diagram in Figure 4.9 tries to describe the possible interactions between the decision maker (stakeholder) and the system. The idea is to show which are the system functions used by the decision maker and what are the interaction

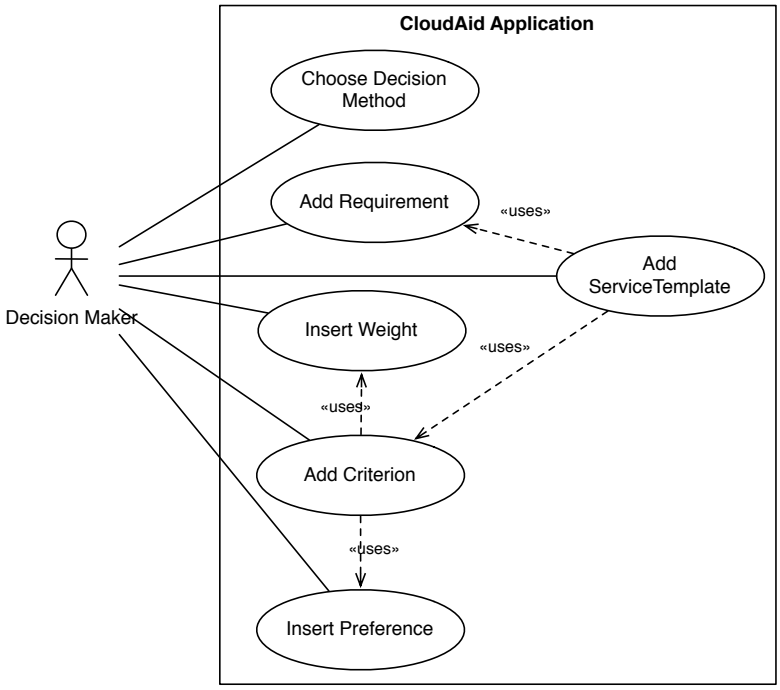


Figure 4.9: CloudAid Architecture: Use Case Diagram

points. As we can see from Figure 4.9 there are five possible interactions for the only existent stakeholder, the decision maker:

1. **Choose Decision Method** - The user interacts with the system in order to choose the decision method he feels comfortable using. The system may ask a series of questions to know what is the best option.

2. **Add Requirement** - The user interacts with the system in order to insert data about a new requirement. The data includes: name, cloud concept, minimum, maximum limit or specific value, if the alternatives must have or must not have the chosen cloud concept and if the requirement is also going to be a criterion or not.

3. **Add Criterion** - The user interacts with the system in order to insert data about a new criterion. These data includes: name, weight for the decision process, the preference direction (maximizing or minimizing) and the preference value if it exists (uses Interactions 5 and 6).

4. **Add Service Template** - The user interacts with the system to insert data about a new Service Template. The data includes: Service Template type and description. Then it is possible to add both requirements and criteria to the newly created Service Template (uses Interaction 2 and 3).

5. **Insert Weight** - The user interacts with the system to insert data about the decision weight for a specific criterion. This weight can also be asked by the system if needed depending on the decision method to be used.

6. **Insert Preference** - The user interacts with the system to insert data about the preference value for a specific criterion. This preference will be later used in the decision process. The system may also ask for a preference if needed.

4.2.4 Data Models

The problem this thesis proposes to solve relates to two different levels of data. One is clearly defined by the decision maker and groups the information about what the decision maker wants to find (service templates, requirements and criteria). The other is the service set where the search for alternatives that meet the first group of information is going to be performed.

With this in mind our architecture has also divided these two types of information into two data models:

- **Composite Service Architecture (CSA) Data Model** - Groups all the decision maker input information related to his needs. This information is the one gathered from the use case interactions depicted in Section 4.2.3.
- **Service Data Model** - Is a representation of the service description imported from the service set (Linked USDL triples) and is intended to represent available cloud services. This representation holds the service related data such as service features, price and service name.

4.2.4.1 CSA Data Model

As stated before the CSA Data Model hold all the decision maker input data. This means that this data model is a representation of the services and their characteristics that the user is looking for. It can also be seen as the overall template for the search engine where the alternatives (candidate services) must fit.

From Figure 4.10 we can see five entities:

- ***CSAData*** - is the wrapper entity that hold all the CSA entities listed bellow. Everytime a decision maker starts the application one *CSAData* will be created to hold his input data.
- ***ServiceTemplate*** - is the equivalent to a service in the aggregated solution. In other words, when creating a new *ServiceTemplate* the decision maker is telling the system that he wants to have a new group of functionalities in its final solution. Each *ServiceTemplate* will correspond to a search and decision process to first find the suitable alternatives an then decide which are the best.
- ***Requirement*** - is a special need specified by the decision maker for one of the two possible scopes: the CSA or a particular *ServiceTemplate*. A requirement can be exclusive or not. If a requirement is exclusive it means that all the candidates that do not comply will be ignored. However, if the requirement is not exclusive the candidates that do not comply will still be suitable candidates (alternatives) for the decision process. For a requirement to be exclusive it must meet at least one of the following constraints:

  - have a maximum value defined
  - have a minimum value defined
  - it is not a criterion

  Note that a requirement can be at the same time a criterion if the decision maker wishes so. In this case a Criterion is created and added to the same scope as the requirement.

- ***Criterion*** - is a service characteristic to have in consideration during the decision process. Holds the information needed for the decision to take place. This information is for example the weight, a preference value if it exist and a preference direction, either maximize, if we want the highest value possible for the criterion or minimize, if we want the lowest possible value.

- ***Result*** - is a wrapper entity used for storing the results of the decision process. It holds the alternative data plus the performance value calculated in the decision process.

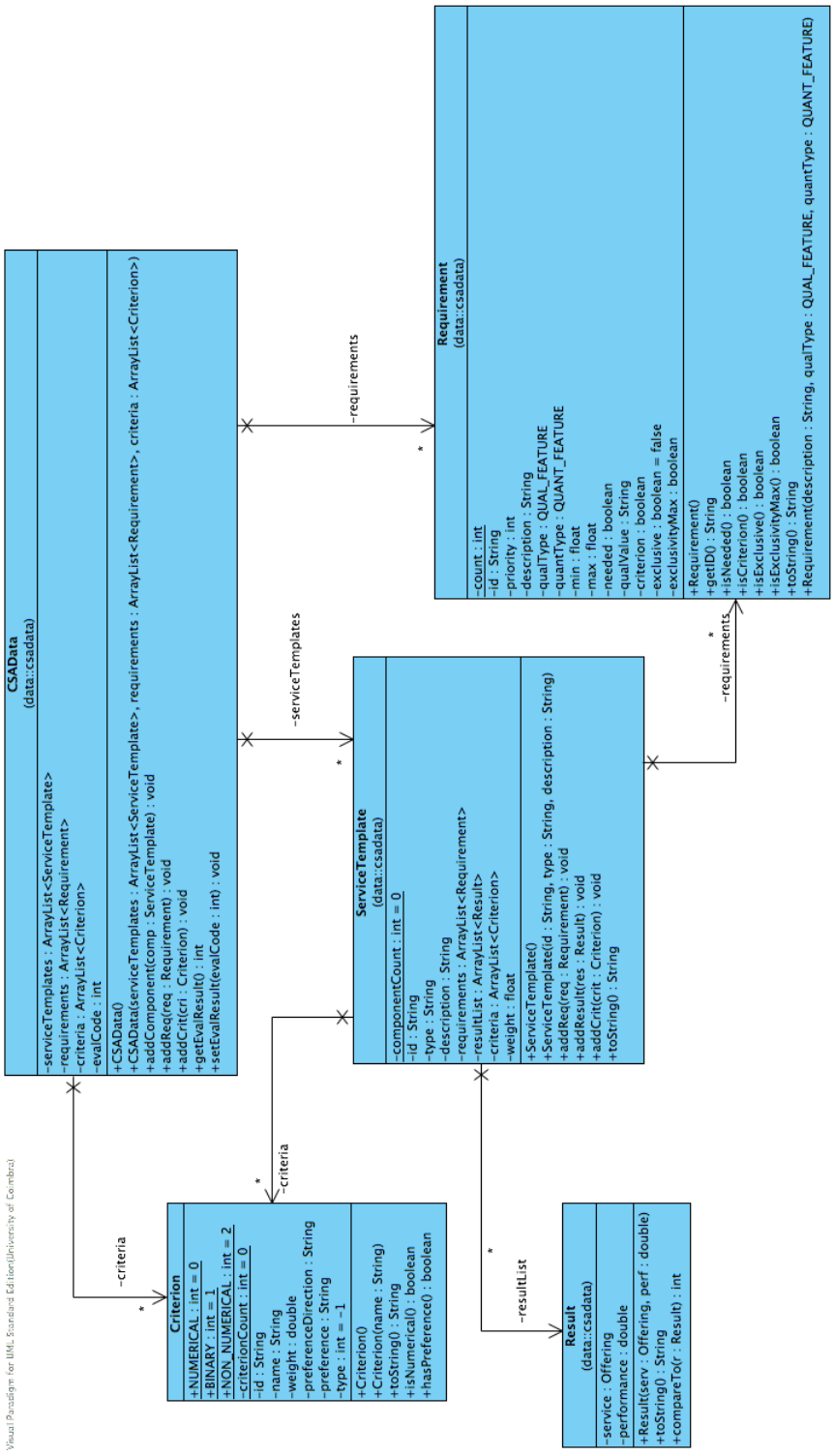


Figure 4.10: CloudAid Architecture: CSA Data Model

From Figure 4.10 we can see the relationships between entities:

- A CSAData entity can have a list of Requirements, Criteria and ServiceTemplates
- A ServiceTemplate can have a list of Requirements, Criteria and Results

4.2.4.2 Service Data Model

The Service Data Model is responsible for storing the service related data. Everytime a suitable candidate for a particular Service Template is found its data is converted from Linked USDL and stored in the Service Data Model.

From requirement INTR3 in Appendix C Table C.8 we take the need for this data model to be as similar as possible to the Linked USDL service description. Therefore, with this requirement in mind the following entities were created and can be seen in Figure 4.11:

- **Offering** - is the representation of the ServiceOffering class from Linked USDL Core module [100]. However, it holds more information than the original class for example it has all the service features. Since what the decision maker is looking for is a concrete offering of a specific service that meets his requirements, it was decided to make the Offering the main entity and store all the service data in it. It was considered redundant to have the Linked USDL model 100% replicated into the Service Data Model so, hereby relaxing the INTR3 requirement. Therefore, the Offering entity groups all the relevant Linked USDL classes and properties in one place.

  Besides the service features the Offering also holds the values for each of the criteria defined by the decision maker, also called attributes, as well as the calculation values for each of these attributes.

- **QualitativeFeature** - is a quality of the service which has a name, a description and a type. The type is related to one element of the QUAL_FEATURE list.

- **QuantitativeFeature** - is a quantitative property of the service, which has a name, a description, a numerical value, a type and a unit of measurement (Gb, Mb, GHz,...). The type is related to one of the element of the QUANT_FEATURE list.

Both QualitativeFeature and QuantitativeFeature entities were created having in mind the Linked USDL usage of the GoodRelations ontology [55], which divides concepts in two groups: the concepts that can be quantified (Eg: storage capacity or number of CPU cores,...) and the concepts that are not possible to quantify referred to as qualities (Eg: storage type, backup policies,...).

Another important need for this particular data model comes from requirement DR3 in Appendix C Table C.6, the use of an external ontology for cloud concepts. This lead to the creation of two lists of concepts to wrap the CloudTaxonomy (see Section 5.1) list of concepts:

- **QUAL_FEATURE** - holds all the qualitative characteristics of cloud services defined in the CloudTaxonomy.
- **QUANT_FEATURE** - holds all the quantitative characteristics of cloud services defined in the CloudTaxonomy.

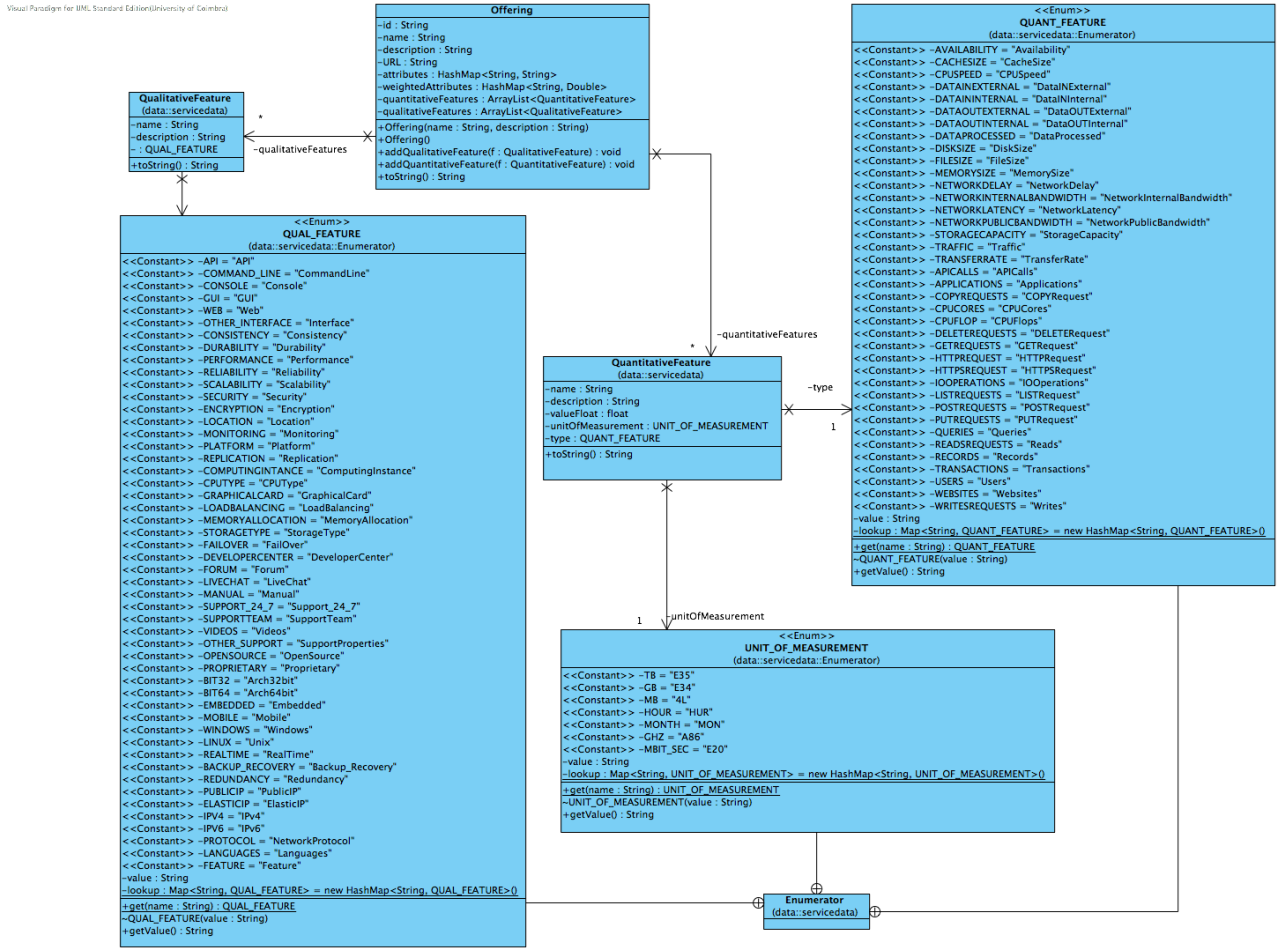


Figure 4.11: CloudAid Architecture: Service Data Model

Note that the GoodRelations approach for the two groups of concepts (Qualitative and Quantitative) was also applied in the distinction of the cloud concepts. The attributes are the values that the candidate (Offering) has for the criteria defined by the decision maker.

Although they are used for different purposes in the application, the CSA Data Model and the Service Data Model can be linked together at least at a conceptual level. An Offering is an alternative (suitable candidate) found by the search process that meets all the requirements of a particular ServiceTemplate. The attributes are the values of the found alternative for the defined criteria. So, if the decision maker defines a ServiceTemplate, T, with a list of Requirements, R, and a Criterion, C, after the search process the Offering will hold the data about a service offering, O, that meets the list of Requirements R and will have an attribute with the Service offering O value for the Criterion C.

4.2.5 Languages, Frameworks and Tools

An important phase in the design of any software system is to choose what languages, tools and frameworks should be used in the development. As we can see from the requirements list in Appendix C there are no constraints about the languages or special frameworks to use. Therefore, most of the choices had to do with personal preferences and knowledge of its use. Nevertheless, there were still some mandatory standards followed in order to cope with some requirements.

JAVA

The use of JAVA language was a personal choice, since the experience with this programming language was far greater than any other option. However, its versatility, community support and amount of different libraries is seen as advantages over other possibilities. Therefore, it was chosen as the development language for the CloudAid prototype application.

RDF Framework

As some requirements denote, the need for the interaction with RDF files is a key element. The Service Set is described with Linked USDL, a RDF model. Hence, the need for a framework to work with semantic concepts was a concern. Moreover, this framework had to be JAVA compliant since JAVA was the chosen language. The work in [53] presents a good analysis report of RDF tools and stores. However, since CloudAid has no major concerns about performance, the only real need is the compliance with both JAVA and the semantic web standards, RDF/XML, N3, SPARQL, OWL, etc.

The final decision was the use of the Apache JENA framework [2]. The main reason was the previous experience with this framework which could facilitate the development time, reducing the learning time that would come from the use of a new framework. Moreover, being one of the most widely-used frameworks allows a fair amount of confidence in its capabilities as well as its community for support.

The search engine is the module that will wrap this framework, particularly, the

Jena Engine component will be responsible to hold all the API calls to build and execute the SPARQL queries.

XMCDA JAVA Library
As already stated in Section 4.1 one of the key requirements is the system capability to use multiple decision methods and allow the later extension to other methods. This need lead to the use of a standard introduced by the Decision Deck group, XMCDA [34]. This standard is a XML schema specially conceived for describing MCDM concepts and data structures and its documentation can be consulted in [34]. This standard allows to model the decision problem in an uniform way, capable of being used by any method that also implements the XMCDA schema.

As the Decision Deck [34] states, the goals of XMCDA are:

- the interaction of different MCDA algorithms;
- the execution of various algorithms on the same problem instance;
- the visual representation of MCDA concepts and data structures via standard tools like web browsers.

In order to use this standard with the JAVA language the Decision Deck provides a JAVA API implementation that wraps all the methods to read and write XMCDA files. The CloudAid application uses this API in all the XMCDA import/export operations.

4.3 Test Plan

As part of the analysis and specification phase a test plan is presented as well as the overall considerations necessary in order to execute it. Thus Section 4.3.1 presents the testing environment where the tests are to be executed. Then from Sections 4.3.2 to 4.3.8 each presents the test plan for the categories of requirements listed in Appendix C: Funtional, Usability, Reliability, Performance, Supportability, the threee system constraints (Design, Implementation and Interface) and finally in 4.3.8 the integration tests are presented. Note that all the requirements mentioned in this section are presented in Appendix C.

4.3.1 Testing Environment

Before presenting the test plan it should be stressed the environment to be used during the entire testing phase. Since this prototype is a standalone version no connection to the internet is required. During the testing it was used the JDK 1.6 version however the application should be compliant with previous JDK versions. The machine specifications used for the entire testing phase are:

- **MacBook Pro 15-inches, mid 2010**
- **Processor**: 2.4 GHz Intel Core i5 with two cores

- **L2 Cache (per Core)**: 256 KB
- **L3 Cache:** 3 MB
- **Memory:** 8 GB 1067 MHz DDR3
- **Graphics:** NVIDIA GeForce GT 330M 256 MB
- **Operating System:** OS X 10.8.3

4.3.2 Functional Requirements

TFR1 Tests for functional requirement FR1 "Add a ServiceTemplate to the CSA":

- Verify if the system successfully adds a new ServiceTemplate with correct to the CSA Data.
- Verify if the system correctly adds ServiceTemplate weight in CSA Data.
- Verify if using non-numerical values as a ServiceTemplate weight is not allowed.

TFR1.1 Tests for functional requirement FR1.1 "Add a Requirement to a ServiceTemplate":

- Verify if the system successfully adds a new requirement with correct data to the ServiceTemplate.
- Verify if the system does not allow to add a new Requirement with wrong Requirement type.
- Verify if the system does not allow to add a new Requirement with bad field types.
- Verify if the system also adds a new Criterion if the added Requirement is defined as being also a Criterion.

TFR1.2 Tests for functional requirement FR1.2 "Add a Criterion to a ServiceTemplate":

- Verify if the system successful adds a new Criterion with correct data to the ServiceTemplate.
- Verify if the system does not allow to add a new Requirement with wrong Requirement type.
- Verify if the system correctly adds Criterion weight to the ServiceTemplate.
- Verify if using non-numerical values as a Criterion weight is not allowed.

TFR2 Tests for functional requirement FR2 "Add a Requirement to the CSA":

- Verify if the new CSA Requirement is correctly generalized to all Service Templates in the CSA.

Besides the above test list, all the tests performed in TFR1.1 are also used in TFR2, however the test scope is no longer the ServiceTemplate but rather the CSA.

TFR3 Tests for functional requirement FR3 "Add a Criterion to the CSA":

- Verify if the new CSA Criterion is correctly generalized to all Service Templates in the CSA.

Besides the above test list, all the tests performed in TFR1.2 are also used in TFR3, however the test scope is no longer the ServiceTemplate but rather the CSA.

TFR5 Tests for functional requirement FR5 "Search for services that fulfill the ServiceTemplate Requirement list":

- Verify if the System successfully returns all the available service offerings that fulfill multiple requirements at the same time.
- Verify if the System successfully returns all the available service offerings that fulfill a requirement of the type: "Services without feature X"
- Verify if the System successfully returns all the available service offerings that fulfill a requirement of the type: "minimum value for feature X"
- Verify if the System successfully returns all the available service offerings that fulfill a requirement of the type: "maximum value for feature X"
- Verify if the System successfully returns all the available service offerings that fulfill a requirement of the type: "specific value for feature X"
- Verify if the System successfully returns all the available service offerings that fulfill a requirement of the type: "minimum price value"
- Verify if the System successfully returns all the available service offerings that fulfill a requirement of the type: "maximum price value"

Functional requirements FR5, FR5.1, FR5.2, FR5.3, FR5.4 and FR6 can and will all be tested in the same test cases as FR5. This has to do with the need to test the system capability of searching for multiple requirements (FR5), this means that the test case can be built in a way that is possible to test all the above functional requirements at the same time. Each test case will directly deal with each independent functional requirement.

TFR7 Tests for functional requirement FR7 "Get the service values for the defined Criteria":

- Verify if the system collects the right features values for the defined Criteria.

TFR9 Tests for functional requirement FR9 "Read Linked USDL service descriptions":

- Verify if the system extracts all the ServiceOffering Qualitative and Quantitative Properties.
- Verify if the system correctly extracts the PriceComponents prices defined with the *hasPrice* property.

Since requirements FR9.1 and FR9.2 are children of requirement FR9 they are both tested in TFR9.

TFR11 Tests for functional requirement FR11 "Import/Export XMCDA decision data":

- Verify if the system exports the data from the decision problem to a XMCDA file.
- Verify if the system imports a XMCDA file and can read the results.

TFR12 Tests for functional requirement FR12 "Rank the services according to their decision value":

- Verify is the system can sort (from the highest to the lowest) the results retrieved from the XMCDA file.

Requirements FR13 to FR13.4 will be tested during the reliability tests in TRR8 since there was no point of testing the capability of the system of performing such tasks without the reliability of the resultant data.

TFR14 Tests for functional requirement FR14 "Generate Aggregated alternatives":

- Verify if the system generates valid aggregated solutions for the Admissible solution algorithm.

TFR15 Tests for functional requirement FR15.1 "Calculate the admissible aggregated solutions based on price requirements":

- Verify if the system can distinguish the admissible aggregated solutions based on price requirements.

TFR16 Tests for functional requirement FR16 "Decide which is the best admissible solution from the admissible solutions list":

- Verify if the system can compare and evaluate admissible aggregated solutions.

4.3.3 Usability Requirements

The usability tests are mainly concerned with user acceptance and user comprehension of the prototype interface. These tests would be performed using questionnaires to be filled by end-users. Some possible questions would be:

- Did you understand the questions asked by the application?
- Did you understand what information was being asked?
- Did you follow all the steps in the execution process from the Data Input to the Final solution?
- Did you have difficulties in proceeding to the next step because you did not understand what to do next?
- Did the application show the desired information at the right time?

Note that these questions are highly related to the usability requirements stated in the requirements list, as they try to answer to what extent are these requirement fulfilled.

4.3.4 Reliability Requirements

TRR1 Tests for reliability requirement RR1 "Correct normalization of ServiceTemplate Weights":

- Validate the result values for the normalization of ServiceTemplate weights.

TRR2 Tests for reliability requirement RR2 "Correct normalization of Criteria weights":

- Validate the result values for the normalization of Criteria weights.

TRR3 Tests for reliability requirement RR3 "Correct Decision Data Export/Import to XMCDA":

- Validate the resultant XMCDA file with the decision problem data.
- Validate the data extracted from the XMCDA file with the decision results.

TRR4 Tests for reliability requirement RR4 "Correct service price calculation":

- Validate the calculation result for the price of a ServiceOffering.

TRR5 Tests for reliability requirement RR5 "Correct decision on the Admissible Solutions Algorithm":

- Validate the list of admissible solutions returned by the Admissible Solutions Algorithm.

TRR6 Tests for reliability requirement RR6 "Correct calculation of the overall admissible solution decision value":

- Validate the final decision value assigned to each of the admissible solutions.

TRR8 Tests for reliability requirement RR8 "Correct normalization of Decision characteristics":

- Validate the result values for the normalization of Numerical Decision characteristics.
- Validate the result values for the normalization of Binary Decision characteristics.
- Validate the result values for the normalization of Non-Numerical Decision characteristics.

4.3.5 Performance Requirements

TPR1 Tests for performance requirement PR1 "Admissible Algorithm performance with high amount of alternatives":

- Check response time of the admissible algorithm with different amounts of alternatives combinations.

TPR2 Tests for performance requirement PR2 "Allow the search of a high number of service descriptions":

- Check the difference in response time for searching a ServiceSet with increasing number of triples.

4.3.6 Supportability Requirements

The tests for supportability requirements are highly conceptual and difficult to test. However, by designing and executing the Test Plan we can ensure the system testability, hence ensuring requirement SR1,"Testable System".

The SR2, "Allow the use of different Decision Methods", and SR3, "Allow the ease of functionality extension", are also justified by the prototype implementation itself. By using the MVC model and increasing the modularity the ease of functionality extension should be ensured by itself. SR2 Requirement should also be justified by the usage of two different MCDM methods.

4.3.7 Design, Implementation and Interface Requirements

These three categories introduced by FURPS+ in addition to FURPS were considered, as the specification itself states, system constraints. Thereby, the testing of these requirements is not possible beyond the usage verification of the standards and procedures listed.

Some requirements such as DR1, IMPR1 or IMPR2 are highly conceptual and can only be attested by analysing the system architectural design. Others, such as DR3 or INTR2 were already validated through the functional requirements testing.

4.3.8 Integration Tests

While in the previous sections the tests were oriented to each module in the system, this phase aims at grouping all the modules and test its interaction.

These tests will validate if the system is working as it should as a whole and if the results are reliable.

SIT System integration tests:

- Validate the results passed from each module to the next.
- Verify if the execution flow follows the right steps.

5
Semantic Models

The objective of this chapter is to present the two semantic models developed herein in this thesis. This models are part of the research effort and are important elements to consider during the development of the CloudAid Prototype. Thus, Section 5.1 discusses the CloudTaxonomy, an ontology of cloud concepts. Section 5.2 presents and discusses the Linked USDL Pricing module developed in collaboration with KMI [1] and SAP Research [2].

5.1 Cloud Taxonomy

As part of the CloudAid project the CloudTaxonomy was a key step to achieve the desired results for the application prototype. In this chapter are explained all the CloudTaxonomy related topics. Section 5.1.1 presents the overall objective of the Cloud Taxonomy and why the need for such artifact. The methodology followed in both the research and development are described in Section 5.1.2. Section 5.1.3 presents the related work in Cloud Ontologies and their use in recent research projects as well as some rational on some of the decisions that lead us to the final result. The full description of all the modules can be found in Appendix F.

5.1.1 Objective

While tring to describe cloud services we found ourselves with the difficulty to distinguish service features and characteristics. However, at the same time, it was clear that most of the services use a common set of characteristics to describe their services and its features. Amazon EC2 [3] for instance describes their computing instances regarding CPU, memory, storage, number of cores and so on. The same happens with GoGrid [4] that uses CPU cores, RAM and Storage. Although these characteristics might have different names they relate to exactly the same thing. The Amazon EC2 Memory is the same concept as the GoGrid RAM.

[1] http://kmi.open.ac.uk/
[2] http://www.sap.com/index.epx
[3] http://aws.amazon.com/ec2/instance-types/
[4] http://www.gogrid.com/products/pricing

This different way of describing the same concept perpetrated by service providers highly increases the complexity of any search engine willing to find services with a defined set of characteristics. While a user may be asking for a service with 2Gb of Memory the service may have RAM in its description. This is exactly where the semantic web can be of crucial importance. By attaching semantic knowledge to a specific concept we can ease description comparability.

By using an ontology that groups these concepts, any service provider can enrich their service description by annotating its features with extra semantic information, allowing not only search engines to find their services more efficiently but also consumers to better understand what the features are all about. As the authors state in [89] "the main problem in Cloud Computing is the lack of unified standards.". Also by the Open Cloud Manifesto [77] one of the key challenges for increasing cloud adoption is interoperability. Therefore, with all these concerns in mind and from [89] we identify the key objectives for using a Cloud Ontology:

- Enable standardization of cloud service descriptions
- Enable interoperability between different service providers
- Enable intelligent discovery of services
- Enable service composition/aggregation

5.1.2 Methodology

With the objective clearly defined, a several steps process was devised in order to achieve it and asses possible development paths. With this in mind we followed these 5 steps:

1. **Identify key service characteristics**
   This was mainly a search task, where a group of major cloud services was analysed and their characteristics were extracted. This first step gave us the insight needed for understanding which are the common service characteristics described by the service providers for their services and that we want to described in an Ontology. All kinds of XaaS were analysed and their characteristics grouped into categories (e.g.: Storage as a Service, IaaS, PaaS, SaaS,...).

2. **Search for suitable alternatives to describe these characteristics**
   After obtaining a list of cloud services characteristics we searched for work already done to group these concepts in an ontology (see Section 5.1.3).

3. **Compare and contrast ontologies alternatives**
   From the different cloud ontologies available the advantages and disadvantages of each had to be considered. Some had not enough info, or were used for completely different scenarios or had different objectives as it will be explained in Section 5.1.3. A good example of the results produced in this step is Figure 5.1, also explained in Section 5.1.3.

4. **Develop our own Cloud Taxonomy**
   Having reached a decision that none of the alternatives is suitable for our purpose, the next step was to start the development of our own ontology to describe all the concepts extracted from step 1 (see Appendix F).

5. **Apply the taxonomy in real services and refine it**
   In order to test and validate the final result an iterative process started. Every time a new service was described with Linked USDL the Cloud taxonomy was used to annotate the service features. If the service needed extra information, or if with the current state of the taxonomy it was not possible to properly describe the service feature, the taxonomy was revised and updated. This step was concurrent with the Linked USDL Pricing model development (see Section 5.2) as all the services used as Pricing examples were also used for the Cloud Taxonomy validation.

5.1.3 Cloud Ontologies

In [134] the authors present a first approach to a Cloud Computing ontology outlining its components and their relationships. However, there is a clear service orientation as the different cloud layers [84] are distinguished from one another. This service oriented approach is also followed by several other authors as in [109], where a taxonomy is developed in order to describe the cloud architecture and then map the various cloud providers. Although these are interesting steps towards the standardization of the cloud, they fail at describing the actual resources provided in a cloud service. Our goal is not to describe the service itself but rather capture its features and provided resources.

The work presented in [47] and [89] by the mOSAIC project takes us a step closer towards our goal. It clearly defines a list of cloud concepts to be described by the ontology. Furthermore, the mOSAIC shares some of the same objectives, "Very few efforts have been done in order to propose a unified standard for Cloud Computing. This is a problem, since different Cloud systems and vendors have different ways to describe and invoke their services, to specify requirements and to communicate. mOSAIC project addresses these problems by defining a common ontology and it aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by users." From [47] we extract a list of cloud services concepts captured by the mOSAIC ontology:

- Service Models
- Service Deployment Models
- Service Capabilities and/or Functional Properties
- Service Availability
- Service Non-Functional Properties
- Service Level Agreements

- Security and Quality of Service
- Service Characterization
- Service Classification
- Service Resources

Since our cloud services are fully described with Linked USDL some of the above topics, such as, *Service Level Agreements* and *Service Models* are already covered. This shifts our interest to other topics such has *Service Resources*, *Service Capabilities and/or Functional Properties* and *Service Non-Function Properties*. Once again the aim is not to describe the service itself but provide the means to ease the search process for cloud resources and capabilities provided by a cloud service.

Notes: Services from http://cloudtaxonomy.opencrowd.com/taxonomy/

label:

✓	Described by mOSAIC
Partial	mOSAIC may have a top Class but it is not enough
✗	Not described by mOSAIC

Data	
	mOSAIC
Storage	✓
GET Request	✗
Delete Request	✗
PUT Request	✗
Copy Request	✗
Post Request	✗
LIST Request	✗
Redundancy	✗
Transfer IN	✗
Transfer OUT	✗
Transfer Rate	✓
Bandwith	✓
File Size	✗
Data Processed	✗
Cache Size	✓
Queries	✗
Writes	✗
Reads	✗
Type	Partial
Transactions	✗
Records	✗

Authentication	
	mOSAIC
OATH	✓
Users	✓
Token Authentication	✗
OpenID	✗

Interface	
	mOSAIC
API	✓
Command Line	✗
Console	✗

IaaS	
	mOSAIC
Operating System	✓
CPU Architecture	✓
CPU Flops	✓
CPU Cores	✓
CPU Speed	✓
CPU Type	✓
Memory Allocation	✓
Memory Size	✓
I/O operations	✗
HTTP Requests	✗
HTTPS Requests	✗
DiskSize	✓
Load Balancing	✗
Network Public Bandwith	✓
Network Internal Bandwith	✓
Network latency	✓
Network Delay	✓
Public IP	✗
Elastic IP	✗
IPv4	✗
IPv6	✗

Others	
	mOSAIC
Availability	Partial
Durability	Partial
Consistency	Partial
Reliability	Partial
Scalability	Partial
Location	Partial
Backup/recovery	Partial
License	✗
Monitoring	Partial
Encryption	Partial
Platform	✓
Consistency	✓
Replication	✓

PaaS	
	mOSAIC
Laguange	Partial
Users	✗
API Calls	✗
Websites	✗
Traffic	✗
Applications	✗

Support	
	mOSAIC
Videos	✗
LiveChat	✗
SupportTeam	✗
"24/7"	✗
Forum	✗
Developer Center	✗

Security	
	mOSAIC
SSL	✗
SSH	✗
Firewall	✗
VPN	✗
Anti-virus	✗

Deployment	
	mOSAIC
Combined Cloud	✓
Private Cloud	✓
Hybrid Cloud	✓
Private Cloud	✓
Public Cloud	✓
Community Cloud	✓

Model	
	mOSAIC
IaaS	✓
PaaS	✓
SaaS	✓
BPaaS	✓

Figure 5.1: Cloud Service Characteristics mOSAIC Comparison

Several other papers point to the use of a cloud ontology to enhance search capabilities. The work in [52] develops a cloud service discovery system and makes use of a cloud ontology to reason about the similarities of different cloud services. This is an interesting approach, however, our problem in not to discover the services, since we already have a full service set of service descriptions based on Linked USDL, but rather decide which of them fulfill the user set of requirements.

Nevertheless, the work developed in [52] and later in [59] and [58] named as Cloudle project, is significant to prove the enhancement capabilities of a cloud ontology in a cloud service search mechanism. However, at the time it was not possible to access the full description of the ontology, hence, hampering the assessment of possible application to our problem.

Since we could not analyse what seemed to be the most promising option, the ontology used in Cloudle project, the other alternative was the mOSAIC [5]. Therefore, a deeper analysis was performed to the mOSAIC ontology in order to assess the degree of completeness for describing cloud resources. Figure 5.1 shows these comparative analysis.

The comparison was made with regard to the list of cloud service characteristics extracted from an analysis of all the providers listed in the Cloud Taxonomy by the OpenCrowd [94]. From Figure 5.1 we can see that mOSAIC fails at describing most of the cloud service characteristics. This lead to the creation of our own Cloud Taxonomy. This Cloud Taxonomy, however, tries to build upon the previous work and uses some of their concepts.

5.2 Pricing Model

Besides the Cloud Taxonomy the other semantic model used was Linked USDL. This time the objective is to describe the Cloud Services themselfs instead of the cloud concepts used by them. Linked USDL is still under development and some of its modules were still very primitive. Therefore, we saw a good collaboration opportunity to leverage the work already done as well as to improve a key element for this thesis main purpose: providing a mechanism to help decision makers to decide which cloud services to aggregate based on the their requirements and criteria.

5.2.1 Motivation

It is hard to dissociate Cloud Computing from the cost reduction aspect since it has been pointed out as one of its biggest advantages. This topic as already been greatly discusses in the literature [10]. In fact, the consensus about the cost reduction importance comes not only from the academic world but also from the actual market. According to the CloudWatch study of 2012 [27] from Cisco "In 2011, reducing cost was the #5 benefit of cloud. In 2012, this is the #1 benefit (57%)...Cost savings are widely known to be the primary driver for adopting cloud services, with promises

[5]The relevant documents about the mOSAIC project were kindly provided by ing. phd Pasquale Cantiello to whom we thank

of reduced capital outlay.". Thus, we can conclude that one of the most important non-technical aspect evaluated when enterprises think in migrating to the cloud is the price.

In this thesis we propose a mechanism to help decision makers to decide which cloud services to use and aggregate for their specific requirements and criteria. With this in mind an aspect such as price cannot be ignored, it should instead be treated with special interest. Consequently, the service description language must have the means to describe the relevant information about services pricing models.

However, it seems that nothing substantial has been done lately to formalize the pricing aspect. As the USDL authors state in [20], "...there seems to be a lack in formalizing non-technical aspects of a service, such as pricing, benefits, quality of service or legal requirements. USDL is an attempt to include such information in a structured way.". In fact, USDL through its business component tried to capture the pricing models and formalize them.

In the semantic field we find ontologies such as GoodRelations [55] that capture pricing components such as values, currencies, payment methods and so on, which by themselves are important. However, they fail at providing a suitable structure to describe pricing models and pricing plans.

Thus, we wanted to endow the Linked USDL with its predecessor capability to describe pricing. However, keeping Linked USDL main objective simplicity.

5.2.2 Challenges

Cloud Services are widely known by their "pay-per-use" model or as the author defines in [10], "Pay-as-you-go". This means a dynamic pricing model adapted to change based on the customer usage of the service (Gbs, Hours, etc). This was in fact the biggest challenge faced when tackling cloud pricing models. Rather than simple price tags, characteristic from products or even most of common services, with Cloud Services we were looking at highly dynamic models where not only the base price is calculated based on the usage but also discounts or freebies are given based on it. This dynamic price models are usually defined with formulas, thus a mechanism to describe this formulas and if possible calculate the final price within our model would be the best option.

Another challenge was to capture the correct affectation of the price to the Service Offering properties. It usually happens that a Service Offering is composed of different services (bundles) and depending on those services and their properties the overall price varies.

Nevertheless, we still wanted the final model to be comprehensible and applicable by those who could really use it to describes their services.

5.2.3 Methodology

In collaboration with the KMI [6] and SAP Research [7], we decided to meet weekly, for discussing the model and the progresses being made. This lead approximately

[6] http://kmi.open.ac.uk/
[7] http://www.sap.com/index.epx

to four months of development.

One of the proposed objectives was the simplicity of the pricing model, allowing for an easy understanding of the model by anyone willing to use it. Therefore, our goal was not to be able to describe 100% of the services pricing models, but it would suffice with 80%, or at least the most common pricing models used in Cloud Services as is pay-per-use or table based pricing.

This lead to an use case driven approach. A first step of identifying critical services was fundamental in targeting a few cloud services that could be representative of most of the Cloud Services pricing characteristics and at the same time serve as example for future service modelings. Three were identified:

- **Amazon EC2** [8] - probably the most complex Cloud Service in the market with many different pricing models and services included. An excellent example of top complexity to achieve.
- **Heroku** [9] - simpler than amazon but interesting due to its almost pure pay-per-use model.
- **SugarCRM** [10] - a simple table based pricing model. It was used in later steps for ensuring the model simplicity when describing simple pricing model.

Based on these three cases the model was iteratively discussed and improved in order to cope with the challenges raised by each of them. By using a ticket system to manage issues we were able to tackle problems as they were emerging as well as documenting decisions.

All these use cases descriptions and examples can be consulted in the Linked USDL Pricing repository [97].

5.2.4 Model

The final model is not a standalone version to describe service pricing, instead it integrates with the Linked USDL Core module in order to enhance the overall service description with the pricing capability. Thus, the classes and properties defined under the Linked USDL Pricing module are strongly linked to the Linked USDL Core module. Moreover, using the reusability concept characteristic from the semantic web, the Pricing module uses some concepts defined in GoodRelations [55] (an overall requirement for Linked USDL) and in SPIN [67] [68]. Among the most important are:

- ***PriceSpecification*** (GoodRelations)
- ***QuantitativeValue*** (GoodRelations)
- ***QualitativeValue*** (GoodRelations)
- ***Funtion*** (SPIN)

[8] http://aws.amazon.com/ec2/pricing/
[9] https://www.heroku.com/pricing
[10] http://www.sugarcrm.com/page/editions-pricing/en

5.2.4.1 Dynamic Pricing

SPIN as the authors define it in [67] "is a W3C Member Submission that has become the de-facto industry standard to represent SPARQL rules and constraints on Semantic Web models." and it is part of the solution to solve the dynamic pricing challenge. By allowing SPAQRL rules within the semantic model SPIN allows to do things such as calculate the value of a property based on other properties. This is exactly what we need in order to model dynamic pricing.

A dynamic pricing is usually composed of several variables that influence the price (also called metrics) and some constants usually defined by the provider himself. These constants commonly relate to the actual value to pay for each unit evaluated by the metrics. The metrics are the units of measurement for these constants. In other words a provider says that a customer has to pay €0.05 per Gb used. The value €0.05 is the constant defined by the provider and the Gb the metric used to measure usage. However, the price cannot be calculated prior to the user usage. Lets say that a user has indeed used 10Gb this month this would mean a final price of €0.5. What SPIN allows us to do is to extract the values stored somewhere else in the model and compute them, achieve a result. Thus, by defining these constant values and the usage variables we can create a formula to calculate the final price. Although the example is a simple linear function where the number of Gb used is multiplied by the price per unit. There are cases were complex functions are need, for discount when a certain amount is used for example. All this can be achieved using SPIN.

The use of SPIN functions is quite similar to the Object-Oriented approach in programming [66]. By defining these functions each of them with a formula for calculating its price we are even allowing for the creation of templates reusable for various price calculations.

The final step was to model the variables to be referenced in this functions. The approach was simple, two types of variables may exist: usage and constants. The usage variables are those which value is not known by the model prior to the service execution and as the name suggest they refer to usage or consumption of resources or service features. Constants, despite its name are also variables, but this name they are known beforehand. In the example above they would represent the unit price, €0.05 per Gb. This two types proved to be sufficient in all the Use Case examples.

5.2.4.2 Model Specification

The Linked USDL Pricing module has a total of six classes, however they are strongly linked to external ontologies, as already stated: GoodRelations (*gr* prefix) and SPIN (*spin* prefix). Figure 5.2 shows the relationships between all the classes involved. Note that the *ServiceOffering* is of the upmost importance since it is the link to the service itself (refer to [100] for the specification of Linked USDL Core Module).

PricePlan

A PricePlan is a set of charges associated with a network-provisioned entity. Al-

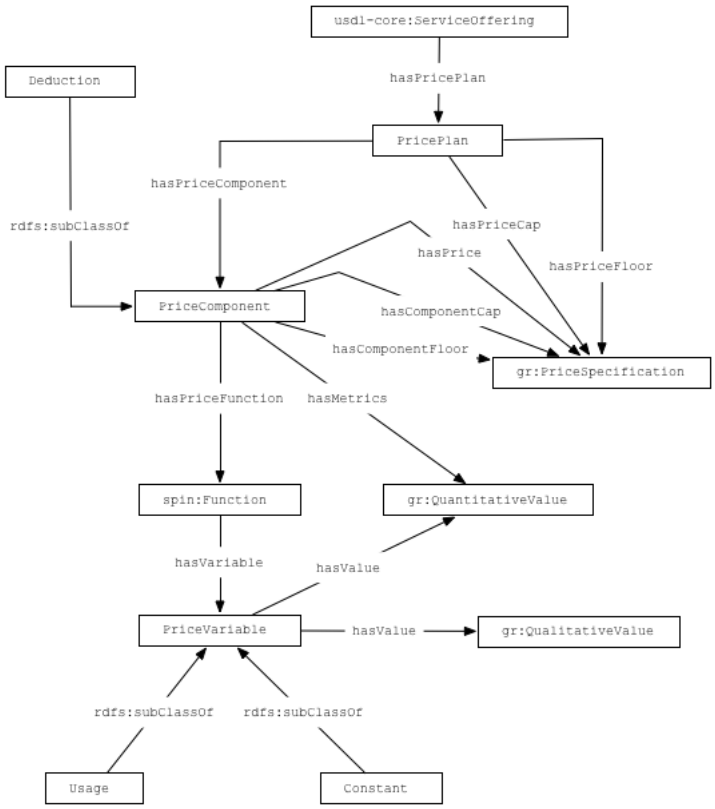


Figure 5.2: Linked USDL Pricing Module

ternative sets of fees (i.e. alternative PricePlans) choose from, for example to offer the consumer the choice between a flat price scheme and a usage-based scheme (a common practice in the telecommunication industry). Each PricePlan has is own price scheme, composed of one or more *PriceComponents*. A PricePlan belongs to the *ServiceOffering* which might only have one PricePlan. If a service has different price schemas to define different price strategies these correspond to different PricePlans that should be defined under different *ServiceOffering* as they in fact correspond to different service offerings as well. For instance: a service provides 10Gb of storage for €1 per month in its basic plan, but in its premium plan it provides 50Gb of storage for €2 per month including extra support. These although related to the same service are indeed two distinct service offering each with his own PricePlan. The model reflects this concept.

Properties with PricePlan as Domain:

- *hasPriceComponent* - A price plan consists of a number of price components that are added to the total price.

- *hasPriceCap* - A upper limit for the price. Providing this maximum PriceComponent value prevents the component final price from exceeding a certain amount, regardless of its levels and the parameters they are indexed to. For instance: A cap may be used to set an upper limit for a component whose levels vary with usage.

- *hasPriceFloor* - Lower limit of the price. Providing this minimum PriceComponent value prevents the component final price from falling below a certain amount, regardless of its levels and the parameters they are indexed to. For instance: A floor may be used to set a lower limit for a component whose levels vary with usage.

PriceComponent

PriceComponents are fees included in a PricePlan, which subject to conditions (expressed as PriceFunctions) may contribute to the total amount charged. Components within the same plan are added together in order to get the total amount (price of the service offering). Common examples of PriceComponents that may coexist in the same PricePlan are: startup or membership charges (to access the service), periodic subscription fees (with a certain recurrence - e.g. monthly - as long as committed to by the contract), pay-per-unit charges (whose total will be proportional to the metered usage), options or feature dependent charges. The final value of the component will depend on the PriceFunctions calculation or a tagged price if defined.

Properties with PriceComponent as Domain:

- *hasPriceFunction* - The SPIN price function to calculate the PriceComponent price.
- *hasPrice* - The pricing specification for a price component if the price is a fixed value.
- *hasComponentFloor* - Similar to the *hasPriceFloor* property but related to the PriceComponent.
- *hasComponentCap* - Similar to the *hasPriceCap* property but related to the PriceComponent.
- *hasMetrics* - By which metrics is the price calculated. Usually the price is multiplied by a factor per period or per amount of use.

Deduction

Deduction is a special case of *PriceComponent* it correspond to a negative priceComponent. The total price will be reduced by a certain amount. Thus, when adding all the price components is the price plan one should have in mind that Deductions should be subtracted.

PriceVariable

A price variable can be used for price function expressions of dynamic price models. It has a name (rdfs:label) and a quantitative or qualitative value. Variables can be referred from different price functions.

Properties with PriceVariable as Domain:

- *hasValue* - It hold the *gr:QuantitativeValue* or *gr:QualitativeValue* of the *PriceVariable*. The reason for using both types it that we may have qualitative

variables such has the type of the operating system being used or a particular feature of the service and not only quantitative values such has number of hours or number of instances. Note that the range of this property is *gr:QualitativeValue* and *gr:QuantitativeValue*.

Constant

The *Constant* price variables are provider specific variables known prior to the service execution. They are usually fixed values. For instance: €0.05 is the provider price per hour of an instance.

Usage

Usage price variables represent unknown values prior to the service execution. Specially used for usage purposes and dynamic pricing (e.g. pay-per-use models). Its value might come from user input or from provider monitoring (i.e. GB of data transferred last month).

5.2.4.3 Final Model Considerations

As already explained the *ServiceOffering* is the link between the Pricing Module and the Core Module. Only one *PricePlan* may exist per *ServiceOffering*. Not also that the final price is a sum of all the *PriceComponents* and the subtraction of all the *Deductions*. However, each PriceComponent may have its price defined in three different ways:

- Using the *hasPrice* property directly providing an instance of *gr:PriceSpecification*. This is the common use for products or services that ha a fixes price value. Eg: The premium subscription of a website costs €10 per month.
- Using the SPIN functionality through the *hasPriceFunction* property. Common in dynamic pricing when the final price must be calculated based on usage. These usage values must be collected through the *Usage* variables and then applied to the function together with *Constant* price Variables to achieve the final value. Eg: A customer pays €0.05 (*Constant* price variable) for each running instance per hour used calculated in the end of the month. the number of hours and the number of instances are *Usage* price variables.
- Using the *hasPriceFunction* although not specifying any SPIN *Function* but rather the formula in textual form (rdf:comment). This is the option for those who are not familiar with SPARQL or SPIN but still want to use a formula for calculating the price. Note however, that this is merely a textual description that has no price computing power.

The reason for such an approach was to keep the simplicity of the model. By using the SPIN functionality we were aware of the complexity increase inherited by the SPIN model itself. However, by allowing users to not use this capability we are ensuring the previously achieved simplicity while at the same time supporting the required functionality for dynamic pricing. Note that despite the that in this thesis we

were looking for a solution to describe the complex pricing models present in Cloud Services, the Linked USDL Pricing Module is intended to be used independently of the domain.

The reutilization of the *GoodRealations* should provide a fair amount of market acceptance since this is already an established ontology. Moreover, its definition of some pricing concepts is also important for keeping our module as simple as possible.

5.2.5 Final Remarks

By following a use case driven approach we intended to focus on the key aspects and challenges that the current market raises regarding pricing models. This approach was highly beneficial since despite the small duration given to the project we were able not only to develop but to test the model at the same time. Moreover, the use of real services in the use cases allows to proof that the model can in fact be successfully applied to describe Cloud Services as well as making these "living" examples of how to do it.

Ensuring the model simplicity even when using complex models such as SPIN is indeed difficult to achieve, however, due to its optionality we intend to reduce the rejection of using such approach and at the same time allowing the modelling of complex pricing schemas such as pay-per-use. This is indeed achieved since only three concepts must be defined in order for a simple service to have its price defined: *PricePlan* with a link to a *PriceComponent* and *gr:PriceSpecification* through the hasPrice property.

Full examples and the model itself can be found in the Linked USDL Repository [97].

6 CloudAid Prototype

This chapter provides a detailed view on the CloudAid prototype development process and its components. The objective is to share the rational behind some of the decisions and paths of development chosen. However, it will be adopted a higher level perspective over the prototype, rather than a code oriented perspective. There are more advantages in this approach since there are no major difficulties in the code itself but rather in the prototype logic. Thus, Section 6.1 starts by introducing the methodology used during the development of the project. Section 6.2 begins the CloudAid Prototype explanation by introducing the methods for data capture and some related topics. Continuing to Section 6.3 we present the links between the Linked USDL and the application itself. Then from Section 6.4 to 6.8 each module is presented in some detail, explaining its purpose in the overall prototype and the rational for its implementation decisions.

6.1 Methodology

Being this a research work, the prototype application development is highly conceptual, meaning that its purpose is to prove the concepts being addressed by the research effort. Nevertheless, it must be planned and developed as any other software application, however, with some special constraints.

There was the need for quick development of key requirements, those that were fundamental for a proof of concept, both for testing and assessment of the problems found during the development in order to discuss the best possible courses of action. These would suggest an agile methodology [116].

Although the key requirements were previously identified, most were only possible to see when some "digging around" has been done. Therefore, most requirements were gathered iteratively at the same time as the implementation.

With this being said, the methodology chosen was the Rapid Application Development (RAD) [80] which allowed a less planning concerned approach in order to produce faster results and at the same time allowing the needed flexibility with requirements. The RAD approach also foresees the high interaction and participation of the users in the development process. The fact is that there were no real users interacting with the development but the continuous discussion and improvemen-

t/alteration of requirements was a fundamental characteristic to extract from this methodology.

The development of the prototype application was divided in a series of smaller problems, or smaller prototypes. Each passed by a series of iterative steps, from analysis to development and preliminary testing/validation. The final result was a prototype of a particular component of the final application. These prototypes were the different modules of the application and their development will be explained in more detail in Sections 6.5 to 6.4. This approach resulted from the RAD methodology and helped to understand the key problems to solve as well as keeping the results flowing. Since every module was being developed separately one at a time, without the need for the others, two key aspects were also favoured: extensibility and modularity.

6.2 User Data Capture

Having a component of decision aid rather large, this prototype is highly data oriented. The whole purpose is to provide a decision aid mechanism able to take advantage of the user data about the needed cloud services and his preferences. Therefore, there is the need for the user to insert data about four major topics:

- Service Templates
- Requirements
- Criteria
- User Preferences.

While the first two topics are directly related to the system that the user is trying to build, the bottom two are metrics and preferences to help the CloudAid prototype to find the best options to fulfill the first two topics.

With the exception of the User Preferences, which may be collected throughout the prototype execution when needed, the information about the other topics is collected prior to any computation. The user is prompted for Service Templates, Requirements and Criteria data before anything else. In other words, the Service Templates, Requirements and most of the Criteria information work as the input data for the CloudAid Prototype.

From Section 4.2.4.1 we can see the data structure used two store all this information. However, this information must be somehow collected. The chosen process was to create a simple "shell based" user interface to allow the user to specify all the relevant information. This way the user should be able to build any type of complex CSA according with his needs.

Figure 6.1 shows the simple menu with which the user interacts to insert CSA Data. By choosing one option the user is allowed to insert data about a new Service Template, a new global requirement or a new global criterion.

Figure 6.2 shows the second menu, invoked when the user chooses to insert a new Service Template. Once again by choosing one option the user is allowed to insert

```
CSA DATA:
1 - New Service Template
2 - New Requirement
3 - New Criterion
0 - DONE!!!
```

Figure 6.1: CloudAid Prototype: CSA Menu

```
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
```

Figure 6.2: CloudAid Prototype: Service Template Menu

data about a newly created Service Template, a new requirement or a new criterion for the newly created Service Template. Figure 6.3 shows the normal interaction when adding a new Service Template, while Figure 6.4 shows Requirement and Criterion data being inserted for the previously created Service Template.

Through this simple "shell based" interaction the user is able to build the CSA data for representing the system he wants to build. This CSA Data is then stored and used by the CloudAid Prototype in its processes.

Requirement

A requirement is a specific resource or capability need that the user has for a certain Cloud Service. Although they are transparent to the user, they control much of the operations performed by the Prototype, specially the Search Engine (Section 6.6).

As we can see from Figure 6.4 the first piece of information the user has to insert is the requirement type. This type is a resource or service capability/functionality to which the requirement is linked to. From the example in Fugure 6.4, by specifying the type as *StorageCapacity*, the user is saying that this particular requirement is related to a specific resource to be provided by the potential cloud service: Storage Capacity.

Note that all these requirement types come from the CloudTaxonomy which is fully described in Appendix F. The reason from adopting such an approach was simple and is linked to the objective of the Cloud Taxonomy itself. The increasing amount of Cloud providers, each with his own taxonomy of cloud concepts, makes almost impossible to match the set of requirements of a customer to the providers service description. For instance: If the provider A says his service has a "Disk Space" of 750Gb, and a user is looking for a service that provides a minimum of 500Gb of "Storage", it might be impossible to match the service from provider A wuth the user requirement due to the usage of different concepts, although they mean the same.

There is however, one type that is not defined in the CloudTaxonomy: Price. The price is considered a special type and in different parts of the CloudAid Prototype is indeed handled differently from the remaining types, which will be explained

```
CSA DATA:
1 - New Service Template
2 - New Requirement
3 - New Criterion
0 - DONE!!!
1
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
1
Please specify the Service Template Type:
database
Please specify the Service Template Description:
this is my database cloud service
Please specify the Service Template decision weight:
3
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
```

Figure 6.3: CloudAid Prototype: Insert new Service Template

```
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
2
Please specify the Requirement Type from the list of Cloud Concepts, or write 'Price' for a Price requirement:
StorageCapacity
Does this requirement has a limit value? (Y/N)
y
Please specify the limit:
500
Is it a minimum or maximum limit? (min/max)
min
Please specify a requirement description:
My database cloud service must have at least 500Gb of Storage
Will this requirement also be decision criterion? (Y/N)
y
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
3
Please specify the Criterion Type from the list of Cloud Concepts:
Price
Please specify the criterion decision weight:
3
Do you want to maximize the Criterion value? (Y/N)
n
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
0
CSA DATA:
1 - New Service Template
2 - New Requirement
3 - New Criterion
0 - DONE!!!
```

Figure 6.4: CloudAid Prototype: Insert new Requirement and Criterion

```
SERVICE TEMPLATE DATA:
1 - Insert Service Template Data
2 - New Requirement
3 - New Criterion
0 - DONE!!!
2
Please specify the Requirement Type from the list of Cloud Concepts, or write 'Price' for a Price requirement:
StorageType
Do you want to specify a particular value for this service feature? (Y/N)
y
Please specify the value:
SSD
Is this a feature that you want to have (Y) or you don't want to have(N) in a service? (Y/N)
y
Please specify a requirement description:
i want my database to have SSD storage
Will this requirement also be decision criterion? (Y/N)
n
```

Figure 6.5: CloudAid Prototype: Insert a Qualitative Value Requirement

whenever necessary. However, in respect to the requirements the price type is dealt as any other requirement.

Another important distinction is between *Quantitative* and *Qualitative* requirements. This has mainly to do with the Linked USDL usage of the GoodRelations ontology [55] which distinguishes from *QualitativeProductOrServiceProperty* and *QuantitativeProductOrServiceProperty* (see Section 6.3 for details).

This lead also to a distinction in the requirement types. Those that are quantifiable as *StorageCapacity*, *CPUSpeed*, etc, and those who are not quantifiable. Note however that, it might be the case that a particular type could be both quantifiable or not. For example, availability could be a percentage value (quantifiable) or a certain feature of the service: "High availability provided by different hosting location". These cases however were not considered in this prototype for sake of simplicity. It should also be stressed that the *Price* type is considered a quantitative value requirement.

In case of a quantitative value requirement the user has the opportunity to define a minimum or maximum limit value for this requirement, as shown in Figure 6.4. These requirements are usually used for defining computing, storage or network resources (e.g.: CPU Cores, CPU Speed, Storage Capacity, Memory Size, Network Latency, etc).

In Figure 6.5 we are adding a requirement that relates to a qualitative value: *StorageType*. As we can see the questions differ from those of the quantitative value requirements. In this case the user can specify a specific value (a search string) for the requirement. As we wanted our database service to have SSD storage we specify that in the requirement. These are usually used for service features, policies or capabilities (e.g.: Security policies, monitoring capabilities, scaling, performance) rather then resources.

It could also happen that a user wants his service to not have a specific feature. In this case he can set the requirement to negative (see Section 6.6). For instance: "we want our database service to not use SSD storage". All the storage types would be considered except those with SSD.

The final remark regarding requirements goes to the possibility to define a requirement also as a criterion. This means that the type of the requirement will

also be used to create a new criterion. It is simply a mechanism to ease the user interaction. By explicitly saying that the requirement will also be a criterion the user is already creating a new criterion, thus not needing to manualy create it.

Criterion

A criterion is defined in the same way as a requirement and respects the same rules concerning the type and the quantitative or qualitative values. However, its goal is completely different. While a requirement is used to define a blueprint in which cloud services, in order to be considered alternatives, must fit, a criterion is a decision variable. By defining a criterion the user is saying that he wants to capture and evaluate the cloud service attribute concerning the criterion type. Figure 6.4 shows, in the final step, how to insert a new criterion.

Note that a criterion may or may not be related to a requirement. A user might define *StorageCapacity* as a criterion but not defining any specific requirement. This means that the amount of StorageCapacity provided by the service has no limit or special constraint but it will be considered in the decision process. The reverse can also happen. A user might want to specify a limit *StorageCapacity* but is not concerned with the actual value provided by the service, provided that the requirement is fulfilled.

Another important concept concerning criteria is the preference direction. The preference direction is the user way to express is desire about the maximization or minimization of the cloud service attribute captured by the criterion. Price for example is a typical criterion to be minimized, the smaller the price the better. StorageCapacity in the other hand could be maximized, the bigger the capacity value the better. However, there may be situation where a specific value is the optimal solution. We want our database to have 500Gb, bigger is not better and smaller is also not better. This is a preferable value for the criterion that substitutes the preference direction. However, if no preferable value is specified, the preference direction is then used. The preferable value is also a good example of a user preference.

User Preferences

User preferences are pieces of information asked throughout the execution process when needed. Usually they are related to a preference value or a criterion weight value. These pieces of information are collected if necessary by the decision method being applied in the moment and may vary depending on the other information inserted in the beginning. In Figure 6.6 we have an example of the system asking if the criterion *StorageCapacity* has a preferable value, and what it might be. Note that this information is asked during the decision process and not in the initial user data input phase.

6.3 Mappings Between Linked USDL Service Descriptions and the Application Prototype

Linked USDL is a key element in this thesis and as already explained in Section 5.2 a great research effort has been put into the pricing model. The Price is after all, one

```
STATUS: Decision for Service Template: database
Does criterion StorageCapacity have a preferable value? (Y/N)
y
Please insert the preferable value for criterion: StorageCapacity:
500
SYSTEM: Offering Heroku Crane Databases : StorageCapacity=524.0
SYSTEM: Offering Heroku Zilla Databases : StorageCapacity=524.0
SYSTEM: Offering Heroku Kappa Databases : StorageCapacity=524.0
```

Figure 6.6: CloudAid Prototype: User Preferences Example

of the most important factors when enterprises think of the Cloud for their solutions. However, in spite of the price model importance we still need to access the service description and extract all the other service information besides price. Thus, the Linked USDL Core [100] module, in addition to the Pricing, was used for encapsulate the main service concepts such has Service Model, Service Offering, or Provider.

The first challenge was to look into the Linked USDL service description and extract the service relevant information to the potential consumer. Thus, the first step was to understand what could be the interesting part of the description from the consumer point of view. The fact is that a consumer is not concerned with the service and its components, or either if the service is composed of several other services or if it is an atomic service. The concern, most of the times, is with the actual offer. What set of functionalities and resources does the provider offers and what is the compensation (price)? This is usually the question asked by the consumer. Looking at the problem from this perspective we decided that the Service Offering should be the central concept. This is, by the way, what we see in the Service Data Model depicted in Section 4.2.4.2.

The *ServiceOffering* class was also a good link between the linked USDL Core and the Pricing since it is indeed the connection point between the two modules.

Accordingly to the Linked USDL core specification a Service Offering can be composed of several Services, "A service offering is an offering...of one or more services to the public or specific customers." [100], thus, two other Linked USDL concepts are needed: *Service* and *ServiceModel*. The former holds all the service features and resources, the latter as [100] says, "is used to represent 'classes' of services, i.e. services that share a number of characteristics. ServiceModel enables the capturing of these characteristics.", and is in fact a subclass of *Service*. This means that in order to capture the full set of the offered resources and features we need to gather all the *Services* and *ServiceModels* included by the *ServiceOffering* class.

The CloudAid Prototype *Offering* class groups all this information. The process used to capture this data is explained in Section 6.6.3.

An important remark should be made to the Linked USDL usage of the GoodRelations ontology [55] which distinguishes the service features from *QualitativeProductOrServiceProperty* and *QuantitativeProductOrServiceProperty*. The difference is evident, while the former describes properties that can be quantifiable (e.g.: Storage, Speed, any number of things or metrics), the latter describes unquantifiable concepts, usually features or service capabilities (e.g.: Security, Policies, etc). This

distinction is also captured and maintained in the Service Data Model (see Section 4.2.4.2) Thus, instead of having only one list of service features we have one for each type: quantitative and qualitative. The way these properties are extracted is also explained in Section 6.6.3.

It should also be stressed out that the price had a special concern as explained in 5.2. However, further explanation on how to deal with the service price are given in Sections 6.6.2.1 and 6.6.3.

6.4 Controller

As the name suggests, the Controller is responsible to control the execution flow. It corresponds to the controller in the MVC model. Among its tasks we have:

1. Environment Startup
2. Start the correct execution mode
3. Choose between the different Decision Methods
4. Control the application execution flow
5. Establish communication between User Interface module and all the other modules.

These first four tasks are sequential, task 5, however, is performed at the same time as task 3 and 4. Task 1 is only executed once, when the application starts. Task 2, 3 and 4 are executed every time the user wishes to start a new CSA. However, the execution mode cannot be changed during the same execution.

Startup

The first task performed by the Controller is the startup of all the application components. Although they can be used several times in one execution, these modules are only instantiated once, their methods can then be invoked whenever necessary to process the required data and returning the results.

The first module to be initialized is the User Interface, followed by the Search Engine (Section 6.6), the Decision Engine (Section 6.7) and finally the Aggregation Engine (Section 6.8).

While the other modules are only instantiated and nothing else is performed, the Search engine, however, has a special task: initializing the Service Set. The reason for doing this task at this point was to reduce the waiting time for the user. We are initializing the entire service triple store, which might take a while depending of the number of triples. Therefore, there was no point in doing this every time a new search was performed. See Section 6.6.1 for more details about the Service Set.

Execution Modes

After the startup task is complete and before the first module is invoked the Controller checks which execution mode is being executed. The execution mode

```
SYSTEM: Initializing CloudAid Components...
STATUS: ----Initializing ServiceSet
SYSTEM: Loading ServiceSet...
SYSTEM: Reading file: ./Services/Heroku/heroku_v5.ttl
SYSTEM: Reading file: ./Services/Heroku/heroku_vocabulary_v2.ttl
SYSTEM: Total triples in ServiceSet: 760
SYSTEM: Successfully loaded 1 Service Descriptions.
STATUS: ----Initializing DecisionEngine
STATUS: ----Initializing AggregationEngine
Are you confortable giving weight to the criteria? (Y/N)
y
CSA DATA:
1 - New Service Template
2 - New Requirement
3 - New Criterion
0 - DONE!!!
```

Figure 6.7: CloudAid Prototype: Application startup and Decision Method Choice Question

defines the type of interaction with the CloudAid application. Currently in the Prototype two execution modes exist:

- **Automatic mode** uses the simulation data from the *DataSimulator.java* and not the User Interface for inserting the CSA data. This mode is only used for testing purposes.
- **Shell mode** is the default execution mode. It uses the shell based user interface for inserting the CSA data.

Choose Between Decision Methods

As specified in Appendix C.2.4 with the SR2 requirement, the application should be able to use different decision methods depending on the user choice. Choose between the available decision methods is the third task of the controller.

The CloudAid Prototype is currently using two different decision methods: Simple Additive Weighting (SAW) and Analytic Hierarchic Process (AHP). In Section 6.7.4 we explain the differences between the two methods in more detail and how they are being used. However, we can say that the entire application is ruled by which decision method is being used. Thus, it must be decided which to use prior to the application process start.

This decision between the different method is intended to be based on the user preference. However, it may often happen that the user does not know about decision methods and what does imply to use one method over the other. Hence, Instead of asking directly to the user, through the user interface, which decision method he wishes to use, the approach was to ask questions to the user about what he is comfortable doing. Then, based on this questions the Controller can decide which is the best method to be used.

Figure 6.7 shows the question asked to the user for deciding which decision method will be used. Since this prototype uses only two methods and they differ in one specific characteristic there was no need for more questions. However, the inclusion of new methods may require extra questions. The question, "Are you

comfortable giving weight to the criteria?" is enough to distinguish the two available decision methods. If the answer is "Y" (Yes), the SAW will be used. However, if the answer is "N" (No) the AHP will be used instead.

Execution Flow

After tasks 1, 2 and 3 are complete the application can start the execution of the other modules. The Algorithm 1 shows the entire process followed until the aggregated solutions are displayed to the user.

Algorithm 1 CloudAid Prototype Execution Flow

```
method ← DecisionMethodChoice()
data ← EvaluateCSA()
if data passes the evaluation process then
  for all serviceTemplate ∈ data do
    alternatives ← Search(serviceTemplate)
    if alternatives not empty then
      decisionResults ← Decision(serviceTemplate, alternatives, method)
      Add decisionResults to data
    else
      No alternatives found
      Start from the beginning.
    end if
  end for
  result ← Aggregation(data, method)
  Write result
else
  data not ok
  Start from the beginning.
end if
```

This process starts by evaluating the CSA data inserted by the user. The evaluation task is performed by the CSAEvaluator module.

If the data passes all the tests we can start the search mechanism, performed by the SearchEngine Module. This search, however is performed for each individual Service Template, allowing to find a list of alternatives for each Service Template. It may happen that different Service Templates share common alternatives, but this only means that the service offering in question is a valid candidate for both Service Templates.

After finding the alternatives for a particular Service Template the Decision mechanism can start. The first step is to rank the found alternatives accordingly to the user preferences and criteria (performed by the DecisionEngine). The second step is performed, after step 1 has been executed for all the Service Templates, by the AggregationEngine. While in the first step we decide upon the best alternatives for each Service Template, the second step decides which are the best aggregated

solutions.

We could indeed merge these two steps. However, it could prove impossible for the user to understand what was being done. Therefore, reducing the accuracy of the answers to the required questions during this decision process (e.g.: User preferences data, comparison between alternatives, etc...). It also proved easier to model the decision problem. Instead of trying to model a complex multi level decision problem we splitted the problem into two steps: the best alternatives for each Service Template, and the best aggregated solution. Being this second step facilitated by the results obtained in the first.

Module Communication

The final responsibility of the Controller module is to ensure communication between all the modules in the CloudAid prototype.

This task is particular important when some module needs user input to proceed. As we saw in Section 6.2 the user preferences are specified throughout the application execution, assuming special importance in the DecisionEngine and AggregationEngine, where most of these preferences are requested (e.g.: Preferable values). However, all the other modules request the User Interface functionality, even if only for status report or data display.

In order to maintain the MVC model and the modularity and extensibility required for IMPR1 and IMPR2 requirements, it was decided to use the Controller to establish these requests to the User Interface. Every time a module requires user input or wants to display something, it calls the Controller which invokes the proper functionality of the User Interface. Then returns the response, if any exists (in case of input requests) to the requester method. Table 6.1 shows the codes used for identifying the different request types. This approach also enables the ease for future extension, in this case enabling the modification of the User Interface without the need for any modification in the other modules.

Table 6.1: CloudAid Prototype: User Interface Requests Controller Codes

CODE	**Method**
GET_WEIGHT	askforCriterionWeight()
Asks the user the desired criterion weight. Returns the criterion weight value.	
GET_PREFERENCE_DIRECTION	askforCritPrefDirection()
Asks the user the desired preference direction. Returns the preference direction value.	
GET_PREFERENCE_VALUE	askforPreferenceValue()
Asks the user the desired preferable value. Returns the preferable value.	
GET_DISTANCE_VALUE	askforDistance()
Asks the user the desired distance between two concepts. Returns the distance value.	
GET_YESNO_ANSWER	promptYesNo()

Continued on Next Page...

Table 6.1 – Continued

CODE	Method
Ask the user a Yes/No question. Returns the answer result.	
PROMPT	prompt()
Displays some message in the User Interface.	
PRINTCSA	printResults()
Displays the CSA in the User Interface.	
PRINTALTDATA	printAlternativesData()
Displays the alternatives in the User Interface.	
PRINTRESULTLIST	printResultList()
Displays the decision results in the User Interface.	

6.5 CSA Evaluator

The CSA Evaluator has two main responsibilities: check the CSA for any inconsistency or potential error and the inference of new data.

The evaluation process is responsible for checking the overall consistency of the CSA data and all its components. For example, duplicate criteria for the same Service Template, or situations where two requirements block each other (StorageCapacity > 500Gb and StorageCapacity < 400Gb, these will never return any suitable service offerings, since al the requirements are used in the search with *AND* operations). However, in this prototype the only test being done is the number of Service Templates in the CSA. There is no point in executing all the search and decision process when there is no Service Template to analyse.

Thus, since the testing and inconsistency check was not fully implemented in this prototype, the purpose of the CSA Evaluator shifts almost entirely to the inferring of new data. A process called Generalization, because it is exactly that, a generalization of global data.

6.5.1 Generalization process

This process takes place after the evaluation process, and only if the CSA passes all the tests. It analyses the entire CSA data inserted by the user and finds information that can produce other information. For the system to know what can be inferred a group of rules was established which can be tested from the data in the CSA:

- If a Requirement has the flag *criterion* set to *true*, a new Criterion can be created with the same type as the Requirement.
- If a Requirement is defined as global (at the CSA level instead of the ServiceTemplate), a copy of the Requirement can be added in all Service Templates.
- If a Criterion is defined as global, a copy of the Criterion can be added in all Service Templates.

These rules are tested in all the CSA components producing the extra information. Only then the CSA is ready to be sent to the Search Engine. Usually when creating new Criteria out of the flagged Requirements some extra information must be asked to the user (eg: criterion weights or preference directions). These information requests may vary depending on the decision method.

By ensuring that all the global requirements and criteria are generalized to all the Service Templates we make sure that no requirement is left behind when searching for the alternatives that fit the template. Note that these mechanisms (global requirements/criteria and requirement flagged as criteria) are ways to ease the user interaction as explained in Section 6.2 that must be expanded in order to obtain the complete CSA.

6.6 Search Engine

The Search Engine is the second step in the execution flow, after the CSA Evaluator, and as depicted in Algorithm 1 is executed once for each Service Template in the CSA. The Controller invokes the *SeachCore* class, which has three tasks:

1. Distinguish the exclusive requirements from the non-exclusive requirements
2. Start the search process for the exclusive requirements
3. Enrich the resulting alternatives with the service offering attributes

As we can see in task 2 the search is only performed with the exclusive requirements. Therefore, before the search mechanism itself the Service Template requirements must be analysed to gather all these type of requirements.

The reason for searching only for service offerings that fulfills the exclusive requirements is because these are those requirements who will limit the search spectrum. Rather than non-exclusive that only state a wish but not a mandatory constraint, an exclusive requirement eliminates service offerings from the list of potential alternatives. For example, a user defines a requirement for StorageCapacity, however, he does not specify any limit value and states that the requirement will be also a criterion and if it is not present there is no problem. This requirement will be a non-exclusive requirement since the user does not define any limit, but he still want to evaluate this attribute even if there is no value for it. In order for a requirement to be exclusive he must fulfills at least one of the following rules:

- **Not flagged as a Criterion** - If the requirement is not linked to a criterion there is no point in defining a non-exclusive requirement since it will not affect the search or decision processes. By specifying that a requirements is not a criterion the user is saying: "I want this feature/resource, but i don't care about its value and it will not be important for the decision process."
- **Has a maximum value** - If the requirement has a maximum value defined all the service offerings with values that exceed this maximum value will be excluded. The user is saying: "I want to have this resource with a lesser value than X".

- **Has a minimum value** - If the requirement has a minimum value defined all the service offerings with values bellow this minimum value will be excluded. The user is saying: "I want to have this resource with a greater value than X".

- **Flagged as not needed** - If the requirement has been flagged as not needed, it means that even if this resource or feature is not present in the service offering the offering is not excluded. The user is saying: "I would like to have this resource or feature although it is not mandatory".

Note that in order for the search to take place at least one exclusive requirement must exist. The rational was that, in case of no exclusive requirements, the search process will be rather a listing functionality instead of a query search, since no limitations to the search were being introduced.

After this first task of requirements distinction the actual search process can start. To do so, the *SearchCore* class invokes a particular method in the JenaEngine: getOfferingByExcludedReqs(). This method receives the the list of excluded requirements gathered in the previous step and returns the list of alternatives found based on those. Section 6.6.2 will explain in detail this method.

This leaves one final step to be performed, the enrichment of the alternatives with the attribute values. The enrichment is the populating process of an HashMap (*attributes* in the *Offering* class) where each pair (key, value) corresponds to the criterion in the Service Template(key) and the actual service offering value for that attribute (value).

For example, we have a Price criterion and a StorageCapacity criterion. During the search process a service offering SO that fulfills the requirements has the price of €25 and 250Gb of storage. The final alternative would have an HashMap with two pairs (key, value): "$Price$" = 25 and "$StorageCapacity$" = 250.

It should be stressed out that the Price is the only attribute permanently in the *attributes* HashMap. This means that even when no criteria is defined for a particular Service Template, the price is set as one, and in this case the only, attribute. The reason for this is the high importance given to the Price, and the fact that it is the actual compensation for the provision of a certain service offering.

After all these three steps are finished the SearchEngine terminates is job for a particular Service Template, returning the list of alternatives to the Controller which will continue with the execution flow.

6.6.1 Service Set

The Service Set is our triple store. The representation of an Apache Jena Model [1]. The purpose for the creation of a separate class to hold the Service Set is the separation between the knowledge base and the application. By doing so we once

[1] `http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html`

more ensure the extensibility needed if in future developments better versions of the triple store are to be implemented.

For simplicity purposes, and since this is a prototype with no big issues regarding durability or performance, the entire Service Set is stored in memory. The reason was mainly concerned with ease of implementation. However, future developments should consider the use of a persistent triple store. With this being said, the concern of the *ServiceSet* class is to import all the service descriptions and deal with all the operations related to data consistency.

Every time the application starts and the Service Set is initialized the *load*() method is invoked. This method reads all the files in a specific folder (./Services/) in search for any turtle or RDF files. Note that only files inside this folder will be considered for the Service Set. However, subdirectories are also supported.

Every file is then submitted to a small validation process, which checks if we are in presence of a Linked USDL description or any other kind of vocabulary. This validation is merely for debugging purposes since even if the file is not a Linked USDL description it must be imported. This has to do with the impossibility to ensure that concepts used to describe Service Properties are hosted in the same RDF file as the description itself. In fact, all the good practices for modelling a service with Linked USDL point exactly to this differentiation between the Service Description and the service concepts vocabulary. An example the the Linked USDL Service Description tuturial presented in Appendix A.

This import process was essentially performed by adding to the main model (ServiceSet) all the individual models extracted from each file. This however, created a few problems with the namespaces and prefixes.

Since theoretically, each file could have their own set of prefixes and link to other resources in other baseURIs, we had to figure out a way to deal with this cross referencing. The solution was to initialize the model with a set of predefined prefixes, those which were more common such as, *RDF*, *GoodRelations*, *SKOS*, all the linked USDL prefixes, etc. Then, every time a new model was read a name was given to the model (typically based on the file name) and this name was added to the prefix list together with its baseURI. Thus, ensuring that any references to resources described in different baseURIs of the service description were also found within the main model.

6.6.2 Jena Engine

The *JenaEngine* class is the wrapper class for all the operations concerning the Jena library [2]. Jena is an RDF manipulation library with support for many of the standards used in Semantic Web, such as SPARQL, and as explained in Section 4.2.5 was the choice for dealing with our triple store. Thus, being directly responsible with the communication with the Service Set.

Note that every instance of the *JenaEngine* class shares the same Service Set. Since there is only one instance of the SearchCore there is also only one instance of the JenaEngine and the Service Set. In spite of the statement in Section 6.4 that the Controller was responsible to initialize the ServiceSet, that is not completely

true. The Service Set is initialized by the JenaEngine, which by himself is initialized by the SearchEngine. However, since the Controller is the one who initializes the SearchEngine, and since the other initialization are performed in each class constructor, or in the case of the ServiceSet, are Static pieces of code, by transitivity the Controller initializes all the SearchEngine components: SearchCore, JenaEngine and ServiceSet.

The Jena Engine can be divided into three different parts: the query builder, the search itself and finally the search results conversion. As any search engine for the search to be performed a query must be submitted to the database, or in our case the knowledge base (Service Set). In order to query a a triple store a SPARQL query must be submitted. The first phase in te JenaEngine is precisely this SPARQL query creation. The *queryBuilder*() method is responsible for creating the query based on the list of exclusive requirements received from the *SearchCore* class.

Only after the query is ready the search can be performed by submitting the final SPARQL query. The *getOfferingByExcludedReqs*() method takes the query and executes it in the ServiceSet (Apache Jena Model). The result is always a two column table. Being the first column an Apache Jena Resource [2] of the Linked USDL ServiceOffering and the second a Literal [3] with the overall calculated price for that particular offering.

The final step performed by the JenaEngine, before returning the list of Alternatives, is to convert the search results. These results are still in the RDF format, which has no use for the next steps in the application execution flow. This need for a conversion operation lead to the creation of a new class which receives a *ServiceOffering* Resource and converts it into an *Offering* instance with all its properties as explained in Section 4.2.4.2. After the conversion has been performed we have a list of Alternatives (which are instances of the *Offering* class) that can be returned to the *SearchCore* class

It should also be noted that the search mechanism assumes a Closed World, which by the author in [107] is defined as "...certain answers are admitted as a result of failure to find proof. More specifically, if no proof of a positive ground literal exists, then the negation of that literal is assumed true.". What this means is that if a *ServiceOffering* described in the ServiceSet does not have information about a specific feature or resource, the system assumes it does not exist. This was the solution fo some issues regarding the lack of completeness of service descriptions. Too often a provider omits information about the service he is providing, or because he thinks it is not important for the user to know or because he thinks it is too complex for the user to understand. Either way, without assuming a closed world the complexity of the system would escalate to unbearable levels for this prototype.

[2] http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Resource.html

[3] http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Literal.html

6.6.2.1 Queries

In the CloudAid Prototype the SPARQL queries are far from trivial, and encompass a lot of different constraints, all of them extracted from the exclusive requirements received as input in the *getOfferingByExcludedReqs*() method.

All the relevant data from these requirements must be converted in a suitable SPARQL query prior to its submission. Besides, a special concern was dedicated to the Price. From the Linked USDL Pricing model described in Section 5.2 and available in [102], we see that one *ServiceOffering* can have multiple price calculations. Thus, in order to calculate the overall offering price, a series of calculations must be performed. We intended to capture this final price in the final query to be submitted. This way all the offerings would have an overall price associated with it directly from the search result without the need for post-processing.

All this reasons lead to the splitting of the JenaEngine query builder into three parts:

- Quantitative Value Requirement query builder (e.g.: Listing 6.1 line 12 to 37)
- Qualitative Value Requirement query builder (e.g.: Listing 6.1 line 39 to 56)
- Price calculation and retrieval query builder (e.g.: Listing 6.1 line 94 to 125)

Each exclusive requirement is treated by one of the above query builders producing one query. The final result is a merged query with all the individual requirement queries. This means that for each Service Template only one query is produced, composed however, of n sub-queries, being n the number of exclusive requirements. Listing 6.1 shows an example of a resulting query based on the heroku testing environment.

Note however, that in case of the non-existence of any Price Requirement, the price query is still executed. Since we use the Price as a permanent attribute we still need it to be calculated and retrieved in order to be added to the attributes HashMap. In Listing 6.1 line 124 however, we define a maximum value of €4000 for the price, since the user had specified a Price Requirement with this limit value.

The use of the CloudTaxonomy (Appendix F) has an important role in the search mechanism. As we already know, each requirement has a specific type, which corresponds to a particular concept from the taxonomy. These types are the main search parameters since they are the "tags" by which the features will be retrieved. If a requirement has a *StorageCapacity* type, the system knows that all the features or resources linked to the CloudTaxonomy *StorageCapacity* class should be retrieved. Therefore, the CloudAid Search mechanism is based in keyword searching, where the keywords are the cloud concepts defined in the CloudTaxonomy. Hence, the more complete the Taxonomy the more efficient will be the search mechanism. In Listing 6.1 we can see the use of the CloudTaxonomy prefix in line 8. This prefix is then used in every query (except price queries) to check if the Service Offering has this particular feature or resource (Line 17 and 47 are two examples for *StorageCapacity* and *Backup_Recovery* properties). Note that each subquery relates only to one

CloudTaxonomy concept. For instance: in Listing 6.1 the subquery from line 12 to 37 only has the StorageCapacity concept.

It may also happen that a user wants to search for Service Offerings that do not posses a particular feature or resource. In these cases we have a negated search as Listing 6.1 shows for the subquery from line 76 to 92. The difference is line 80 where the *MINUS* operator is used. This means that the user does not want Service Offerings with the *Platform MySQL* (Listing 6.1 lines 83 and 84).

Another important remark is the intersection (logical AND) of all the exclusive requirements. This means that all the requirements must be fulfilled otherwise the ServiceOffering will not be considered an Alternative for the Service Template. In Listing 6.1 lines 38, 57, 75 and 93 show this requirement intersection.

```
 1 PREFIX core: <http://www.linked-usdl.org/ns/usdl-core#>
 2 PREFIX price: <http://www.linked-usdl.org/ns/usdl-price#>
 3 PREFIX pf: <http://jena.hpl.hp.com/ARQ/property#>
 4 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 5 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
 6 PREFIX gr: <http://purl.org/goodrelations/v1#>
 7 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
 8 PREFIX CloudTaxonomy: <http://rdfs.genssiz.org/CloudTaxonomy#>
 9 SELECT REDUCED ?offering ?finalPrice
10 WHERE {
11    {
12       SELECT REDUCED ?offering
13       WHERE {
14          ?offering rdf:type core:ServiceOffering .
15          ?offering core:includes ?a .
16          {
17             ?a gr:quantitativeProductOrServiceProperty CloudTaxonomy:
                   StorageCapacity .
18             ?f gr:hasValue ?value
19             FILTER(?value >= 500.0)
20          }UNION{
21             ?a gr:quantitativeProductOrServiceProperty ?f .
22             ?f rdf:type CloudTaxonomy:StorageCapacity .
23             ?f gr:hasValue ?value
24             FILTER(?value >= 500.0)
25          }UNION{
26             ?a core:hasServiceModel ?model .
27             ?model gr:quantitativeProductOrServiceProperty CloudTaxonomy:
                   StorageCapacity .
28             ?f gr:hasValue ?value
29             FILTER(?value >= 500.0)
30          }UNION{
31             ?a core:hasServiceModel ?model .
32             ?model gr:quantitativeProductOrServiceProperty ?f .
33             ?f rdf:type CloudTaxonomy:StorageCapacity .
34             ?f gr:hasValue ?value
35             FILTER(?value >= 500.0)
36          }
37       }
38    } . {
39       SELECT REDUCED ?offering
40       WHERE {
41          ?offering rdf:type core:ServiceOffering .
42          ?offering core:includes ?a .
43          {
44             ?a gr:qualitativeProductOrServiceProperty CloudTaxonomy:
                   Backup_Recovery .
```

```
45          }UNION{
46             ?a gr:qualitativeProductOrServiceProperty ?f .
47             ?f rdf:type CloudTaxonomy:Backup_Recovery
48          }UNION{
49             ?a core:hasServiceModel ?model .
50             ?model gr:qualitativeProductOrServiceProperty CloudTaxonomy:
                   Backup_Recovery .
51          }UNION{
52             ?a core:hasServiceModel ?model .
53             ?model gr:qualitativeProductOrServiceProperty ?f.
54             ?f rdf:type CloudTaxonomy:Backup_Recovery
55          }
56       }
57    } . {
58       SELECT REDUCED ?offering
59       WHERE {
60          ?offering rdf:type core:ServiceOffering .
61          ?offering core:includes ?a .
62          {
63             ?a gr:qualitativeProductOrServiceProperty ?f .
64             ?f rdf:type CloudTaxonomy:Platform .
65             ?f gr:name ?value
66             FILTER regex(?value, 'PostgreSQL', 'i')
67          }UNION{
68             ?a core:hasServiceModel ?model .
69             ?model gr:qualitativeProductOrServiceProperty ?f.
70             ?f rdf:type CloudTaxonomy:Platform .
71             ?f gr:name ?value
72             FILTER regex(?value, 'PostgreSQL', 'i')
73          }
74       }
75    } . {
76       SELECT REDUCED ?offering
77       WHERE {
78          ?offering rdf:type core:ServiceOffering .
79          ?offering core:includes ?a .
80          MINUS{
81             {
82                ?a gr:qualitativeProductOrServiceProperty ?f .
83                ?f rdf:type CloudTaxonomy:Platform .
84                ?f gr:name ?value FILTER regex(?value, 'MySQL', 'i')
85             }UNION{
86                ?a core:hasServiceModel ?model .
87                ?model gr:qualitativeProductOrServiceProperty ?f.
88                ?f rdf:type CloudTaxonomy:Platform .
89                ?f gr:name ?value FILTER regex(?value, 'MySQL', 'i')
90             }
91          }
92       }
93    } . {
94       SELECT ?offering ?offeringPrice ?deduction ((?offeringPrice-?deduction)as
             ?finalPrice)
95       WHERE{
96          {
97             SELECT ?offering ?offeringPrice (COALESCE(?finalDeductPrice, 0) AS
                   ?deduction)
98             WHERE{
99                {
100                  SELECT ?offering (SUM(?price) AS ?offeringPrice)
101                  WHERE{
102                     ?offering rdf:type core:ServiceOffering .
103                     ?offering price:hasPricePlan ?plan .
104                     ?plan price:hasPriceComponent ?comp .
105                     ?comp rdf:type price:PriceComponent .
```

```
106                     ?comp price:hasPrice ?priceSpec .
107                     ?priceSpec gr:hasCurrencyValue ?price
108                 }
109                 GROUP BY ?offering
110             }OPTIONAL{
111                 SELECT ?offering (SUM(?deductPrice) AS ?finalDeductPrice)
112                 WHERE{
113                     ?offering rdf:type core:ServiceOffering .
114                     ?offering price:hasPricePlan ?plan .
115                     ?plan price:hasPriceComponent ?comp .
116                     ?comp rdf:type price:Deduction .
117                     ?comp price:hasPrice ?priceSpec .
118                     ?priceSpec gr:hasCurrencyValue ?deductPrice
119                 }
120                 GROUP BY ?offering
121             }
122         }
123     }
124     FILTER((?offeringPrice-?deduction) <= 4000.0)
125   }
126 }
127 }
```

Listing 6.1: Heroku Use Case Query Example

6.6.3 Resource Converter

The Resource Converter is an helper class, with the sole purpose to convert RDF resources from the Linked USDL model to Java objects. More specifically to the *Offering* class from the Service Data Model (See Section 4.2.4.2). This is a static class, thus is never initialized but rather used when necessary by other components.

The converter links directly to the Linked USDL model, both the Core and the Pricing modules in order to extract all the relevant data about the *Serviceffering*. It is important to not forget that the ServiceOffering is the central concept for the CloudAid prototype as stated in Section 6.3. In fact the ResourceConverter receives an HashMap with a list of *ServiceOffering* resources and their associated prices. Then, for each of them, it follows the graph of the ontological model to retrieve its data. The retrieved data is:

- Offering name
- Offering description (*rdf:comment* property)
- List of Qualitative properties (*gr:QualitativeProductOrServiceProperty* property)
  - Property value (*gr:name* property)
  - Description (*rdf:comment* property)
- List of Quantitative properties (*gr:QualitativeProductOrServiceProperty* property)
  - Numerical value (*gr:hasValue* property)
  - Description (*rdf:comment* property)

– Unit of measurement (*gr:hasUnitOfMeasurement* property)

It also sets the permanent attribute Price in the *attributes* HashMap by converting the Literal from the input HashMap received. As said the ResourceConverter directly links to the Linked USDL model, thus being very sensitive to any modification in the model. However, this susceptibility to change was the reason that lead to the usage of this class as an external class, allowing its modification without any effect in the application behaviour.

6.7 Decision Engine

Following the Search Engine, the next step is the Decision Engine. Likewise, the Decision Engine is also executed once for every Service Template in the CSA and immediately after the execution of the Search Engine. In other words, only after the Search and Decision processes are finished with a particular Service Template, is that the Controller passes to the next Service Template to analyse.

However, unlike the Search Engine, that only receives as input the current Service Template being processed, the Decision Engine also receives the alternatives as a result of the search process. The purpose of the Decision Engine is therefore to look into the Service Template criteria, to the user preferences, asking for other information if necessary, and rank the alternatives accordingly. Thus, this is probably the module where the majority of the CloudAid calculations are performed. As a result it outputs a ranked list of alternatives sorted from the highest ranked to the lowest ranked alternative.

The ranking of the alternatives is performed by an external method the so called Decision Method. These methods are Multi-Criteria Decision Making methods or simply MCDM methods. Each of them has special requirements about which data is needed and the type of user interaction required. However, independently of the decision method used, the output ought to be the same: a list of the alternatives ranked accordingly to the user preferences and criteria.

The option for this approach was originally due to the wish of allowing different MCDM methods to be used, possibly from the Decision Deck Project [35]. Later on, the initial idea was abandoned. However, there is still a strong desire to allow the usage of different methods for the decision process even if they are not from the Decision Deck Project, as is the case for the two used examples (Section 6.7.4). This desire is otherwise evident in the usage of a standard for modelling and publishing decision problems: XMCDA [38].

Independently of the method to be used several steps must be performed to achieve the final sorted list of ranked alternatives:

1. **Normalize All the Data** - The Service Template data can have values too with different intervals, this would add undesirable noise to the decision process. Thus, we need to process the entire Service Template data and standardize the values (Section 6.7.1).

2. **Express the Decision Problem in XMCDA Format** - After all the values

have been normalized they must be modelled and described with the XMCDA format (Section 6.7.2).

3. **Publish the XMCDA File** - After the decision problem has been expressed in XMCDA, it must be published for the external decision method to use (Section 6.7.3).

4. **Retrieve and Read the Decision Results** - The moment the XMCDA file is published until the decision method executes its decision process the Decision Engine has to wait. When finally the Decision Methods publishes the results they must be once again converted, this time in the opposite direction, from XMCDA to Java Objects.

5. **Sort the Ranked List** - Since the results returned by the decision method might not be properly sorted from the highest to the lowest, the final step is to perform such sorting on the ranked list of alternatives.

6.7.1 Normalization Process

The normalization is performed by the *Normalizer* class. This process is a series of mathematical calculations performed to standardize the alternatives attributes values.

In order to be able to compare multiple criteria we need for the attributes of each alternative to be in the same scale. Otherwise, a value in an wider interval could compromise the entire decision process.

For example we are deciding upon a Service Template with two criteria: *Price* and *StorageCapacity*. Two alternatives are found in the Search Engine, A_1 with $[price = 5, StorageCapcacity = 1024]$ and A_2 with $[price = 0.5, StorageCapcacity = 750]$. Also, the decision weights for the criteria are $price = 4$ and $StorageCapcity = 1$. In this example we can see that even with a lower decision weight the *StorageCapacity* criterion will make the *Price* irrelevant to the calculations since the *StorageCapacity* order of magnitude is much bigger than the one used for *Price*. The normalization process makes these different scales disappear by standardize the attribute values to the interval [0,1]. Other problem with this example is the *Price* itself. We usually want to minimize the price (preference direction), but in order for that to happen the result must be inverted, so the alternative A_2 would have a better performance in *Price* criterion than alternative A_1.

Therefore, the *Normalization* class is not only responsible for the standardization of the alternatives attribute values but also the application of the preference direction (minimize/maximize) to its normalized value.

However, it is usually the case where the criterion being normalized is not numerical. This happens for example when Qualitative Types are being evaluated instead of Quantitative Types. In such situations, after identifying the type of the attributes being evaluated, the *Normalization* class may need a series of questions to be answer in order to asses the difference between values.

For example, the Criterion *Performance* is being evaluated, and the values of the alternatives are A_1 with $[performance = "NetworkLow"]$, A_2 with $[performance =$

"$NetworkHigh$"] and A_3 with [$performance =$ "$NetworkMedium$"]. In this case a distinction between the different values must be performed. After asking which is the preferable value, which in this cases the user must define, a series of questions is performed about the distance between each alternative attribute value and the preferable value. Note that equal values are only asked once. These types of attributes are called *non-numerical*.

It may also happen that a certain attribute is *non-numerical* but has only two possible options. This is in fact the first question asked when a *non-numerical* attribute is found. If the answer is "yes, there are only two options" the attribute is treated as a *binary* attribute, where one value is the best and the other the worst. If not, the attribute is treated as a normal *non-numerical* attribute as explained above.

Therefore, there are three different types of attributes:

- Numerical
- Non-Numerical Binary
- Non-Numerical

Another important task is to deal with preferable values. As explained in Section 6.2, sometimes the application has the need to ask the user extra information in order to enhance the results. This normalization process is the case of such a situation. Every time a new criterion is being evaluated the application asks if the user has a special value that would be preferable to the simple maximization/minimization process. In case of positive answer, this new value will be used instead of the preference direction. In such cases the calculation changes slightly since, instead of using the highest or lowest value from all the alternatives as the default preferable value, it is used the inserted preferable value. Then, the distance between each attribute value and the preferable value is calculated. The preferable value will be assigned the highest value and all the others will have lower values according to their distance to the preferable value. It may also happen than no alternative has an attribute value equal to the preferable value. In these cases nothing changes in the calculation process, the only difference is the non-existence of the highest possible value.

It should be stressed out that all the extra data needed for the normalization process is requested through the Controller as explained in Section 6.4.

The purpose, once again, is to the alternatives attributes to be standardized in an interval from 0 to 1. Where 1 is the best value and 0 the worst for the Criterion type to which the attribute relates. The entire process to achieve the desired results is:

1. **Check which type of attribute is being processed** - Check wether we are analysing a numerical, non-numerical or non-numerical binary attribute.

2. **Ask if there is a preferable value to the attribute** - It may happen that the user has a specific value that he considers the preferable value for that attribute. These values should be asked before the normalization in order to use them in the calculations ahead.

3. **Normalize the ttribute according to its type** - Once all the data has been collected the attribute can be normalized calcuating the distance to the preferable value and then standardizing to the interval [0,1].

4. **Return the results** - When the process is finished the normalized values are stored in a new hasMap identical to the attributes. The difference is that this version has the normalized values instead of the original attribute values. The original, however, are still necessary, essentially for displaying purposes.

Note also that this process is repeated for each Criterion in the Service Template.

6.7.2 XMCDA Standard

The XMCDA is a XML like format developed by the Decision Deck [35] for describing decision problems to be solved using a MCDM method. Therefore, this format is a data structure to wrap all the information about the decision problem. The Decision Deck provides a Java library, J-XMCDA [37], to deal with XMCDA transformations. This Library was the tool used in this prototype to import and export the XMCDA files.

Depending on the Decision method used, the XMCDA file will have different data. However, all the operations performed with this Library have been wrapped by the *XMCDAConverter* class. This class has not only the exporting methods but also the importing ones. This means that both read and write operations can be done through this class. Once again the idea was to enable the extensibility. So, if by any case, in future developments we see the need for changing any process concerning the XMCDA operations there is only one place where this changes might happen.

A list of the methods in the *XMCDAConverter* class and their descriptions is in Table 6.2. It is also presented the XMCDA tags produced by each method.

Table 6.2: CloudAid Prototype: XMCDA Methods and the Tags Used

Method	**XMCDA Tags**
getFromFile(FileName)	
Read the file with FileName, and converts it to a XMCDA Object for processing.	
createAlternatives (Alternatives)	
Creates the alternative list in the XMCDA Object.	
createAlternativeValues (Alternatives, MethodID)	alternatives, alternative

Continued on Next Page...

Table 6.2 XMCDA Methods and the Tags Used – Continued

Method	XMCDA Tags
Assigns the alternatives attributes values to each alternative in the XMCDA Object. Depending on the Decision Method the values assigned are different.	
createCriteria(Service Template)	criteria, criterion, scale, qualitative,rankedLabel, rank, label
Creates the criteria list in the XMCDA Object. Each Criterion is assigned the preference direction and the preferable value	
createWeights(Service Template)	criteriaValues, criterionValue, criterionID, value, real
Assigns the Criteria Decision Weights to the Criteria in the XMCDA Object.	
attachCompTimestamp (Time, Service Template)	methodParameters , parameter , value, label
This method serves only to write in the XMCDA Object which Service Template is being decided upon and a timestamp for differentiation.	
getMethodParameters(XMCDA)	
Extracts from the XMCDA Object the Timestamp and Service Tempalte being analysed.	
getPerformance (XMCDA, Alternatives)	performanceTable, alternativePerformances, alternativeID, performance, criterionID, value, real
Extracts the decision results (performances) from the XMCDA Object. Returns the list of *Results* which is a list of alternatives with their assigned performance.	
append(XMCDA list)	
Appends a list of XMCDA objects into one single Object.	
export(XMCDA, DestinationFileName)	
Exports the XMCDA data to the *DestinationFileName* file in the FileSystem.	

6.7.3 External Methods Communication

Once the decision itself is performed by an external method or application, the communication between the CloudAid Prototype and those third party applications is an important issue. Even more so when this communication link must be able to send the XMCDA file with the decision problem data.

Therefore, the approach was to publish this file in a specific folder and wait for the Decision Method to do the same. Note that we had the advantage of developing part of the decision methods used, thus allowing the modification of the external application to cope with this approach. However, that might not be the case in

future application. Hence, it is advisable to modify this approach if the need arises.

However, since in this Prototype we had no such concerned beyond the proof of concept, this approach was more than suitable.

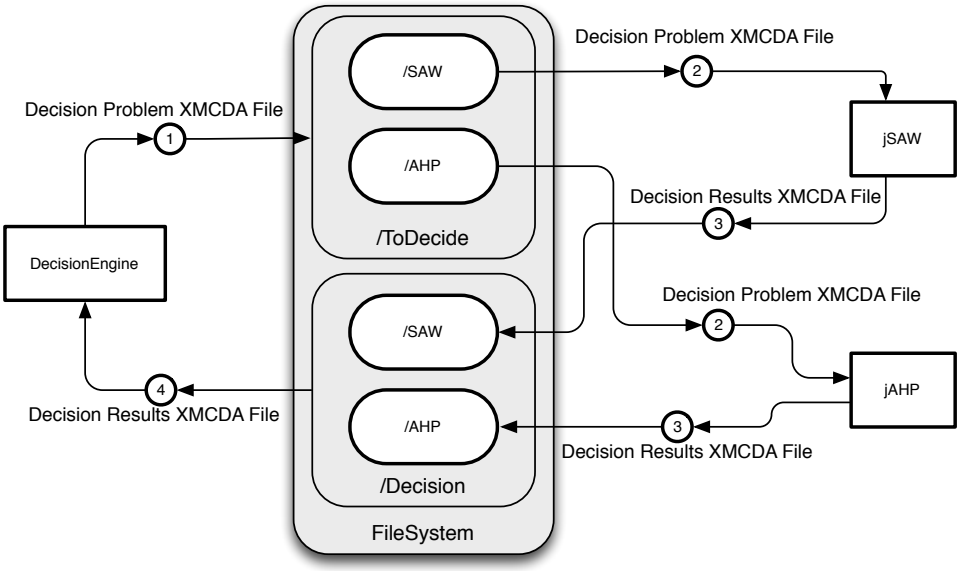


Figure 6.8: CloudAid Prototype: Communication between CloudAid and External Decision Methods

In Figure 6.8 we can see the communication process. The Decision Engine, after creating the XMCDA file with the decision problem data, publishes it in a specific folder intended for the Decision Method to be used (./ToDecide/$< DecisionMethod >$). Then it waits until the external method returns the decision results. These results are also published in the same way, in a specific folder for the particular Decision Method (./Decision/ $< DecisionMethod >$).

In order to handle this process it was necessary to develop a wrapper class (*FileChecker* class) to handle the events in the FileSystem. What this class does is waiting for events performed in the specific directory and awake the right method to deal with the event whenever it happens.

6.7.4 Decision Methods

As previously stated in this thesis, one of the objectives was to allow the CloudAid Application to use different Decision Method depending on the user preferences or comfort about the definition of some data. Therefore, for the purpose of this Prototype two different Decision Methods were used: Simple Additive Weighting (SAW) and Analytical Hierarchic Process (AHP).

While the first is a simple mathematical process, the second is rather more complex allowing different decision capabilities. The idea was not only to prove that the application could support different methods but also show the differences they

imply.

Another concern was the rapid development of these methods allowing for rapid prototyping and results discussion as explained in Section 6.1. This lead to the use of some already existent application that implement a particular method.

6.7.4.1 SAW

The Simple Additive Weighting is the simplest of the MCDM methods and is based on a series of multiplications. The idea is to multiply each attribute of each alternative by its Criterion Decision Weight value. Then all the attributes of an alternative are added and the final alternative performance is obtained.

Being a simple method allowed for a quick development instead of applying some third party library, skipping its learning curve. This also allowed the CloudAid Application to use a full Decision Engine in its early stages.

The JSAW was the resulting application that reads the XMCDA file and makes all the calculations necessary to perform the SAW method. After the decision results have been obtained the application creates a new XMCDA file and publishes it for the Decision Engine to use.

Note that the *XMCDAConverter* class used in the Decision Engine is also used in this external application, only with smaller modification to cope with the data types.

An important requirement of the SAW method however, is the need for the Criteria Decision Weights to be defined somehow, which most of the cases is not an easy task to do, either because they are not easy to assign or because the user does not have a clue about the importance to assign to each of them. Still, as explained, they are a key element for the calculations. Therefore, in case of using this method the user needs to comfortable in assigning these Decision Weights to each Criterion. Besides this special constraint the other required data is the normalized attribute values for each alternative.

6.7.4.2 AHP

The Analytic Hierarchic Process is a more complex method that the SAW. It makes use of a structured way to describe decision problems by dividing the problem in sub-problems. Although, we do not make full use of this capability since all the Criteria is considered to be at the same level. This multi-level could be applied if we had merged the aggregation decision also in this step. In that case we would have a multi-level decision, where the top level would be the individual Service Templates and second the alternatives for each of the Service Templates. However, since all the information would be displayed in front of them at the same time, this approach would prove too complex for users to actually take advantages of.

The process follows a series of comparisons between the various alternatives. Each Criterion can also be compared, allowing for the indirect definition of Decision Weights. Basically the user is answering a series of question of the type: "I prefer Alternative A_1 over Alternative A_2" or "For me it is more important Criterion C_1" than Criterion $C2$.

This type of decision is therefore more suitable to users less comfortable with the decision information, or with less certainty of the domain they are deciding upon.

The implementation of this method is an example of the usage of external applications. It was used the project in [87] to apply the AHP method. The application has a graphical user interface, which greatly leverages the comprehension of the decision method being used. In fact this was one of the key points for choosing this particular implementation. However, the application was slightly modified in order to cope with the XMCDA needs. The support for the import/export of XMCDA was once again implemented with a version of the *XMCDAConverter* class which had to be modified to cope with data types already present in the original application. Another modification was the capability to display the alternative attribute data at the same time of the decision process.

An example of the JAHP application is showed in Figure 6.9. The Figure shows the sliders for changing alternatives performance regarding the *StorageCapacity* Criterion. By sliding up or down we are saying for example that Alternative A_1 is better regarding *StorageCapcity* then Alternative A_2. Note that the user should compare all the alternatives in all the criteria. Figure 6.10 shows the comparison of Criteria, in this example *Price* and *StorageCapacity*. In the example we are saying that *Price* is way more important than *StoraceCapcity*.

After all these comparison are finished the application returns the calculated results (Figure 6.11) to the Decision Engine by publishing the XMCDA file.

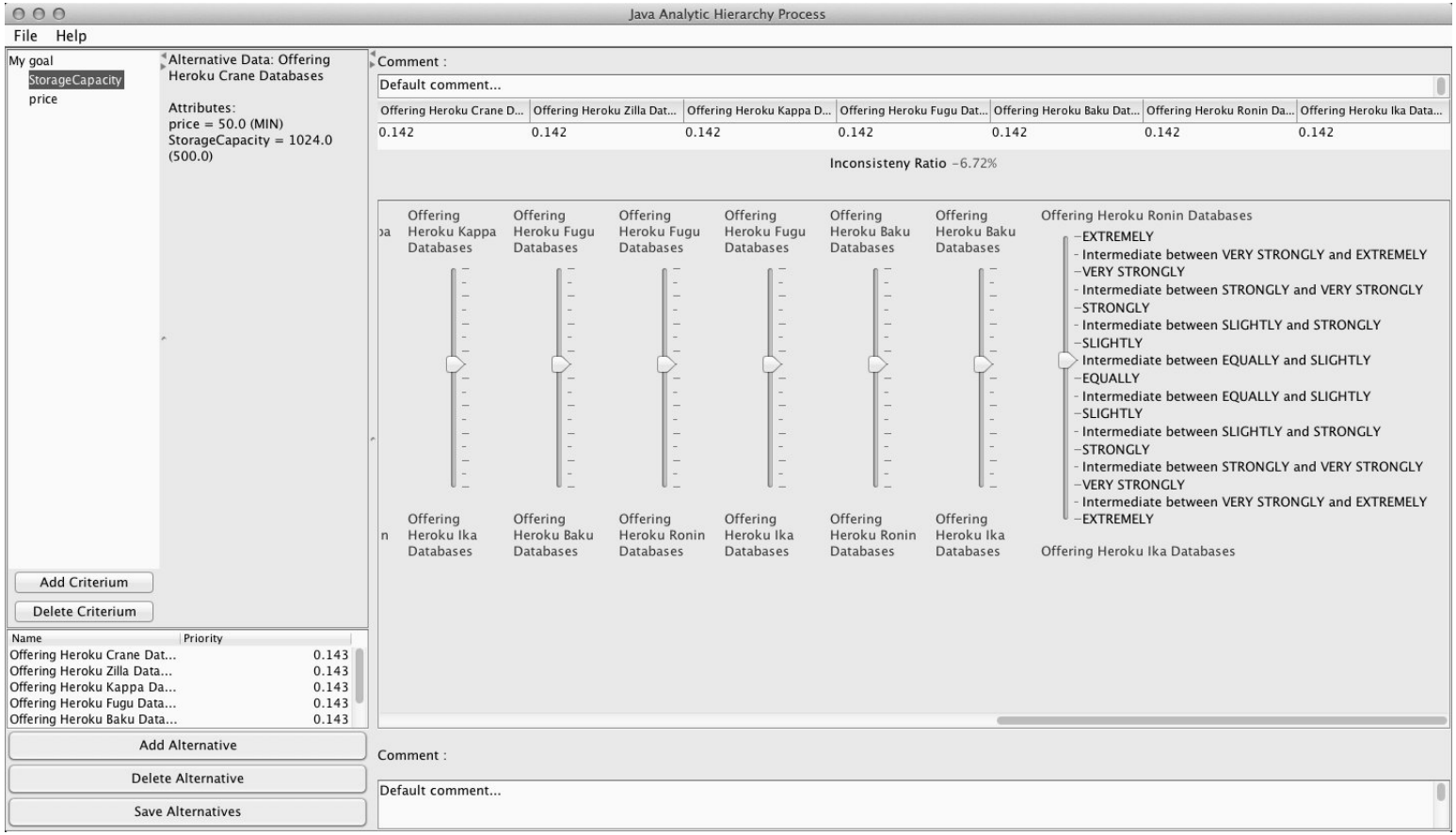


Figure 6.9: JAHP: Example of AHP Decision Process

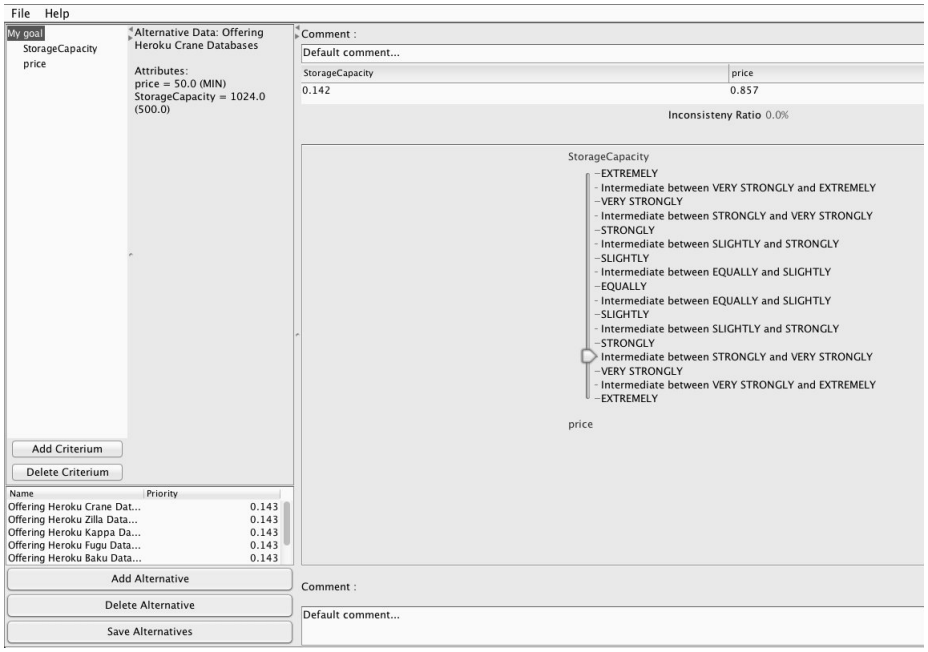


Figure 6.10: JAHP: Example of AHP Criterion Comparison

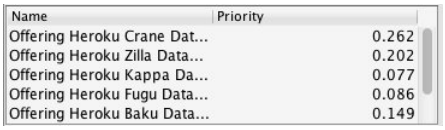


Figure 6.11: JAHP: Example of AHP Alternative Performances

6.7.4.3 Application Execution Differences

Throughout the CloudAid Application the distinction between the two above described methods is evident.

The most significative difference is about the data they need. While the SAW requires an exact definition of all the Criteria and Service Template Decision Weights the AHP does not. This difference however, influences the execution of the application. Table 6.3 shows the list of major changes. Among them is the difference in the methods used for the creation of the XMCDA file. This has to do exactly with the data needed by the two methods. In case of the saw the XMCDA file must carry the Criteria Decision Weights defined by the user. We can also see that both the Controller and the Search Engine do not suffer any change in its execution. However, the Controller has the responsibility of invoking the User Interface for deciding which method will be used and then communicate this information to the modules that need it.

Table 6.3: CloudAid Prototype: Decision Methods Differences in Application Execution

Module	SAW	AHP
User Interface	Requires the definition of Criteria Decision Weights and Service Tempalte Decision Weights plus the default data	Uses the default data (Criteria, Requirements, Service Templates) Criterion
Controller	Default Execution	Default Execution
Search Engine	Default Execution	Default Execution
Decision Engine	Uses the full *Normalizer* class	Only uses the *addPreferences*() method in the *Normalizer* class
	Uses the createAlternatives(), createCriteria(), createWeights() and createAlternativeValues from the *XMCDAConverter* class	Uses the createAlternatives(), createCriteria() and createAlternativeValues from the *XMCDAConverter* class
Aggregation Engine	Uses the default process. No need to call the SAW external method for calculating final Admissible Solutions performance	Uses the *XMCDAConverter* class again and the JAHP application for defining the Service Template Decision Weights

6.8 Aggregation Engine

The CloudAid application is a prototype that proposes an Aggregator of Cloud Services, recommending possible Aggregated Solutions to the user requirements. Therefore, the final step in the CloudAid Execution flow is the Aggregation Engine which unlike the previous two modules (Search and Decision Engines) is executed only once as shown in Algorithm 1. It's purpose is to aggregate all the information collected so far, specially from the Decision Engine and present the best Aggregated Solutions to the user.

Since this module aggregates information and processes it, it receives the entire CSA already enriched by the previous modules. The Search Engine finds the alternatives and joins them with their related Service Template. The Decision Engine takes these alternatives and creates a list of sorted ranked alternatives also assigning them to their related Service Template. The Aggregation Engine will then take all these sorted ranked lists of alternatives, one for each Service Template, and test their admisibility when aggregated. Finally, after deciding which are the best options, it outputs the results to the user.

Note that, each Aggregated Solution must have one alternative from each Sorted

Ranked List. This is the same as saying that each Service Template must have one alterantive present in the Aggregated Solution. So, if, for example, we have a CSA with four Service Templates we would have four Sorted Ranked Lists of alterantives, one for each Service Template, returned by the Decision Engine. This would mean an Aggregated Solution composed always of 4 alternatives, one from each Sorted Ranked List.

In order to achieve these final list of solutions the Aggregation Engine uses one of two specific algorithms developed in this thesis (Section 6.8.1). The *Combinations* class is responsible for the execution of the right algorithm. This class holds all the necessary logic regarding the Admissible Solution Algorithms.

There was no initial plan for the development of more than one algorithm. However, after the first Aggregation Engine prototype it was decided that the first version could be enhanced to support extra functionality. Thus, a second version was developed to support incomparability between Aggregated Solutions. This topic will be further discussed in Section 6.8.1

Once the Admissible solutions have been found, we have a smaller group of solution, all of them viable for the user requirements. However, we can still check what are the best options from this reduced set by using extra information from the user. This process once more depends on the Decision Method being used.

In case of the AHP, there is the need for asking this extra information, the Service Templates Decision Weights. These values are the user preferences about the CSA Service Templates. In other words, they represent the importance of the particular Service Template to the overal system. The higher the value the higher the importance. The Aggregation Engine needs these values to decide which are the best options from the results returned by the Admissible Solutions Algorithm. Since when using the AHP there is no previous information about these values a new request to the external Decision Method must be performed. This means that the JAHP must be called again, which requires the creation of a new XMCDA file. However, this time the alternatives will not be the found service offerings but rather the Service Template names themselfs. This change is only a conceptual change since the file is created exactly in the same way. The decision results returned by the JAHP now hold the Service Template Decision Weight values, which can be used for calculating the Best Admissible Aggregated Solutions.

On the other hand, if using the SAW, there is no need for asking the user for more information, since the user has already inserted the Service Template Decision Weights in the CSA definition phase, together with all the other Service Template, Criteria and Requirements data.

This final process of deciding which are the best Admissible Aggregated Solutions is another application of the Simple Additive Weighting. However, this time the calculations are performed by the Aggregation Engine rather than by the JSAW. The reason for such an approach was simplicity. Since, no matter which Decision Method is being used this step will always be performed in the same way, there was no need for using the external application.

6.8.1 Admissible Solutions Algorithms

As previously stated, an Aggregated Solution is composed of one and only one alternative from each Sorted Ranked Alternatives List. This means that if we have a CSA with ten Service Templates each with ten alternatives we would have a total of 10^{10} possible combinations. Each of these combinations needs to be tested for its admissibility and compared with the other to see if it is better or not. Although the admissibility test might not be too heavy from the computing point of view, the comparison between each combination demands a lot more. It would be impossible to achieve such a solution using a simple brute force algorithm.

Therefore, the Admissible Solutions Algorithm and its two versions were developed with the purpose of finding the Admissible Aggregated Solution from the Sorted Ranked Alternatives Lists in a more suitable way from the computation point of view.

The first question to be asked was if we really needed to compare each and every combination. In fact, we already know that the Alternatives lists are sorted, from the highest to the lowest. This means that being SRA, SRB, SRC the Ranked Sorted Lists for Service templates A, B, C and SRA_i, SRB_j, SRC_k $i, j, k \in \mathbb{N}$ alternatives of the SRA, SRB, SRC Ranked Sorted Lists, then $SRA_i > SRA_{i+1}$ is always true. Applying this knowledge to the entire combination we have that: being $[SRA, SRB, SRC,]$ the set of all possible Aggregated Solutions, we know that $[SRA_i, SRB_j, SRC_k,]$ dominates $[SRA_{i+l}, SRB_{j+m}, SRC_{k+n},], i, j, k \in \mathbb{N}$ therefore, there is no need for testing the set of combination $[SRA_o, SRB_p, SRC_q,]$ if $o > i \cap p > j \cap q > k$. However, we cannot assume dominance of $[SRA_i, SRB_j, SRC_k,]$ over $[SRA_o, SRB_p, SRC_q,]$ if $o < i \cup p < j \cup q < k$.

With this knowledge and applying the notion of combinatory tree to generate all possible combinations of a set of lists, it was possible to use the Breadth-First Search [4] algorithm to traverse the tree and applying a method similar to the one used by the Branch and Bound [5] to reduce the effort for calculating the Admissible Aggregated Solutions. Now every time an Admissible Aggregated Solution is found we only have to test it with the already found Admissible Aggregated Solutions if no dominance is found between them we can add the solution to the list and discard its children. If however, dominance is found between the node being tested and other Admissible Aggregated Solution we can discard this node and all its children. Algorithm 2 shows this first version of the algorithm which does not support incomparability.

In Algorithm 2 when a new Admissible Aggregated Solution is found we still need to compare it with all the Admissible Aggregated Solutions already found. This is the only way to ensure that we are not adding an Admissible Aggregated Solution that is already dominated by another already found. If it is dominated it means that its overall performance value will be lower than the dominant solution, thus, there is no need for storing it.

The second version of the algorithm adds support for incomparability. This

[4] https://en.wikipedia.org/wiki/Breadth-first_search
[5] http://en.wikipedia.org/wiki/Branch_and_bound

Algorithm 2 Admissible Solution Algorithm: Without Incomparability

```
Create queue Q
Add root to queue Q
while Q is not empty do
  node ← Q.pop()
  Add node to tested
  if node is Admissible then
    if node is not dominated by some admissible then
      Add node to admissible
    end if
  else
    for all possible children c of node do
      if (c ∉ Q) ∧ (c ∉ tested) ) then
        if c is not dominated by some admissible then
          Add c to Q
        end if
      end if
    end for
  end if
end while
```

means that we are no longer assuming that the Sorted Ranked List is a total order, but rather a partial order. By admitting this new paradigm we might be able to say that $SRA_i > SRA_{i+1}$, however it may happen that $SRA_j \sim SRA_{j+1}$. In this cases we cannot assume that when an Admissible Aggregated Solution is found all their children in the tree can be ignored. We must check for incomparability between the father and its children and only in case of dominance is that we can ignore those dominated solutions. But if no father dominance is ensured the new child must be compared with all the already found Admissible Aggregated Solutions. If any dominance is found the child can be discarded, if not the child must be added to the Admissible Aggregates Solutions and the process continues for its children. Algorithm 3 shows the new algorithm version with support for incomparability.

6.8.1.1 Check Admissibility

The process of checking admissibility is a key element in finding all the Admissible Aggregated Solutions. An Admissible Aggregated Solutions is an Aggregated solution that successfully fulfills all the global requirements defined in the CSA. Therefore a series of tests must be performed in order to ensure that the combination of alternatives being testes can in fact be aggregated and considered and Admissible Aggregated Solution. All these admissibility testes are performed by the *AggChecker* class.

There are many possible constraints that can forbid the aggregation of several alternatives. Either because some alternative used is not compatible with another, or because together they surpass a certain user defined constraint. Imagine that we

Algorithm 3 Admissible Solution Algorithm: With Incomparability

```
Create queue Q
Add root to queue Q
while Q is not empty do
  node ← Q.pop()
  Add node to tested
  if node is Admissible then
    if node is incomparable with all admissible then
      Add node to admissible
      for all possible children c of node do
        if (c ∉ Q) ∧ (c ∉ tested) ) then
          if c is incomparable with node then
            Add c to Q
          end if
        end if
      end for
    end if
  else
    for all possible children c of node do
      if (c ∉ Q) ∧ (c ∉ tested) ) then
        if c is not dominated by some admissible then
          Add c to Q
        end if
      end if
    end for
  end if
end while
```

have two alternatives for different Service Templates, one being an Oracle Database and another a SaaS for data analysis. If by any case the SaaS alternative does not support Oracle databases we cannot aggregate these two alternatives. Therefore we have and Aggregated Solution but not an Admissible Aggregated Solution.

Many other constraints for admissibility may exist, however, in this prototype it was only implemented a simpler version of this admissibility check system. The only test being performed is the global price.

When the user defines a Global Price Requirement he is saying that the Aggregated Solution must not exceed a certain value. Therefore, in order for an Aggregated Solution to be admissible all its alternatives prices when added must not exceed the fixed maximum value. Note that only maximum values are being considered for this prototype.

The reason for limiting these tests was essentially for implementation simplicity. It is the intention to prove the concept of the admissibility test and how it should be applied in the overall application but it was not in the set of requirements the implementation of a full admissibility testing solution. However, by using another class for this purpose, the *AggChecker* class, we ensure the extensibility and modifiability if further developments arise in this topic.

7 Test Results

This chapter presents the results obtained through the application testing phase as well as the conclusions.

For time constraint reasons it was decided not to formally describe all the tests presented in Section 4.3. Nonetheless, from those not formally described, most were merged with other tests instead of individually tested, allowing to still prove their correct implementation. Note also that due to the development methodology used (RAD), all the functionalities were target of informal individual preliminary tests (Unit Testing) alongside the development phase. This allowed, not only to have a certain degree of confidence about the small prototypes being developed but making the entire development/testing process much more agile as well.

Note that all the tests discussed in this Chapter and their related data can be consulted in the CloudAid Public Git Repository [7].

We start by presenting the list of functional requirements implementation status and the functional tests performed in Section 7.1. Section 7.2 presents the approach, results and conclusions of the reliability tests performed. The specific tests performed to the Search Engine, including performance testing are discussed in Section 7.3. Section 7.4 discusses the Decision Methods and what should be made to ensure the use of multiple support for different Decision Methods from those used. All the tests performed to the Admissible Solutions Algorithms and its conclusions are discussed in Section 7.5. The last tests, integration tests, are presented and discussed in Section 7.6. Finally Section 7.7 draws the final conclusions to the overall prototype testing.

7.1 Functional Tests

The functional tests aimed at testing the application key features. Those that were fundamental to achieve a complete prototype. By complete it is implied the capability to prove the concepts proposed in this thesis. In order to achieve this objective, it should be noted the final status of each functional requirement defined in Appendix C.1. Table 7.1 shows the full list of functional requirements and their status. The two used status are:

- **DONE** - The requirement has successfully been implemented and tested.

- **NOT DONE** - The requirement was not implemented.

As we can see all the "Must" and "Should" requirements were implemented. This allowed to have a fully functional prototype that at least does what was proposed. Only the two "Could" requirements were not implemented: FR8 and FR15.2. It should be stressed out that all the "Won't" requirements were not considered in Table 7.1 list as they were ruled out from the beginning and were moved to the future development tasks.

Table 7.1: CloudAid: Functional Requirements List Implementation Status

ID	Name	Priority	Status
CSA Data			
FR1	Add a Service Template to the CSA	**MUST**	**DONE**
FR1.1	Add a Requirement to a Service Template	**MUST**	**DONE**
FR1.2	Add a Criterion to a Service Template	**MUST**	**DONE**
FR2	Add a Requirement to the CSA	**MUST**	**DONE**
FR3	Add a Criterion to the CSA	**MUST**	**DONE**
Search Engine			
FR5	Search for services that fulfill the Service Template Requirement list	**MUST**	**DONE**
FR5.1	Search for services that do not have a certain functionality	**SHOULD**	**DONE**
FR5.2	Search for services with a minimum value requirement	**SHOULD**	**DONE**
FR5.3	Search for services with a maximum value requirement	**SHOULD**	**DONE**
FR5.4	Search for services with a specific value requirement	**SHOULD**	**DONE**
FR6	Search for services that fulfill price requirements	**MUST**	**DONE**
FR7	Get the service values for the defined Criteria	**MUST**	**DONE**
FR8	Allow the usage of different units of measurements for the requirements	**COULD**	**NOT DONE**
FR9	Read Linked USDL service descriptions	**MUST**	**DONE**
FR9.1	Extract the Service Qualitative & Quantitative Values	**MUST**	**DONE**
FR9.2	Calculate the service price when defined with the price:hasPrice property	**MUST**	**DONE**
Decision Engine			
FR11	Import/Export XMCDA decision data	**MUST**	**DONE**
FR12	Rank the services according to their decision value	**MUST**	**DONE**
FR13	Normalization of Decision characteristics	**MUST**	**DONE**

Continued on Next Page...

Table 7.1 CloudAid: Functional Requirements List – Continued

ID	Name	Priority	Status
FR13.1	Normalization of Decision Numerical characteristics	**MUST**	**DONE**
FR13.2	Normalization of Decision Non-Numerical characteristics	**MUST**	**DONE**
FR13.3	Normalization of Decision Binary characteristics	**MUST**	**DONE**
FR13.4	Normalization based on User preferences	**SHOULD**	**DONE**
Aggregation Engine			
FR14	Generate Aggregated alternatives	**MUST**	**DONE**
FR15	Calculate the admissible aggregated solutions	**MUST**	**DONE**
FR15.1	Calculate the admissible aggregated solutions based on price requirements	**MUST**	**DONE**
FR15.2	Calculate the admissible aggregated solutions based on service restrictions	**COULD**	**NOT DONE**
FR16	Decide which is the best admissible solution from the admissible solutions list	**MUST**	**DONE**

In order to asses the correct implementation of the list of requirements in Table 7.1 a series of tests were performed. In fact beyond the simple assessment of the correct implementation these tests allowed to find possible problems in the prototype execution, which in some cases existed. However, by identifying them it was possible to fix or, if it was a conceptual problem rather than an implementation problem, to change the original prototype concept. Thus, some of these tests were not simply pass or no pass tests but also a tool to validate the overall prototype concept.

In this Section we present two examples of Test Cases performed. In Figure 7.1 the insertion of a new Service Template is being tested. The test was successfully executed, passing all the steps. Moreover, this test can be generalized for all the FR1 to FR3 requirements since its execution is partially the same, only changing the data inserted.

Figure 7.2 on the other hand is a good example of the test merging approach followed, where several test cases were grouped in only one. It is being tested the Search Engine process and its capability to search for all requirement types. Therefore, by creating a set of requirements that include all these types we could, in one test, prove the correct implementation of the search mechanism. The requirement types being tested are:

- Quantitative Type Requirement with a minimum value (*StoragaCapacity*)
- Qualitative Type Requirement (*Backup_recovery*)
- Qualitative Type Requirement with a specific value (*Platform = "PostgreSQL"*)
- Qualitative Type Requirement with a negated feature (*Platform = "MySQL"*)

- Price Requirement which is at the same time a Requirement with minimum value.

Therefore, with this Test Case we can prove Requirements FR5 to FR6. Most of the other functional requirements such as FR7, FR9, FR11, FR12 and FR15 are tested, in a similar merging approach, along with the Reliability tests depicted in Section 7.2.

Project Name: CloudAid			
Test case ID:	TFR1	**Test Designed By;**	Jorge Araújo
Module Name:	UI	**Test Designed Date:**	27/05/2013
Test Title:	Add a ServiceTemplate to the CSA	**Test Executed By:**	Jorge Araújo
Test Description:	Test if the ServiceTemplate data is added to the CSA	**Test Executed Date:**	27/05/2013

Pre-Conditions:	
Dependencies:	It depends on the Controller module
Post-Consitions:	A new ServiceTemplate must be in the CSA Data

Step	Test Step	Test Data	Expected Results	Actual Results	Status (Pass/Fail)	Notes
1	Answer the first question about weights insertion.	"y"	User interface CSA Data menu screen	User interface CSA Data menu screen	Pass	
2	Choose option 1 in the menu		User interface Service Template Data menu screen	User interface Service Template Data menu screen	Pass	
3	Choose option 1 in the menu		Ask for the ServiceTemplate type	Ask for the ServiceTemplate type	Pass	
4	Insert type data	"database"	Ask for the ServiceTemplate description	Ask for the ServiceTemplate description	Pass	
5	Insert description data	"this is a database service template"	Ask for the ServiceTemplate weight for a decision	Ask for the ServiceTemplate weight for a decision	Pass	
6	Insert weight data	"2.3"	User interface Service Template Data menu screen	User interface Service Template Data menu screen	Pass	
7	Choose option 0 in the menu		User interface CSA Data menu screen	User interface CSA Data menu screen	Pass	
8	Choose option 0 in the menu		CSA SERVICE TEMPLATEs: ServiceTemplate [id=Comp0, type=database, description=this is a database service template, requirements=[], resultList=null, criteria=[], weight=2.3] CSA GENERAL REQUIREMENTS: CSA CRITERIA:	CSA SERVICE TEMPLATEs: ServiceTemplate [id=Comp0, type=database, description=this is a database service template, requirements=[], resultList=null, criteria=[], weight=2.3] CSA GENERAL REQUIREMENTS: CSA CRITERIA:	Pass	

Figure 7.1: Test Case: TFR1

Project Name: CloudAid

Test case ID:	TFR5	**Test Designed By;**	Jorge Araújo
Module Name:	SearchEngine	**Test Designed Date:**	27/05/2013
Test Title:	Search for services that fulfill the ServiceTemplate Requirement List	**Test Executed By:**	Jorge Araújo
Test Description:	A set of requirements is specified ofr a particular ServiceTemplate. The result of the search process is the total of all the services that fulfill the requirments list.	**Test Executed Date:**	27/05/2013

Pre-Conditions:	The system must have been initialized with the Heroku ServiceSet (HerokuServiceSet.zip)
	Run in Simulation Mode with the HerokuSimulation environment.
Dependencies:	it depends on the CSAEvaluator module and the Controller module
Post-Consitions:	The system shows all the alternatives found

Step	Test Step	Test Data	Expected Results	Actual Results	Status (Pass/Fail)	Notes
1	Load ServiceSet	HerokuServiceSet.zip	Total triples in ServiceSet: 760	Total triples in ServiceSet: 760	Pass	
2	Start Heroku Simulation (Run Method HerokuSimulation() in Class DataSimulation)		3 (number of ServiceTemplates simulated)	3	Pass	
3	Insert weight data	"2", "n", "2", "y", "3", "y", "1", "y", "4", "y", "3", "y"	7 offerings found: Offering_Heroku_Crane_Database = 50.0 Offering_Heroku_Zilla_Database = 1600.0 Offering_Heroku_Kappa_Database = 100.0 Offering_Heroku_Fugu_Database = 400.0 Offering_Heroku_Baku_Database = 3200.0 Offering_Heroku_Ronin_Database = 200.0 Offering_Heroku_Ika_Database = 800.0	TFR5_TestResults.txt	Pass	The requiremnts list used in this simulation environment hold all the type of requirements nedded for testing TFR5

Figure 7.2: Test Case: TFR5

Note also that in Appendix G an example of the CloudAid Application is presented. This example executes the application with the Use Case defined in Chapter 3 in mind.

7.2 Overall Reliability Tests

The reliability tests play an important role to ensure correct data collection, manipulation and results and so a big effort has been put into these tests.

In order to validate the results obtained by this thesis collection model (in practice represented by the application prototype), it is first mandatory to validate all the data gathering and manipulation. Thus, the reliability tests are focused in the following application prototype operations:

- Service Templates and Criteria weights normalization
- Linked USDL data import
- Service Offering price calculation
- Alternatives attribute values normalization
- Decision XMCDA data import/export
- Admissible Aggregated Solutions calculations

The approach to test these operations was to use the simulation scenarios defined in Appendix E and then manually execute all the calculations. This allowed to produce a comparison sheet. Then by running the prototype with the same simulation scenario we could assess if the results were the same or if there was some sort of miscalculation. In some cases there was the need for external tools. An example is the XMCDA file validation that was performed by using the XML schema provided by the decision deck [39]. The Linked USDL data import was also compared with the service description itself to check if all the data was being correctly imported. The Admissible Aggregated Solutions were tested alongside with the algorithm itself and therefore will be further discussed in Section 7.5.

Therefore, the reliability tests were mostly a manual process, composed of data comparisons. However, they proved fundamental during the development since many miscalculations have been found and dealt with. In other cases it was also possible to enhance some functionalities such as the normalization operations.

All the tests performed and its instructions can be consulted in the CloudAid Reliability Tests Document [8].

7.3 Search Engine Tests

Unlike the Decision Engine which was tested only for its data reliability, which in fact is its biggest concern (mainly the normalization and XMCDA data Import/Export operations) and should offer no performance issues (we are talking about simple mathematical calculation and writing data into a file), the Search Engine has a

different constraint. In order for this thesis concept to be applied the system should be able to perform with a reasonable service set size and realism. Since our service set is composed of Linked USDL service descriptions and more specifically the system is interested in *ServiceOfferings* (see Section 6.3) we need the Search Engine to be able to search specific sets of requirements in a large number of *ServiceOfferings*. This could in fact be an issue. Thus, testing the system response to this scenario is important.

The first step to tackle this performance topic is to find a suitable service set that could serve our purpose. However, due to the Linked USDL recent appearance, there are still not many services described with it. In fact, most of the existing ones were contributions of this thesis (see Section 5.2) but despite their importance for the unit testing and the fact they are descriptions of real cloud services, they are not in sufficient number for testing the Search Engine performance. The solution to solve this issue is to generate our own service set. With this objective in mind it was decided to create a simple application that could generate service sets based on real services and big enough for testing the CloudAid Search Engine. The result is the CloudGen application.

7.3.0.2 CloudGen

The CloudGen is a simple application built with the sole purpose of generating reliable Linked USDL service descriptions based on real service characteristics. What CloudGen does is to take an input file (XML) with a specific configuration of the service features or resources and generate all possible combinations of that configuration. Then it outputs all these combinations as Linked USDL service descriptions. This simple process allows to automate the creation of big service sets but still have some control over the features generated. It then depends on the input file whether or not the services being generated are realistic.

Note that this approach also solved another problem. As stated in Section 6.6 the Search Engine highly relies on the usage of the CloudTaxonomy concepts to serve as keywords. However, for the search system to work properly, an early annotation task must be performed in order for the service description to hold the extra information about their features and resources. These annotation task however, is already integrated into the CloudGen application. Therefore, not only the service set creation is automated but also its services are automatically annotated with the CloudTaxonomy concepts.

The full application and some configuration files examples are avalilable in the CloudAid Public Repository in [7].

7.3.0.3 Performance Tests

With the CloudGen functionality we are able to generate service sets suitable for the performance tests necessary for the Search Engine.

First of all it should be stressed out that more than one service set is used, some generated with the CloudGen application, other built from a real service. However,

all of them are available in the CloudAid Public repository [7]. The used service sets are:

- **HerokuServiceSet** - Based on the Heroku service [1], this service is composed of only a few services, but all real services.
- **FullServiceSet** - Generated by CloudGen it is the biggest service set, composed of 132225 *ServiceOfferings* and 1407726 triples.
- **PartialServiceSet** - Generated by CloudGen it is a smaller version of the FullServiceSet, with 5625 *ServiceOfferings* and 606558 triples.
- **NormTesting** - Also generated by CloudGen it is even smaller than the PartialServiceSet, it was used for some specific tests related to normalization.

With this in mind we decided to use the two biggest service set: FullServiceSet and PartialServiceSet. It was also made a division between two different steps in the search engine. In one side the query execution and in the other the conversion performed by the *ResourceConverter* class. This division was made in order to asses if the conversion task is having any effect on the overall search time.

Table 7.2: Search Times with the Use Case data and Full Service Set (1407726 Triples & 13225 ServiceOfferings)

TemplateID	#Reqs	QueryTime(s)	ConvertTime(s)	#Alt
1	6	13.568579	0.039408	8
2	5	6.643025	0.004751	10
3	10	106.378179	0.015667	10
4	9	103.711556	0.012158	10
5	6	7.589101	0.003059	9
6	4	4.893188	0.002602	8

Table 7.3: Search Times with the Use Case data and Partial Service Set (606558 Triples & 5625 ServiceOfferings)

TemplateID	#Reqs	QueryTime(s)	ConvertTime(s)	#Alt
1	6	8.690287	0.025897	8
2	5	2.578836	0.004781	10
3	10	13.574883	0.011872	10
4	9	12.758888	0.010729	10
5	6	3.423489	0.004228	9
6	4	1.928326	0.002282	8

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

In Table 7.2 and 7.3 we have the obtained results for the search process with the Use Case Scenario. The first conclusion we can draw, without even comparing both tables, is that the conversion task has no significative effect on the overall search time. In fact, comparing the two tables we see that, as expected, the conversion task does not depend on the triple stores size since their values are virtually the same in both tables. These values might however, vary with the number of alternatives found or the number of service properties of each alternative to be converted. Nevertheless, there is no need for testing it with an higher amount of alternatives since from these tables we can extrapolate that the core time of the overall search time is spent processing the search query.

Regarding the search query processing time we see a clear difference between the two triple store sizes, which is a normal conclusion. When the search spectrum grows so does the query processing time. Figure 7.3 shows precisely this difference, based on the triple store size the values for the FullServiceSet are far greater the those of the PartialServiceSet. However, it seams that the query processing time is being influenced by the number of requirements defined for each Service Template. As explained in Section 6.6.2.1 the search query is composed of n subqueries, being n the number of exclusive requirements in the Service Template. Therefore, the higher the number of requirements the more complex the query will be. In fact, further tests were performed in order to assess to what extent this could affect the search time. Figure 7.4 shows for the two service sets used the query search time for an increasing number of requirements. there is a clear increase on the query processing time with the increase on the amount of requirements included in the query. Note that instead of the Use Case Scenario we now used the Requirements Scenario specially conceived for this purpose.

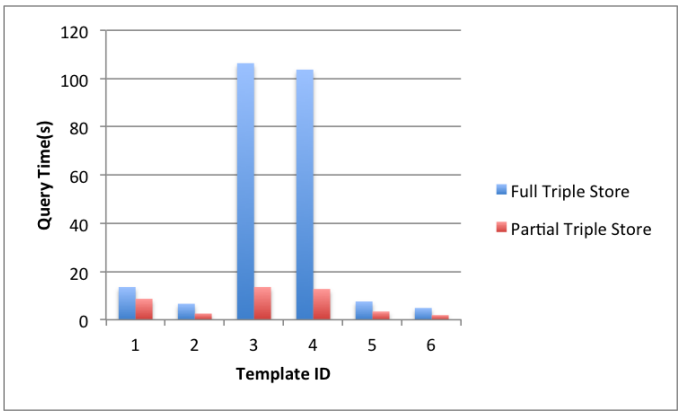


Figure 7.3: Search Time for Each Service Template in the Use Case depending on the Triple Store Size

Consequently, we can say that besides the service set size also the number of exclusive requirements in a Service Template influences the overall search time.

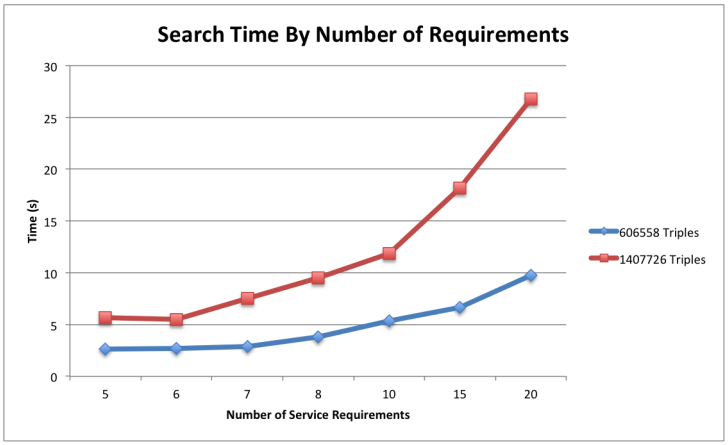


Figure 7.4: Search Time Depending on the Number of Requirements and Triple Store Size

However, by defining too few exclusive requirements the search engine will not be able to reduce the search spectrum, probably presenting too many alternatives for a Service Template, which is also not desirable. This can and will, influence the remaining application steps. By returning a high amount of alternatives the system will need to ask the user too many information to compare all the alternatives (with the usage of the AHP Decision Method for example). The ideal solution would be to reach a compromise in the number of alternatives to return. This could be achieved either by filtering the alternatives or by grouping those that present similar features or resources not being evaluated as criteria. These options should however, be considered for future developments.

7.4 Decision Methods

Although the Decision Methods were not considering a part of the CloudAid Prototype since they are considered external tools. They were in fact tested for their capability to return the required results (ranked alternatives). These unit tests were executed prior to the development of some of the modules. The reason is simple, we needed to assess which could be the methods to be used, and how would they integrate with the CloudAid prototype.

One of the proposed objectives was the system capability to use different decision methods. While the use of two decision methods (SAW and AHP) should by itself prove this capability, it should be stressed out that in order for the system to use a different Decision Method some changes to the prototype might be needed.

Although it is possible to use any kind of Decision Method simply by opening the XMCDA file produced by the Decision Engine module, it might be the case that particular methods require different information. What we are saying is that it is virtually possible to use any kind of decision method based o the current CloudAid Prototype implementation if its information requirements cope with the current produced XMCDA file. However, if different information is required, for example

thresholds for fuzzy calculations some alterations must be made to the prototype. In this situations the system must be updated to support the new types of information to request to the user, if the user must provide them. It should also be added the new support for the generation of the XMCDA file with this new information.

Due to its modularity capability this changes should only be performed in the decision modules (Decision and Aggregation Engines). The support for new information however, should be introduced in the User Interface module as well as the new questions for choosing the right method to apply. Finally, the Controller module must use this new method code to let the decision modules know what they should do.

7.5 Admissible Solutions Algorithm Tests

The Admissible Solutions Algorithm is a key element for the CloudAid prototype since it is responsible by the aggregation component of this thesis conceptual model. Therefore, in order to make sure that the objectives proposed for the algorithm have been achieved, tests for both reliability and performance have to be performed. In spite not having big concerns about performance, we still wanted to know how does the algorithm performs in certain circumstances. The reliability tests, on the other hand are fundamental to prove the accuracy of the resulting data (Admissible Aggregated Solutions found).

7.5.0.4 Reliability Tests

In fact, as depicted in Section 7.2 the Admissible Aggregated Solutions calculations are one of the topics targeted by the reliability tests, which means the testing of the algorithms responsible for finding these solutions. Also the testing approach was similar to the one used for all the other reliability tests. However, in this particular case the complexity of the algorithm lead to the decision of not merging these with other tests in order to ease the comparison process.

One scenario was defined for testing both algorithms. Using the same initial data allows not only for testing results reliability, but also to compare both approaches. In Figure 7.5 we have the expected results for the algorithm with no incomparability support and in Figure 7.6 the expected results for the algorithm with incomparability support. Note that, as already explained, by incomparability we mean the similarity between two alternatives, the threshold used was 0.1. This means that two alternatives are considered incomparable if their performances distance is lower than 0.1 (Incomparability delta).

Applying the algorithm depicted in Algorithm 2 we must obtain the results in Figure 7.5. These include the generated tree. We see that the main differences between the two algorithms is the number of visited nodes and the number of admissible solutions, which are both greater in Figure 7.6 for the incomparability algorithm. By using the Algorithm 3 we must obtain the results in Figure 7.6. As explained in Section 6.8.1, with the incomparability algorithm we must not stop the tree traversal for that node if we find an admissible solution. We still have to compare its chil-

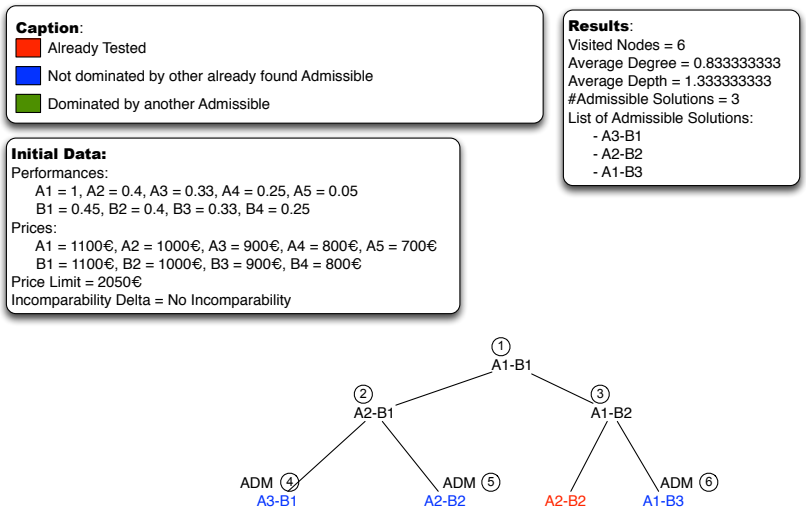


Figure 7.5: Algorithm Testing Scenario and Expected Results (no incomparability support)

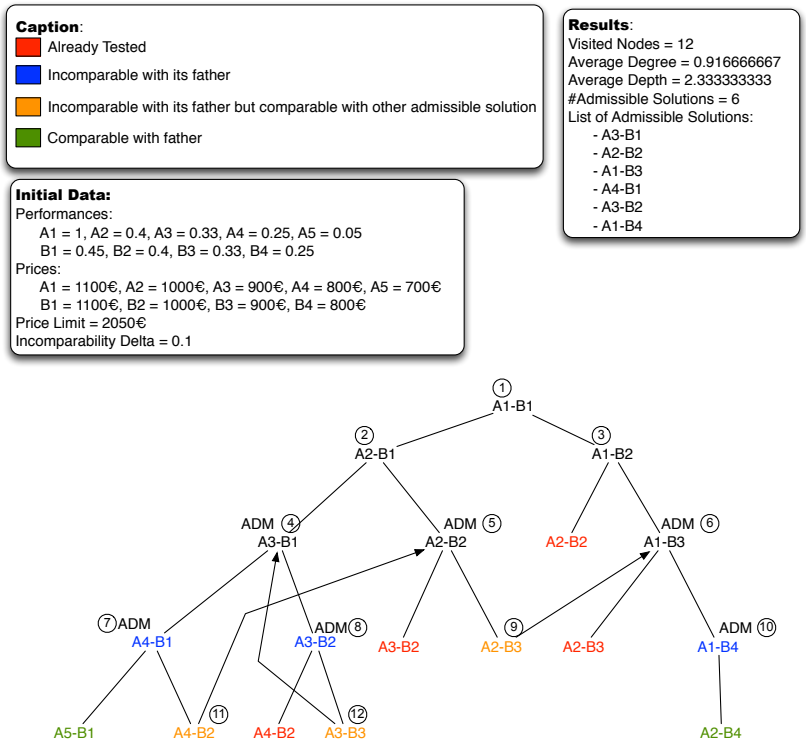


Figure 7.6: Algorithm Testing Scenario and Expected Results (with incomparability support)

dren for incomparability. We can only stop if that incomparability in fact does not exist. This approach leads to an increasing number of visited nodes which can be proven by the two figures. The extra admissible solutions found are also realted to this incomparability. Since we cannot compare these nodes with any other already

found admissible solution we must add them to the list. In Figure 7.6 the orange nodes are examples of cases where the nodes are in fact incomparable with its father but are dominated by other already found admissible solution. In this cases we can discard them from the solutions list and stop the process for this node.

It should be stressed out that all these tests are available in the CloudAid Public repository [7]. In addition a more complex scenario used for deeply testing the algorithm with no incomparability support can be found in the "Scenario3" No Incomparability Document [9].

7.5.0.5 Performance Tests

In order to test the algorithms for its performance, it was first necessary to prepare a testing environment. We needed to simulate the inputs necessary for the algorithm to work. This means that a simulation of a CSA should be produced.

In fact, since the Admissible Aggregated Solutions Algorithms requires not only the user defined data from the CSA (Service Templates and Requirements), but also the data produced and added to the CSA by the previously executed modules (list of Sorted Ranked Alternatives and their attribute values), creating this testing environment was a challenge. Too many input variables exist and the results greatly depend on these initial conditions. However, it would be impossible with the time given to thoroughly test all combinations. This lead to the decision of running a number of different combinations of these values.

Table 7.4: Test Parameters Generation Intervals

Parameter	Interval
#Templates	[2,6]
#Alternatives	[2,9]
Template Weight	[1.0, 5.0]
Price Limit	[0.0, $\#Templates * 1000.0$]
Prices	[0.0, 1000.0]
Performances	[0.0, 1.0]

The approach followed was to automatically generate random scenarios within determined intervals, each of them corresponding to one execution of the algorithm. Although this approach does not allow to achieve the best testing data since there is no linearity among tests, it still allows to extrapolate some conclusions. In Table 7.4 we have the parameters generated for each test and their intervals. Note however, that for each Test Scenario we have a Price Limit (the maximum price of the Aggregated solution for testing admissibility) and the number of Service Templates. Then for each Service Template we have the template decision weight and the number of alternatives. Finally, for each alternative in each Service Template there is

Table 7.5: Tests Data Inputs for the Algorithm Comparison (Simplified)

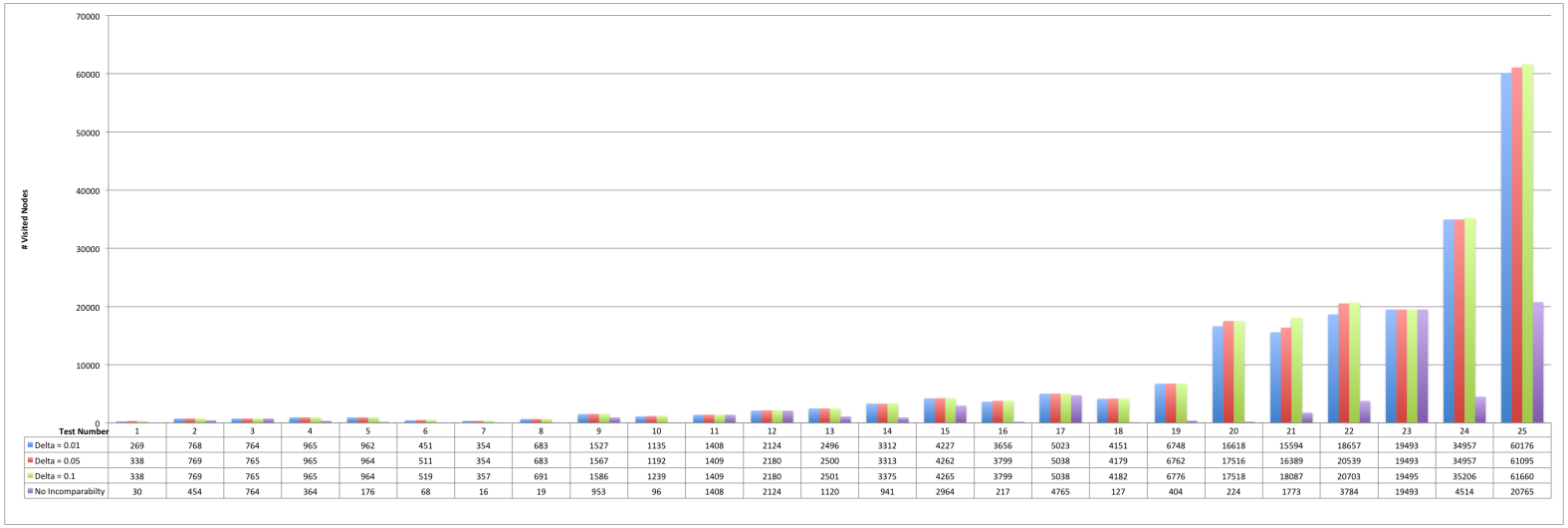
			Templates						
Test Nr	**Combinations**	**Price Limit**	**#Temps**	**Temp 1 #Alts**	**Temp 2 #Alts**	**Temp 3 #Alts**	**Temp 4 #Alts**	**Temp 5 #Alts**	**Temp 6 #Alts**
1	512	1918.96802	4	2	8	8	4		
2	784	1036.20016	4	4	7	7	4		
3	840	406.872278	4	5	4	6	7		
4	980	1337.82079	4	4	7	7	5		
5	1050	1693.84132	4	7	5	5	6		
6	1225	2565.91205	4	7	7	5	5		
7	1296	2960.44128	5	3	9	4	2	6	
8	1344	3567.3097	6	2	6	8	7	2	1
9	1600	1305.97988	5	5	5	4	4	4	
10	1764	2531.32818	4	6	6	7	7		
11	1764	321.245663	4	7	7	6	6		
12	2240	1378.73929	5	5	4	7	4	4	
13	2592	2473.31467	6	9	4	3	6	1	4
14	3500	1461.56963	5	5	5	4	7	5	
15	4320	1181.21801	5	6	6	5	6	4	
16	4536	2427.75394	5	7	3	8	9	3	
17	5040	855.608643	5	6	5	6	7	4	
18	5880	2852.83549	5	7	6	5	4	7	
19	8000	3392.92358	6	5	4	4	5	4	5
20	19600	2752.31012	6	5	7	4	7	5	4
21	20160	2513.24823	6	4	6	7	5	4	6
22	21168	1770.42742	5	7	8	7	6	9	
23	36288	1044.25947	6	6	6	6	4	6	7
24	36864	2307.22284	6	4	4	4	8	8	9
25	61740	1360.26683	6	7	6	7	7	5	6

a performance and price values. Moreover, with each test the following metric are being captured:

- Execution time
- Number of visited nodes
- Number of Admissible Solutions returned
- Average tree depth
- Average tree degree

We started by comparing four different approaches to the algorithm: the algorithm without incomparability support, and the algorithm with incomparability support but with three different values for the incomparability delta (0.1, 0.05 and 0.01). By reducing the incomparability delta we are saying that the alternatives must be much more similar in order to be considered incomparable.

In order to compare the results from these four approaches we used the same 25 tests. A simplified list of the initial conditions of these tests is displayed in Table 7.5. For sake of comprehension it was omitted the performance and price values for each of the alternatives. However, the full version is available in the Comparison Test Inputs Document [4].



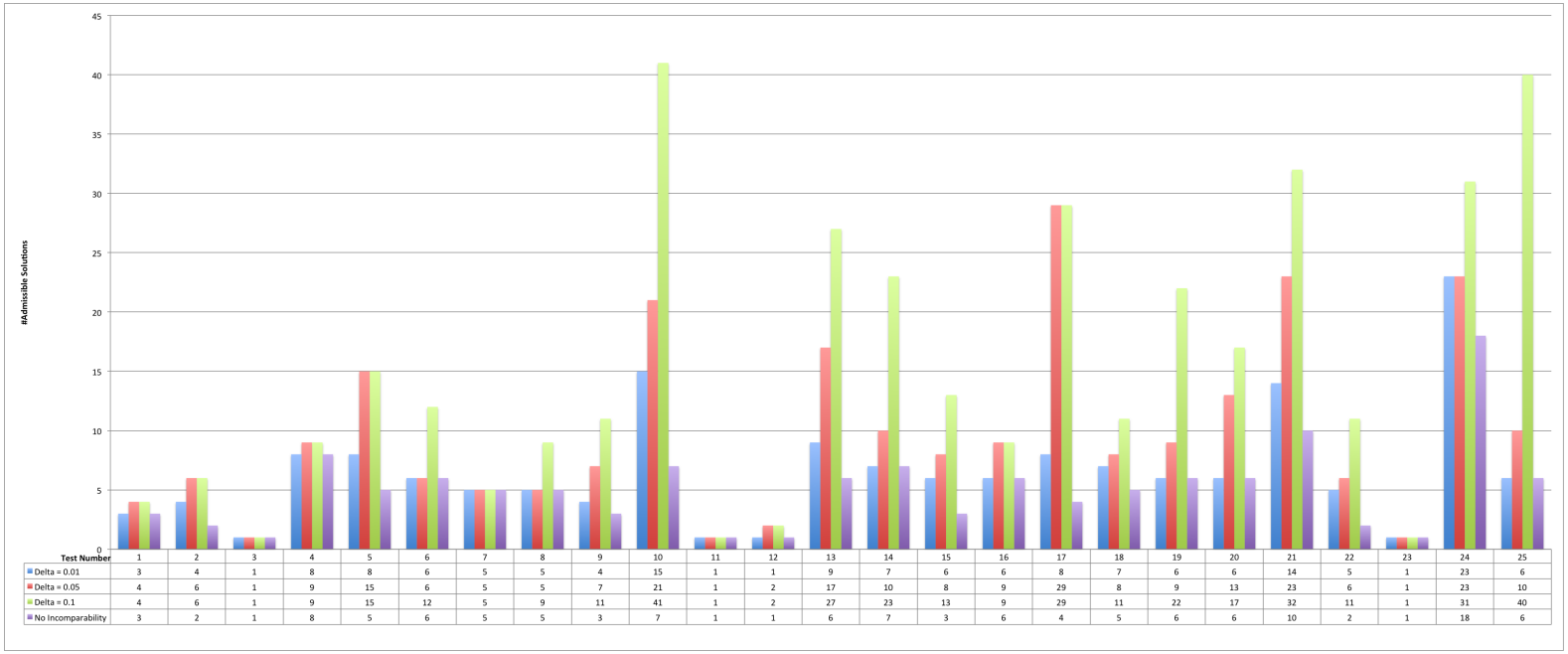
Test Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Delta = 0.01	269	768	764	965	962	451	354	683	1527	1135	1408	2124	2496	3312	4227	3656	5023	4151	6748	16618	15594	18657	19493	34957	60176
Delta = 0.05	338	769	765	965	964	511	354	683	1567	1192	1409	2180	2500	3313	4262	3799	5038	4179	6762	17516	16389	20539	19493	34957	61095
Delta = 0.1	338	769	765	965	964	519	357	691	1586	1239	1409	2180	2501	3375	4265	3799	5038	4182	6776	17518	18087	20703	19495	35206	61660
No Incomparabilty	30	454	764	364	176	68	16	19	953	96	1408	2124	1120	941	2964	217	4765	127	404	224	1773	3784	19493	4514	20765

Figure 7.7: Comparison between Number of Visited Nodes with different Algorithm Variations

Figure 7.7 shows the results obtained for the number of visited nodes. As expected with the increase of possible combinations the number of visited nodes also increases. However, there are some tests where this tendency does not occur. To understand this phenomenon we must look into the price limit parameter. Test 15 and 16 are a good example were the number of visited nodes decreases although the number of combinations is increasing. But in fact, test 15 has a lower price limit, this means that it will be more difficult to find Admissible Aggregated solutions to fit this price requirement. The consequence is the increase on the number of visited nodes. With test 17 and 18 this phenomenon is even more evident.

As we see the algorithm without incomparability support shows much lower values. This was an expected result since, as we already know, unlike the other algorithm, this version stops when an Admissible Solution is added to the final solutions list rather than continuing testing its children. This of course reduces the overall number of visited nodes. However, in respect to the other three cases there are not many differences. A possible reason could be the low number of incomparable alternatives, meaning that the alternatives performances are distant enough from one another to not create many incomparability. Test 23 is an interesting example case though. The values for the four cases are very similar. This in fact can be an extreme case of incomparability non-existence. However, a deeper look into the initial conditions (including prices and performances) of test 23 revealed that in fact there is a lot of incomparability between its alternatives. The reason might once more be linked to the low price limit, but in this case there is also the high values for the alternatives prices leaving not many possible Admissible Solutions. Note that for a solution to be considered Admissible (in this prototype) the sum of all its alternatives prices must be bellow the price limit, meaning that a low price limit and the high alternatives prices reduce the possibility of finding Admissible Solutions.

Therefore, by the results obtained in Figure 7.7 we can only say that the number of visited nodes is influenced by all the initial conditions.



Test Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Delta = 0.01	3	4	1	8	8	6	5	5	4	15	1	1	9	7	6	6	8	7	6	6	14	5	1	23	6
Delta = 0.05	4	6	1	9	15	6	5	5	7	21	1	2	17	10	8	9	29	8	9	13	23	6	1	23	10
Delta = 0.1	4	6	1	9	15	12	5	9	11	41	1	2	27	23	13	9	29	11	22	17	32	11	1	31	40
No Incomparability	3	2	1	8	5	6	5	5	3	7	1	1	6	7	3	6	4	5	6	6	10	2	1	18	6

Figure 7.8: Comparison between Number of Admissible Aggregated Solutions with different Algorithm Variations

In Figure 7.8 we are now comparing the same cases but regarding the number of admissible solutions found. Starting by test 23 we see that only one Admissible solution was returned. This comfirms the theory used for explaining the results of test 23 in Figure 7.7.

However, this new data allows us to conclude a little bit more. As expected, the number of Admissible Solutions found with the no incomparability support algorithm is always equal or smaller than the other cases. In fact we can say that if the value is equal it means that there were no incomparability between Admissible Solutions. That is precisely the case of test 23, but also tests 3 and 11. Note that this does not mean there is no incomparability between alternatives in the initial conditions. It does mean the other tested Admissible Solutions were dominated by at least one of the already found Admissible Solutions. In the combination tree there may exist many Admissible Solutions, however, the algorithm (both with and without incomparability support) only stores in the final solution list those that are not dominated by another Admissible solution already in that list.

Finally, we can also conclude that by increasing the incomparability delta we are also increasing the potential for finding more Admissible Solutions not dominated by another (incomparable). Consequently, we can say that an algorithm with an higher incomparability delta value will return more Admissible Solutions or, in limit cases as already seen, the same amount as an algorithm with a lower incomparability delta value. This does not mean that a lower incomparability delta reduces the possibility to find all the Admissible Solutions. What it does mean is that an high incomparability delta considers alternatives less similar as incomparable thus not dominated, resulting in returning more Admissible Solutions. The trade off in this situation is the number of visited nodes. If an algorithm finds more admissible solutions it must visit more nodes. Thus, a compromise must be achieved in order to maintain a certain degree of incomparability support but also not increasing too much the number of visited nodes, which in limit cases could reach the total number of possible combinations.

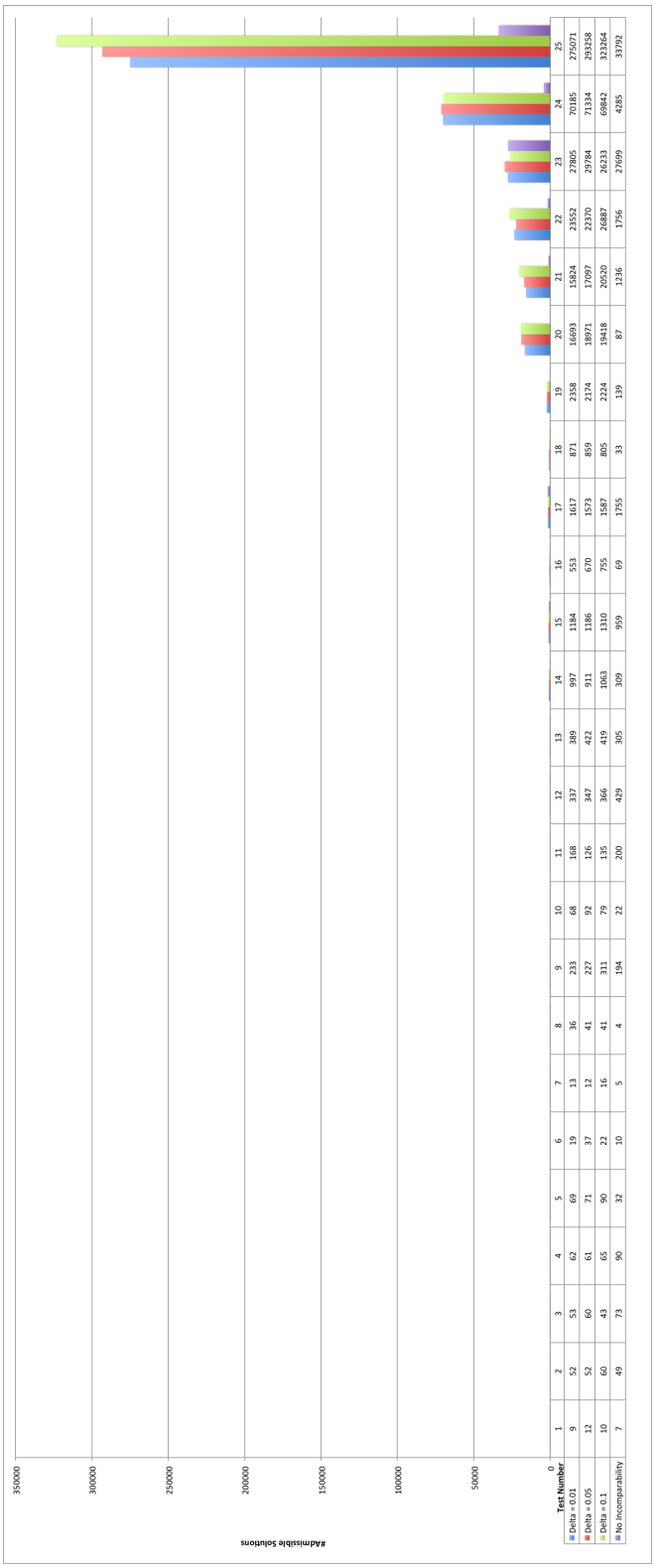


Figure 7.9: Comparison between Execution Time with different Algorithm Variations

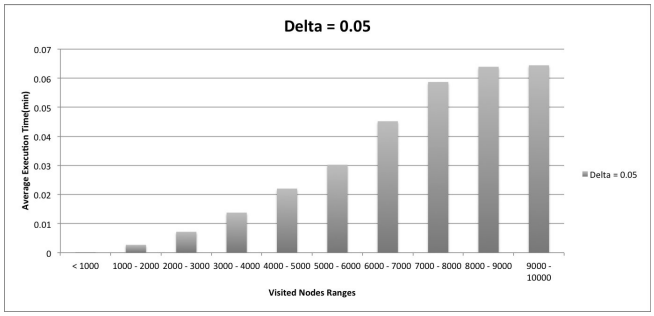


Figure 7.10: Average Times for Delta = 0.5 and less than 10000 Visited Nodes

The last metric analysed is the execution time for each of the tests performed. Figure 7.9 shows the results obtained. Here we see a direct consequence of the number of visited nodes. The greater the visited nodes value the the longer it takes to execute the test. It is in fact an expected result. Going back to the previous discussion and adding this new element, by increasing the incomparability delta we are increasing the number of visited nodes and now we know that we are also increasing the execution time.

In order to better understand the limits of the algorithm further tests were performed to analyse the execution time for different numbers of visited nodes.

From the previous conclusions we decided to use an incomparability delta of 0.5, a medium value of those tested previously. It was also tested the no incomparability support algorithm, although, we cannot compare both algorithms results since they return different visited nodes values. The reasons is that for the same interval of visited nodes different initial conditions were present (e.g.: where for test 200 the incomparability algorithm might return 10000 visited nodes the no incomparability algorithm might return only 50 what would mean that test 200 would be considered for a different interval in the results averages).

For this stress tests a set of 750 tests were executed. However, since many of those tests produced results with too few visited nodes, it was decided to add 15 more tests with high number of combinations and a low price limit. From our previous experience, this would increase the number of visited nodes, thus allowing to stress the algorithm with a lot of work.

The first conclusion was that, in fact for small number of visited nodes the results were satisfactory. However, when increasing values of visited nodes the execution time became longer and longer. Figures 7.10 and 7.12 show the execution times for less than 10000 visited nodes for the algorithm with incomparability support and the one without incomparability support respectively. Figures 7.11 and 7.13 show the execution times for more that 10000 visited nodes.

As we see the times are always increasing, coming to the point where it took more than an hour to execute a test, in this case with more that 210000 visited nodes. It is indeed a lot of time to process the best Admissible Aggregated Solutions. However, it might not be an issue for the user if he does not have to wait for further interaction. Meaning that a user waiting to find the best Aggregated Solution for his required

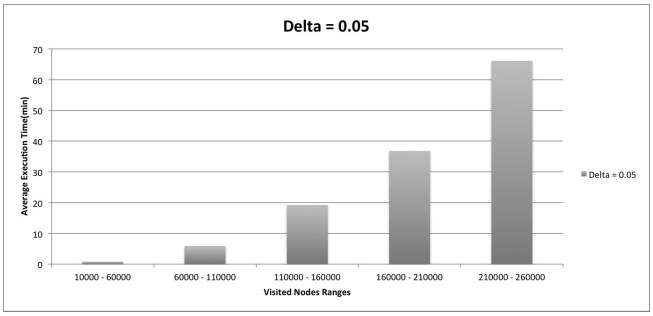


Figure 7.11: Average Times for Delta = 0.5 and more than 10000 Visited Nodes

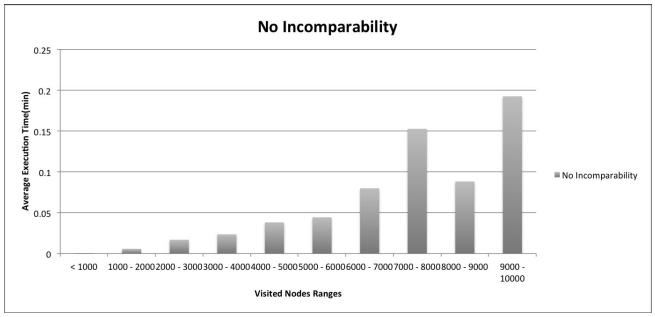


Figure 7.12: Average Times for No Incomparability and more than 10000 Visited Nodes

system would not mind waiting this long for a decision if he could just let the system work and come later for the answer.

This conclusion lead to some changes in the Aggregation Engine execution flow. Instead of collecting the required data about the Service Templates decision weights after the algorithm execution, it was decided to do it before the algorithm execution. The reason is precisely to make it possible for the user to insert all the required data and then leave the system with its processing.

All the tests executed along with the results can be found in the Cloud Aid Public Repository in the Algorithm Tests folder [5].

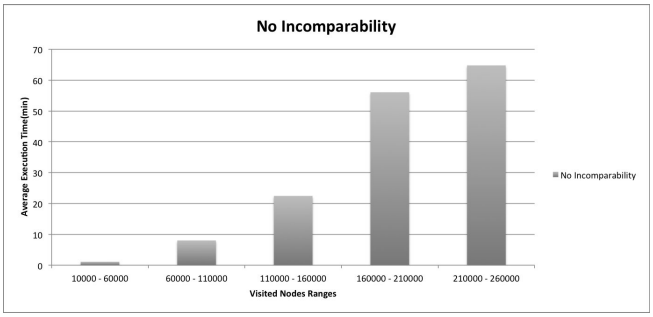


Figure 7.13: Average Times for No Incomparability and more than 10000 Visited Nodes

7.6 Integration Tests

The integration tests represent the final step towards the final prototype. Looking at the CloudAid architecture we see that the integration tests can be seen as simply testing the Controller module since he is the responsible for invoking all the other modules when necessary and keep the information flowing. However, a small exception exists to this simplistic view of the integration tests. We must not forget the external call to the Decision Methods applications. These should also be considered an integration test since these methods although external are still an important asset of the overall prototype.

Using the RAD methodology enables quick development of the prototype modules that can be tested independently. Nevertheless we still have to integrate all these modules and make sure they are working together.

However, as already stated most of the formal test cases were merged. This provided the means to execute the integration tests along with the functional and reliability tests. By testing several modules at the same time, as it was done in most of the reliability tests, it was also being tested the communication between modules and the expected flow of execution. Therefore, ensuring successful integration.

Note that this also includes the communication with the external Decision Methods which are part of the decision process. In this case, their testing is ensured with the execution of the merged normalization and XMCDA data import/export reliability tests.

7.7 Final Test Conclusions

By executing the test plan and ensuring successful implementation of all the "MUST" and "SHOULD" functional requirements but also reliability in the data collection, manipulation and results, we could prove to be possible the creation of a system capable of recommending aggregated cloud services based on a set of requirements.

However, several constraints were raised by these tests. There is the need for a better solution regarding the number of possible alternatives found during the search mechanism. By applying too few requirements it might prove impossible to achieve any kind of feasible decision due to the amount of alternatives. The grouping of similar alternatives or the limitation to only a few are possible solutions. While the first would not erase any possibility, but could possibly hide the differentiation features of some alternatives, the second could easily eliminate better alternatives if there is no previous knowledge of their decision performance. In the other hand by using to many requirements the query execution time might be too long, if any alternative is found at all.

Regarding the decision process, it was proven the utility of the MCDM methods for ranking the set of alternatives. By applying two different methods with different information requirements expressed by the XMCDA file, we also proved the system capability to use different methods. The division of the multi level decision process into different steps allowed to use these external methods for the ranking of the

alternatives and then their results for the computation of the aggregated decision.

The tests performed to the algorithms developed for returning the best Admissible Aggregated Solutions were only preliminary tests, however, fundamental to reach a conclusion that further and more thorough tests must be performed to assess the true nature of the algorithm. Nevertheless, we could draw some conclusions about the various versions of the algorithms.

A final remark goes to the example presented in Appendix G where the Use Case defined in Chapter 3 is used to prove that it is indeed possible to discover and recommend cloud service aggregations based on decision aid mechanisms. Although this example is not a test it highly relies on tested data to show the key functionalities and results of the CloudAid Prototype.

8 Conclusions

This chapter presents the final conclusions of this thesis as well as the contributions achieved during its research and implementation phases.

Section 8.1 summarizes what was proposed and how it was achieved focusing in outlining the key elements described throughout this thesis. The findings and final conclusions are discussed in Section 8.2. What are expected to be the benefits of the work herein presented is the topic of Section 8.3. Finally, it is presented the future work and possible future applications and developments based on this thesis.

8.1 Summary

The main purpose of this thesis was to produce an application capable of proving the feasibility of a cloud service aggregation system based on user requirements and decision methods. The concept by itself aggregates knowledge from several different scientific areas as described in Chapter 1 where much work as been done as stated in the vast state of the art research performed in Chapter 2.

The task of aggregating these different areas, such as service description, composition, aggregation, semantic web, linked data and multi-criteria decision making is by itself a challenging job. However, we are confident that merging these concepts and use them to solve smaller parts of the final task would add to its final value achieving new and forward-looking tools with real world application.

This thesis starts by presenting an innovating new concept to discover new service aggregations by introducing decision methods and applying user preferences besides the constraint based aggregation system such as the one used in [130]. Although, our approach is mainly focused in decision methods, there is still an important constraint influence. This influence is visible through the requirements of the "to-be composed" system specified by the user and used to perform the service discovery (service search).

The search mechanism, although it presents no big developments in its semantic approach, it applies indeed some novel concepts since it uses a service set built from Linked USDL service descriptions. In fact, we can say that this thesis presents one of the first real application of Linked USDL service descriptions for such property-

based discovery purpose. This usage of Linked USDL was not merely the usage of the specification. A collaborative work as been done in order to enhance the description of a particular non-technical aspect of services: pricing. The price is indeed a key element in cloud services adoption by enterprises as expressed in Section 5.2 and therefore was target of an extra effort in order to achieve a suitable description. It should also be stressed the effort put into the elicitation of cloud concepts used by providers in their service descriptions. This effort as stated in Section 5.1 resulted in a new cloud taxonomy that tries to capture features and resources provided by cloud vendors. This allowed also for an improved search mechanism.

This works presents a new algorithm for computing possible service aggregations. We introduce the concept of Admissible Aggregated Solution in an effort to distinguish what is an aggregated solution from what in fact can be aggregated and feasible from both the user and the service constraints point of view.

Finally, all these concepts were compiled and ultimately applied in a prototype application. This prototype recommends service aggregations capable of providing what the user has defined has fundamental (requirements) for his composite system based on his own criteria or preferences. The system was tested with a real world use case presented in full version in Appendix B. Other small use case examples were also used for unit or manual testing. These scenarios are presented in Appendix E.

Note that all the data produced as well as the prototype application itself can be accessed through the Cloud Aid Public Repository [7].

8.2 Findings

Throughout this thesis considerable findings and conclusion were being reported. However, a global and final remark should be done.

According to the tests results presented in Chapter 7 we were able to prove the prototype functionality as desired, as well as ensuring data reliability. Even some performance tests were executed in order to asses the use of the system with higher information processing demands, which, with some raised constraints, returned some reasonable results. All these tests served to prove that, although there are obviously room for improvements, the results obtained prove both the feasibility of applying such concept but also its interest and added value. We can indeed extract useful information for service aggregation based on decision methods.

We also applied the model to our real Use Case (Appendix B) proving that it is possible to use and extract useful information for real world application. Indeed, our entire approach was use case driven. Not only the prototype testing but also the Cloud Taxonomy and the Linked USDL Pricing module development, as stated in Section 5.2. This gives an extra credibility to the overall concept as well as all this thesis contributions.

Summing up, we can say that the presented prototype is indeed a mechanism capable of aggregate (conceptually) cloud services based on user requirements and decision methods. Thus it can be used as a starting platform for improved version and research.

8.3 Implications for Society

As we saw in Chapter 1 Cloud Computing and Cloud Services in particular are a blooming market with huge advantages for those willing to enter it. Thus, this work biggest impact in an highly service oriented society is to increase the accessibility to the cloud, facilitating its adoption to any interested enterprise. In fact the CloudAid prototype impact goes much further than simply reducing the effort in cloud service adoption:

- **Facilitate cloud service discovery based on service properties (requirements)** - by automating the discovery process we are allowing cost and time saving, that can be spent in building designing better systems or businesses.

- **Facilitate provider agnostic cloud service aggregation** - the provider agnostic approach allows for a wider view on the cloud global market. Aggregating several cloud services from multiple providers, knowing they will work together (as Admissible Aggregated Solutions), can reduce costs and improve enterprise efficiency.

- **Providing decision mechanisms to evaluate the best solutions based on the enterprise needs** - the decision aid approach benefits companies interests since they can adapt the results to their goals, relaxing less critical criteria and enforcing those which really make a difference for their business.

Not only the consumers get all the benefits, cloud providers can also extract valuable information from the work herein presented:

- **Allowing providers to test their own service possible aggregations** - the possibility of testing possible aggregations can allow improvements in their own composed service offerings (Service bundles)

- **Evaluate the consumers expectancy** - by knowing what costumers expect of cloud services, through the requirements defined when searching, providers can design new services to complement such customer expectations.

Another important remark goes to the standardization task, achieved both by the pricing description with Linked USDL Pricing module and the Cloud Taxonomy for cloud services features and resources. The first formalizes a novel way to describe pricing models, specially from cloud services (Pay-per-User model), recurring to a semantic approach. The latter tries to standardize how cloud providers describe their services concepts by providing an ontology to be used as annotation tool for cloud service descriptions in RDF. Thus contributing for a federated cloud [1].

[1] http://www.datacenterknowledge.com/archives/2012/09/17/federation-is-the-future-of-the-cloud/

8.4 Future Work

The contributions provided by the work herein presented are indeed a first approach towards, what we expect to be, a full decision-based cloud service discovery and aggregation system. However, precisely from the diversity of concepts from various different research fields it was impossible to fully research each and everyone of them, thus leaving many room for future developments. However, much of the impact comes exactly from the possibilities opened by the prototype presented. Therefore we hope that further developments can build upon this thesis to improve the presented concept. Only a few possibilities, but surely not all, are listed bellow:

- **Better support for the user inserted information** - although there already exists an interesting support for Requirements, Criteria, and Service Templates specification, there is still room for improving how they interrelate, specially regarding global requirements. Improvements towards allowing the recognition of requirements that must be fulfilled by the system as whole are just a possible improvement. There might even occur situations where a user wants his system to have a requirements but he does not care which Service Template fulfills it provided that it is indeed fulfilled.

- **Creation of real cloud service sets** - Although we testes our prototype with concrete data generated from real services a bigger effort should be put to the creation of a cloud service set with descriptions from the providers themselves. In other words the service set should be created based on web crawling of Linked USDL service descriptions, which at a time is not yet possible due to its "still in development" status.

- **Improved search mechanism** - There are several parameters that can be improved in order to enhance the search mechanism: support for different units of measurement, natural language search mechanism and so on.

- **Integration of price functions** - One important contribution was the cloud service dynamic pricing model formalization with Linked USDL. However, it should be considered in future developments the integration of this capability in the search mechanism.

- **Continuos revision of the CloudTaxonomy** - In order to cope with the cloud market developments also the CloudTaxonomy should be focus of constant revisions in order to keep its concepts up to date.

- **Use of different decision methods** - As stated throughout this thesis one objective was to allow the use of multiple decision method. Although two are already used, much more exist with different approaches. It would be interesting to test these other decision methods and assess the consequences for the overall solution.

- **Further Admissible Solutions Algorithm testing** - From the results obtained in Section 7.5 it was already possible to draw some conclusions and to

positively evaluate the approach. However, further and more thorough tests must be executed in order to asses the full nature of the presented algorithm.

The big concern about modularity in the CloudAid Prototype enables precisely these future developments, allowing for easy modification of parts of the full application and testing its results. The presented prototype should be seen as a starting platform for testing new approaches in all its steps (Search Engine, Decision Engine, Aggregation Engine).

A final remark should go to the user interface. By providing a suitable GUI the overall concept can greatly benefit from it. An example is the use of the JAHP graphical interface for the AHP decision method explained in Section 6.7.4.2. Without it, it would be really painful for the decision maker to answer all the required question even more to understand them.

Globally, each component of the prototype can be improved and subject of further research. As already stated, this works aggregates several distinct scientific fields in order to extract added value from them. Thus, this first prototype proves exactly that this can be achieved, leaving however, space for further research and improvement.

Concluding we should say that this thesis presents the first steps towards a full decision-based cloud service aggregation system. However, its potential opens even more doors for future applications thus requiring further development and research.

Appendices

A

Linked-USDL Service Modeling

This chapter aims to describe the process and explain some of the decisions made while modelling a service in Linked-USDL [101].

The service being modelled is BIME [29], a business analytics SaaS provided by "We Are Cloud"[1]. BIME is a service that provides multiple analysis features both for management a business intelligence, and was one of the components described in the use case.

The following sections are organized in the following manner: some of the underlying technologies in section A.1; an explanation of why choosing the BIME service is given in section A.2; an overview of the methodology used to model this service is in section A.3; in section A.4.1 a brief descriptions of the main prefixes and other ontologies used during the modelling process; the service modeling in Linked-USDL is described step by step in section A.4, and finally in section A.5 the description of the service vocabulary where for each feature an example is presented.

A.1 Technologies Used

The technologies used are mostly based in the Semantic Web[2], or Linked Data[3]. All the code was written in Turtle[4] which is a more "human friendly" way of write RDF[5] triples.

A.2 Why BIME?

Looking at the use case description we can easily identify the services needed for the solution, then, why choosing BIME? As a matter of fact any other component could have been used, however as a first exercise in Linked-USDL modelling we wanted a service that had a fair amount of information available and, at the same time, not too complex.

[1] http://www.crunchbase.com/company/we-are-cloud
[2] http://www.w3.org/standards/semanticweb/
[3] http://www.w3.org/standards/semanticweb/data
[4] http://www.w3.org/TR/turtle/
[5] http://www.w3.org/RDF/

This takes out of the equation components like IaaS and PaaS, too complex for the first approach, nevertheless interesting for later use. Also some services have to little information which would make our model a weak example for later use.

With this being said and since BIME is a SaaS and it has a website full of information the choice was evident. Note that the fact that the information is available in the website was decisive.

A.3 Methodology

The methodology used was mainly based on the one followed in [54], where is described one way of modeling a service in Linked-USDL.

In this case before starting to model the service into linked-USDL, a vocabulary was created. This vocabulary is explained in A.5. The construction of this vocabulary was an exhaustive procedure of "scrapping" manually the BIME website to list all the features and concepts of the service. Some concepts are directly extracted from the features list other are inferred from the service description or other information available in the website [6] and its pricing information [7].

After this first step, the linked-USDL modelling starts. Unfortunately not all the required information is available as is the case for the SLA (in A.4.4), one of the linked-USDL modules. This step encompasses the mapping of the concepts described in the vocabulary to the different price plans. This mapping is once again based in the information from the website.

At the time of the writing the Linked-USDL module for the legal aspects is still not complete, this matter will be addressed in A.4.3.

As a final result two final documents are obtained, both turtle files, one is the Linked-USDL instantiation of the BIME service and the other is the vocabulary of the service.

A.4 Linked-USDL Service Modeling

In this section will be explained how the BIME instantiation was made, and some key concepts in linked-USDL vocabulary will be introduced. Note that like the vocabulary definition the service modelling is also a subjective work, as a different person might have choose a different approach. It is also important to state that the Linked-USDL was still under development at the time of writing, and so, some considerations had to be taken for this matter.

A.4.1 Prefixes

This section tries to explain the more important prefixes for the vocabulary and the linked-USDL.

A good thing of Linked-DATA is that you can always refer to the knowledge already collected, and as you can see in Listing A.1, we are doing the same. The

[6] http://www.bimeanalytics.com/

[7] http://www.bimeanalytics.com/pricing.html

first 4 prefixes relate to the linked-USDL description, in this case all 4 modules, they will be used in the linked-USDL instantiation of the BIME service.

The rest of the prefixes are pretty common for an ontology, however some deserve a little note, *GoodRelations*[8], and *SKOS*[9], because they will be the most used through out the service description. The *GoodRelations* is a widely used vocabulary for e-commerce, and so it describes some concepts related to prices, product, and so on. Also *SKOS* is a widely used vocabulary for knowledge organization, typically it is used for hierarchy like concepts.

```
1  @prefix usdl: <http://www.linked-usdl.org/ns/usdl-core#> .
2  @prefix legal: <http://www.linked-usdl.org/ns/usdl-legal#> .
3  @prefix price: <http://www.linked-usdl.org/ns/usdl-pricing#> .
4  @prefix sla: <http://www.linked-usdl.org/ns/usdl-sla#> .
5
6  @prefix owl: <http://www.w3.org/2002/07/owl#> .
7  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
8  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
9  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
10 @prefix dcterms: <http://purl.org/dc/terms/> .
11 @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
12 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
13 @prefix gr: <http://purl.org/goodrelations/v1#> .
14 @prefix time: <http://www.w3.org/2006/time#> .
```

Listing A.1: Vocabulary Prefixes

A.4.2 Service Instance

The service instantiation uses the *usdl-core* and *GoodRelations* vocabularies and some of its concepts.

Let's start by explaining line 3 in Listing A.2, as you can see the attribute *hasClassification* of the *usdl-core* vocabulary is used, this means that the BIME service has a classification of the type SaaS, which is a *SKOS* concept defined in Listing A.3. This was the first consideration taken for the BIME modelling, we wanted to state that the BIME service was a SaaS, and this way we ensure it.

```
1  <#service_BIME> a usdl:Service ;
2   dcterms:title "Bime - SaaS Business Intelligence | Analytics & Dashboards "@
        en ;
3   usdl:hasClassification :SaaS ;
4   usdl:hasProvider :provider_BIME ;
5   usdl:hasLegalCondition :legal_BIME ;
6   gr:quantitativeProductOrServiceProperty
7    :resource_BIME_Connector ,
8    :resource_BIME_Named_Viewers ,
9    :resource_BIME_Dashboard ;
10  gr:qualitativeProductOrServiceProperty
11   bime:Genetal_Features ,
12   bime:Key_Features ,
13   bime:Security_Features ,
14   bime:Customer_Support .
15
16 :resource_BIME_Named_Viewers a bime:Named_Viewers ;
17  gr:hasValueInteger 10.
```

[8] http://www.heppnetz.de/ontologies/goodrelations/v1
[9] http://www.w3.org/2009/08/skos-reference/skos.html

```
18
19 :resource_BIME_Connector a bime:Connector ;
20  gr:hasValueInteger 10.
21
22 :resource_BIME_Dashboard a bime:Dashboard ;
23  gr:hasValueInteger 20.
```

Listing A.2: Linked-USDL BIME Instance

In Listing A.2 line 4 we are stating that the service provider is the one defined in the *provider_BIME* concept, as you can see in Listing A.4. This class holds the data of the company who provides the BIME service.
The next step is to add the legal terms to the service as shown in line 5 of Listing A.2 by using the attribute *hasLegalCondition*. The legal terms will be further explained in section A.4.3.

Finally we add the quantitative and qualitative properties of the service. by using the *GoodRelations* vocabulary, refer to the *GoodRelations* specification for further guidance.

The qualitative properties are the concepts described in the service vocabulary explained in A.5 and basically they are features of the service, in the BIME example, *General_Features*, *Key_Features*, *Security_Features* and *Customer_Support*. The quantitative properties are the service features that can be quantified, this could be storage capacity, number of servers available, or any other quantities, for BIME this is, *Connector*, *Named_Viewers* and *Dashboard*, this means that the BIME service as a specific amount of this resources as described from lines 16 to 23 in Listing A.2. More specifically in line 16 and 17 we are defining that the number of Named Viewers is exactly 10.

```
1 :SaaS a rdfs:Class, skos:Concept ;
2  dcterms:description "Service model to all SaaS like services" .
```

Listing A.3: SaaS Concept

```
1 :provider_BIME a gr:BusinessEntity ;
2  foaf:name "We Are Cloud" ;
3  foaf:homepage <bimeanalytics.com> ;
4  foaf:logo <http://bimeanalytics.com/wp-content/uploads/2011/09/BimeLogo213x66
       .png> ;
5  vcard:email "contact@wearecloud.com" .
```

Listing A.4: BIME Provider

A.4.3 Legal

The legal part of the service is quite simple to model, mostly because the Linked-USDL specification was not yet final at the time of the writing.

As Listing A.5 shows in line 4, the only attribute used is the *hasClause*, this way we are specifying that the class *TermsAndConditions* from the legal module has one clause, stated in line 5 by the class *Clause*. Note that we are only pointing an URL where the information resides.

```
1 :legal_BIME a legal:TermsAndConditions ;
2  dcterms:title "BIME legal statements"@en ;
3  dcterms:description "BIME legal statements are accessible at 'http://www.
       bimeapp.com/terms_of_use.html'. Please consult this website for further
       information"@en ;
4  legal:hasClause
5    [ a legal:Clause ;
6     legal:name "Terms of Use" ;
7     legal:text "http://www.bimeapp.com/terms_of_use.html"@en ] .
```

Listing A.5: BIME Pricing

A.4.4 SLA

The SLA, Service Level Agreement, was one of the missing parts from the website. Unfortunately the providing company gives no significant information about this matter which makes it impossible to model this particular part of the service.

A.4.5 Pricing

The pricing is probably the most important part of any service, and in Linked-USDL it is where we define the components which make a product offering. The first step is defining the service offerings, this is, what the provider offers, the product.

A.4.5.1 Service Offering

Listing A.6 shows the definitions of the BIME offering. A *usdl-core* class is used for this effect *ServiceOffering* and as any product a price plan must be associated with this offering as you can see in line 5 by using the *usdl* attribute *hasPricePlan*. In BIME we have 6 different price plans one for each product edition, defined in the vocabulary A.5.1. Note that by using the *includes* attribute in line 4 we are saying that this offering is from the service described above in Section A.4 . Defining the 6 price plans is the next step.

```
1 :offering_BIME a usdl:ServiceOffering ;
2  dcterms:title "BIME instance"@en ;
3  dcterms:description "Offering for a BIME instance."@en ;
4  usdl:includes <#service_BIME> ;
5  usdl:hasPricePlan
6    :pricing_10_day_free_trial ,
7    :pricing_Quick_start_pack ,
8    :pricing_Eterprise_pack ,
9    :pricing_Premium_pack ,
10   :pricing_Big_data_pack ,
11   :pricing_Server_pack .
```

Listing A.6: BIME Offering

A.4.5.2 Price Plans

In Listing A.7 you can see the definition of the enterprise pack pricing. Here 2 concepts from the *usdl-pricing* vocabulary were used, the *hasBillingCycle*, (line 4),

which defines how often and the cycle that the client pays, in this example the client pays every one month; and the *hasPricingComponent* (line 8) which will have the description of the amount that has to be paid during this cycle and which service components are included in the price.

```
1 :pricing_Enterprise_pack a price:PricePlan ;
2  dcterms:title "Enterprise pack"@en ;
3  dcterms:description "Fee for general use"@en ;
4  price:hasBillingCycle
5    [ a gr:QuantitativeValue ;
6     gr:hasValueInteger "1" ;
7     gr:hasUnitOfMeasurement "MON" ] ;
8  price:hasPriceComponent
9    :priceComponent_Enterprise_pack_General .
```

Listing A.7: BIME Enterprise Pack Example: Price Plan

A.4.5.3 Price Components

The third and final step is to define this *PriceComponent* class used in Listing A.7 line 9. We do this as shown in Listing A.8. In line 4 we are stating that this particular price component (*priceComponent_Enterprise_pack_General*) is linked to a list of service features by using the attribute *isLinkedTo*. Now we have to list all the BIME features that the Enterprise Pack includes. Note that all the next lines from Listing A.9 to A.20 start with the word *bime*, this means that they are concepts defined in the service vocabulary A.5.

```
1 :priceComponent_Enterprise_pack_General a price:PriceComponent ;
2  dcterms:title "General price"@en ;
3  dcterms:description "Fee for general usage of the instance."@en ;
4  price:isLinkedTo
5    [ a bime:Named_Viewers ;
6     gr:hasValueInteger "10"] ,
7    [ a bime:Connector ;
8     gr:hasValueInteger "10"] ,
9    [ a bime:Dashboard ;
10      gr:hasValueInteger "20"] ,
```

Listing A.8: BIME Enterprise Pack Example: Price Component (Part 1)

The final lines in Listing A.8 define the quantitative properties of the enterprise pack, from line 5 to 10 we are saying that this pack has exactly 10 *Named_Viewers*, 10 *Connectors* and 20 *Dashboards*.

The following code examples, are the description of the several features of the service included in the Enterprise Pack.

General Features

In Listing A.9 we describe the general features included in this pack. As the name says general features are those included in all service editions, and they include things like exporting data to multiple formats, desktop application, on premise or cloud storage and so on.

```
1  #generel features
2   bime:On_premise_or_cloud_data_storage ,
```

```
3    bime:Dashbord_unique_url_sharing ,
4    bime:Export_data_to_multiple_formats ,
5    bime:Mobile_device_dashboard_comsumption ,
6    bime:Desktop_application ,
7    bime:Data_dashboard_security_options ,
8    bime:Data_snapshot_and_upload ,
9    bime:Integration_via_web_service ,
```

Listing A.9: BIME Enterprise Pack Example: Price Component (Part 2: General Features)

All these features are described in more detail in A.5.2. However as an example Listing A.10 shows the definition of the *General_Features* group. As you can see in line 4 we are using the *skos:narrower* attribute, this mean that all the bellow lines are children of the *general_features* class, this is, all the bellow concepts are general features of the service.

```
1  :general_features a rdfs:Class , skos:Concept ;
2   rdfs:subClassOf gr:QualitativeValue ;
3   skos:prefLabel "General Features"@en ;
4   skos:narrower
5     :On_premise_or_cloud_data_storage ,
6     :Dashboard_unique_url_sharing ,
7     :Export_data_to_multiple_formats ,
8     :Mobile_device_dashboard_consumption ,
9     :Desktop_application ,
10    :Data_dashboard_security_options ,
11    :Data_snapshot_and_upload ,
12    :Integration_via_web_service .
```

Listing A.10: BIME Service Features: General Features

```
1  :On_premise_or_cloud_data_storage a rdfs:Class , skos:Concept ;
2   rdfs:subClassOf gr:QualitativeValue ;
3   skos:prefLabel "Storage in the cloud or on-premise"@en ;
4   skos:broader :general_features .
```

Listing A.11: BIME Service Features: On Premise or Cloud Storage

In Listing A.11 the first concept of the general features group is defined, the *On Premise or Cloud Storage*, in line 2 we are saying that this concept is a subclass of the *GoodRelations:QualitativeValue* and in line 4 we explicitly specify the inheritance to the *general_features* class by using the *skos:broader* attribute.

Security Features

In Listing A.12 we continue to list the Enterprise Pack features, in this example we list the security features. Again this features are explained in the service vovabulary in section A.5.4. Security features are those responsible for ensuring data confidentiality or connection security.

```
1  #security features
2    bime:Encrypted_connections ,
3    bime:Data_encryption ,
4    bime:Daily_backups ,
5    bime:Secure_login ,
```

Listing A.12: BIME Enterprise Pack Example: Price Component (Part 3: Security Features)

The example of Listing A.13 shows the same procedure of the general features, again, note the use of the *skos:narrower* to define the child classes of the security features group and in Listing A.14 the skos:broader. As you have already noticed we are using all the security features in the Enterprise Pack, this means that all the features are included in this pack.

```
:Security_features a rdfs:Class , skos:Concept ;
  rdfs:subClassOf gr:QualitativeValue ;
  skos:prefLabel "Security Features"@en ;
  skos:narrower
    :Encrypted_connections ,
    :Data_encryption ,
    :Daily_backups ,
    :Secure_login .
```

Listing A.13: BIME Service Features: Security Features

```
:Encrypted_connections a rdfs:Class , skos:Concept ;
  rdfs:subClassOf gr:QualitativeValue ;
  skos:prefLabel "Encrypted connections to both amazon S3 and BIME servers"@en
      ;
  skos:broader :Security_features .
```

Listing A.14: BIME Service Features: Encripted Connections

Customer Support

Again the customer support follows the same process. In Listing A.12 we list the customer support components included in the Enterprise Pack. The customer support are those components that the service have to help the customer to better use the service, like product manuals, chat rooms, forum and so on, in total there are 7 components, and as you can see we are including all of them in this pack.

```
  #Customer Support
    bime:Product_manual ,
    bime:Chat_room_for_questions ,
    bime:Support_forum ,
    bime:Tuturial_videos ,
    bime:Getting_started_guides ,
    bime:Free_live_online_demo ,
    bime:Glossary ,
```

Listing A.15: BIME Enterprise Pack Example: Price Component (Part 4: Customer Support)

Listing A.16 shows the Customer Support class definition, again the *skos:narrower* is used to list the child concepts. An example of a child concept of the customer support is shown in Listing A.17, the *skos:broader* is again used for showing inheritance. This concepts are explained in more detail in Section A.5.5.

```
:Customer_support a rdfs:Class , skos:Concept ;
  rdfs:subClassOf gr:QualitativeValue ;
```

```
3   skos:prefLabel "Customer Support"@en ;
4   skos:narrower
5     :Product_manual ,
6     :Chat_room_for_questions ,
7     :Support_forum ,
8     :Tutorial_videos ,
9     :Getting_started_guides ,
10    :Free_live_online_demo ,
11    :Glossary ,
12    :2_hour_personalized_support.
```

Listing A.16: BIME Service Features: Customer Support

```
1 :Product_manual a rdfs:Class , skos:Concept ;
2   rdfs:subClassOf gr:QualitativeValue ;
3   skos:prefLabel "Product manual"@en ;
4   skos:note "https://docs.google.com/a/wearecloud.com/document/pub?id=1
        V4SpbGJrU3P3zg-XkWlrGC-ZXXPa9FJQPxDYx3Z73rY" ;
5   skos:broader :Customer_support .
```

Listing A.17: BIME Service Features: Product Manual

Dashboards

A dashboard is a screen for data visualization, refer to section A.5.7 for more information about dashboards.
In the Enterprise Pack example we use 2 dashboards (Listing A.18), however as shown in Listing A.19 bellow the *skos:narrower* attribute 4 dashboard concepts exist in total, this means that the enterprise pack only allows two of them, the *Basic_Dashboard* and the *Standard_dashboard*.

This information is extracted directly from the website, however from all the features listed in section A.5 not all of them have to be used, this is, for each service edition some decisions have to be made to choose which features are included in which editions.

```
1   #Dashboards
2     bime:Basic_Dashboard ,
3     bime:Standard_Dashboard ,
```

Listing A.18: BIME Enterprise Pack Example: Price Component (Part 5: Dashboard)

```
1 :Dashboard a rdfs:Class , skos:Concept ;
2   rdfs:subClassOf gr:QualitativeValue ;
3   skos:prefLabel "Dashboard definition"@en ;
4   skos:narrower
5     :Basic_Dashboard ,
6     :Standard_Dashboard ,
7     :Advanced_Dashboard ,
8     :Customization_Dashboard .
```

Listing A.19: BIME Service Features: Dashboard

Connectors

A connector is a point where BIME is getting information from, a data source. Read section A.5.6 for more information about connectors.

Continuing listing the enterprise pack features, in Listing A.20 3 connectors are used, all of them described in the service vocabulary. Once again not all connectors are used, in total 4 connectors are described in Listing A.21, but only 3 are included in the enterprise pack price plan.

```
#Connectors
  bime:Basic_Connector ,
  bime:Sql_Connector ,
  bime:Advanced_Connector ,
```

Listing A.20: BIME Enterprise Pack Example: Price Component (Part 6: Connector)

```
:Connector a rdfs:Class , skos:Concept ;
 rdfs:subClassOf gr:QualitativeValue ;
 skos:prefLabel "A connection to a source of data"@en ;
 skos:narrower
   :Basic_Connector ,
   :Sql_Connector ,
   :Advanced_Connector ,
   :Big_data_Connector .
```

Listing A.21: BIME Service Features: Connector

Key Features

The key features are those related to data filtering and manipulation. These are the features that truly influence the differences between editions. This means that some of the features listed in Listing A.23 will not be used in all pack, as the enterprise example shows only 7 out of the 13 key features are included.

As shown in Listing A.22 the enterprise pack has features like post processing, value/atribute filtering, multiple sources query blender and so on. These features are all described in more detail in section A.5.3.

Note that some of the features might not be directly mentioned in the website, and so some have to be inferred from the editions descriptions given. These concepts are a personal choice and you have to have this in mind while modeling a service, however subjectivity should be avoided, a goal that is not always possible to achieve in the first attempt, which makes the modeling process an iterative process.

```
#Key features
  bime:Post_processing_functions ,
  bime:Value_attribute_filtering ,
  bime:Google_Analytics_API_integration ,
  bime:Multiple_sources_query_blender ,
  bime:Calculation_engine ,
  bime:Drag_and_drop_querie_creator ,
  bime:Messaging_function ;
```

Listing A.22: BIME Enterprise Pack Example: Price Component (Part 7: Key Features)

```
:Key_features a rdfs:Class , skos:Concept ;
 rdfs:subClassOf gr:QualitativeValue ;
 skos:prefLabel "Key Features"@en ;
 skos:narrower
   :Post_processing_functions ,
   :Full_customizing_dashboard_functions ,
   :What_if_analysis ,
   :Drill_Down_analysis ,
   :Drill_Through_analysis ,
   :Cross_filtering ,
   :Value_attribute_filtering ,
   :Google_Analytics_API_integration ,
   :Multiple_sources_query_blender ,
   :Calculation_engine ,
   :Drag_and_drop_query_creator ,
   :Messaging_function ,
   :Behind_firewall_deployment .
```

Listing A.23: BIME Service Features: Key Features

Price

The final step is the price itself for this component, in Listing A.22 line 1, the attribute *hasPrice* of the the *usdl-price* module is used for this matter, then we define the pricing specification by using the *GoodRelations* vocabulary, in this example 180 US Dollars per analyst per month.

```
price:hasPrice
  [ a gr:UnitPriceSpecification ;
    gr:hasCurrency "USD" ;
    gr:hasCurrencyValue "180" ;
    gr:hasUnitOfMeasurement "Analyst/MON" ] .
```

Listing A.24: BIME Enterprise Pack Example: Price Component (Part 8: Price)

Now we have to repeat this process for all the price plans we wish to define, in BIME case 5 more times as you can note in Listing A.6. Only after this the full service will be modeled.

A.5 Service Vocabulary

As mentioned in A.3, in order to express the service a vocabulary had to be created, this vocabulary is nothing else but a description of the features provided by the service. In other words the vocabulary is a number of classes each one describing a key concept of the service. A concept can be a feature, a resource or any other relevant information about the service being modelled. These concepts are fundamental to understand the service, what it does, how it is done and so on. Note that all the information in this vocabulary was extracted directly from the website[10].

Because the interpretation of the information retrieved can vary with the person making the modelling, there is an high amount of subjectivity associated with this concepts, this is the goal of this section, to reduce the subjectivity of the concepts presented in the vocabulary.

[10] http://bimeanalytics.com/

For each group of concepts a brief explanation of it's purpose is given, and then each concept will be explained with an example if possible to better understand their purpose. Note that model used is a personal choice so a different person might have chosen a different approach, for this matter all the relevant choices will be explained whenever necessary.

```
:general_features a rdfs:Class , skos:Concept ;
  rdfs:subClassOf gr:QualitativeValue ;
  skos:prefLabel "General Features"@en ;
  skos:narrower
    :On_premise_or_cloud_data_storage ,
    :Dashboard_unique_url_sharing ,
    :Export_data_to_multiple_formats ,
    :Mobile_device_dashboard_consumption ,
    :Desktop_application ,
    :Data_dashboard_security_options ,
    :Data_snapshot_and_upload ,
    :Integration_via_web_service .
```

Listing A.25: General Features Concept

In Listing A.25 we have an example of a concept definition, in this case the general features concept. By using the attribute *narrower* of the SKOS ontology we are specifying that all the below concepts are children of the general features concept. In the opposite way in Listing A.26 we have the child using the *broader* attribute meaning that the below class is the father of the current concept, this meant that the *On_premise_or_cloud_data_storage* concept is one of the *general_features* concepts. This 2 examples are illustrative of all the other concepts defined in the vocabulary since all of them are described in the same way.

```
:On_premise_or_cloud_data_storage a rdfs:Class , skos:Concept ;
  rdfs:subClassOf gr:QualitativeValue ;
  skos:prefLabel "Storage in the cloud or on-premise"@en ;
  skos:broader :general_features .
```

Listing A.26: On Premise or Cloud Data Storage Concept

A.5.1 Product Editions

As any other software product, BIME also have different levels of service, each one with different features and different prices.

There are 4 main product editions: Enterprise, Premium, Big Data, Server. However a decision was made to include 2 more concepts and treat them as 2 more product editions in spite of them being a simple variation of the previous 4 editions. They are: 10 Day Free Trial and Quick Start Pack.

Since the first 4 packs are pretty well described in the website [11], there is no need for further description. However the 2 extra packs need a little description. The first, 10 Day Free Trial, a nothing more than a full feature edition for 10 days time where the client can test all the available features for later choosing the best pack to

[11] http://www.bimeanalytics.com/pricing.html

subscribe. The second, Quick Start Guide, is more like a promotional starting kit, where the client has 1 Premium Licence of the service, plus 2 hours of personalized support per month, this kit lasts 3 months and has fee of $500 per month.

These editions could have been simply used as pricing plans in the Linked-USDL pricing description, however, by including them in vocabulary the goal is to achieve a richer service description

A.5.2 General Features

This group of concepts contains the basic concepts of the service. Mainly these are the features fundamental for the core of the service, the data, sharing and accessibility. The concepts related to data manipulation and analysis were grouped under the Key Features, explained in section A.5.3.

Below each concept will be briefly explained and an example presented when possible.

On Premise or Cloud Data Storage BIME allows clients to choose between two data storages options. The data can be stored either on the cloud, and BIME as their own servers, or on the client environment.

Dashboard Unique URL Sharing The definitions of dashboard is explained in section A.5.7. What this feature allows is to refer to any dashboard created with BIME using an unique URL. Imagine that an analyst is compiling a sales report for his boss, however his boss wants to share this information with all department managers, they can all access this dashboard through the URL, embedded in the company web page for example.

Export Data to Multiple Formats The client can export his views or dashboards to any file format, jpg, excel, PDF, CSV, and so on.

Mobile Device Dashboard Consumption BIME has the possibility to be used in mobile devices. Dashboard consumption means that the created dashboards can be viewed in all major mobile devices. It is also possible to use BIME as a mobile application in iOS or Android phones or tablets.

Desktop Application Besides de service being available as a SaaS in the cloud it has also a desktop application that the analysts can use to work. It is called BIME Desktop and is available for Windows and Mac OS.

Data Dashboard Security Options This is not a feature well explained in the website, a few considerations had to be done here. By data dashboard security options we mean the possibility to create groups of viewers or passwords for the dashboards or only parts of the dashboards, this way only a few defined persons can visualize de information.

Data Snapshot and Upload BIME gives the possibility to, if you don't have your data stored in the cloud, take a snapshot of it and upload it to BIME distributed cache. This feature is also called "Déjà Vu".

Integration Via Web Service This means that you can connect to web services to extract data. Imagine that your company has a webservice for data retrieval, with BIME it is possible to use that web service and integrate it with the rest of the data.

A.5.3 Key Features

The key features group has the concepts related to data analysis and dashboard manipulation. Some of these features are particular to some editions others are present in all editions.

Below each concept will be briefly explained and an example presented when possible.

Post Processing Functions Post Processing functions are the options available to the user after creating the query for the data, organizing columns, rows, aggregate, change how the data is presented, percentage or simple numbers and so one.

Full Customizing Dashboard Functions This feature is actually a set of features, instead of listing all the customization possible to a dashboard it was decided to group them in this feature. Customization features include all operations of re-sizing, adding documents, pictures, rich formatted text, etc, to a dashboard.

What If Analysis What if analysis allows the client to suppose possible scenarios and see the results. For example, What would happen to my revenues if i change price?

Drill Down Analysis BIME has the possibility to make a Drill Down analysis, by going into more detailed data if the client wants to. For example, Category - Sub-Category - Product Name

Drill Through Analysis Drill Through analysis allows the client to display underlying data by examining results across different dimensions. For example, Man, Woman, Children.

Cross Filtering This is the most advanced filtering option and it allows to make any kind of filtering to the data retrieved.

Value Attribute Filtering This feature is the more basic filtering option, it lets the client filter by attribute or by value. For example filter cities by country, or filtering cities by country name Portugal.

Google Analytics API Integration It is possible to fully integrate the client website data using Google Analytics API.

Multiple Sources Query Blender The query blender is a features that allows the client to make queries from different data sources. imagine that you have a Mysql database of clients, and a CSV with the billing information, with query blender in the same query you can retrieve data from the two sources.

Calculation Engine The calculation engine is an easier way to compose complex functions for calculating data. It allows to group things dynamically, filter them based on complex rules or measure the impact of a change on your other data.

Drag and Drop Query Creator To create a query in a dashboard the only thing the client has to do is to drag&drop data fields, attributes or operations.

Messaging Function It is possible to leave comments in any dashboard, either for explained complex data, or reducing subjectivity of graphs and so on.

A.5.4 Security Features

Since BIME is a data analysis tool and works with important business data it is key to ensure that all connections and communications are secure, this was the reason to include a pack of security concepts provided by the BIME service. The security features were found through out the website, and they are the same to all product editions. Four features regarding security were defined and are explained below:

- **Encrypted Connections** - this means that all the connections are encrypted by default. The connections can be to the BIME servers or to Amazon S3 servers;
- **Data Encryption** - whenever data is transferred from one point to the other this data is encrypted by default;
- **Daily Backups** - if the clients opts for keeping the data in the BIME servers, the service ensures that daily backups are scheduled;
- **Secure Login** - all the logins made to the system, are secure.

A.5.5 Customer Support

As any good service BIME also have customer support, as an important part of the service it was included in the vocabulary. However most of its concepts were not explicitly explained in the website and had to be inferred by other features. For example, the *2 Hour Personalized Support* is never mentioned in the support area of the website, the only time it is mentioned is when the *Quick Start Pack* is described, nevertheless, it was included in the *Customer Support* class of concepts.

Since this concepts are self-explanatory there is no need for further explanation. A full list is shown bellow:

- Product Manual;
- Chat Room for Questions;
- Support Forum;
- Tutorial Videos;
- Getting Started Guides;
- Free Live Online Demo;
- Glossary;
- 2 Hour Personalized Support.

A.5.6 Connector

A connector represents a data resource, this is, a database or a spreadsheet is a connector, because it is a peer where BIME can extract data from. Since this is a key concept for the overall understanding of the service it was included in the vocabulary, in fact, several different types of connector are described below.

- **Basic** - A basic connector can retrieve data from: Excel, SalesForce, Google Docs, CSV, SimpleDB, Lighthouse and SOAP;
- **SQL** - A sql connector can retrieve data from almost any RDBMS available in the market, for example, SQL Server, Mysql, Oracle,...;
- **Advanced** - The advanced connector can retrieve data from Google Analytics and XMLA OLAP engines such as Mondrian, Analysis Services, ...;
- **Big Data** - The big data connector is intended for the Google BigQuery and SAP HANA.

A.5.7 Dashboard

A dashboard by definition is a screen where information can be displayed, in a quick and effective way. For BIME a dashboard is the core of the product, this is, the dashboard is the final result, where all the data is presented after being processed and customized by the analyst. Once again this is a key concept that should be included in the service vocabulary. The service also establishes 4 different levels of dashboards, some allow give the user full customization power other are a little more limited. A full list of the dashboard descriptions follows:

- **Basic** - The basic dashboard allows the user to make basic interactions and customization, such as value and attribute filters and some post processing options;
- **Standard** - This dashboard includes data filter prompts for slicing data;

- **Advanced** - Has different types of analysis, such as What If, Drill Down, Drill Through, Cross filtering and allows the user to re-size windows as well as positioning different objects (images, text,...);
- **Customization** - Full customization, insertion of logos, pictures, documents, hyperlinks,...

A.5.8 Extra Considerations

Besides the above explained concepts 2 more were later added. These concepts concern sharing and viewing capabilities, and were considered of value for the service description. They are the *Basic Viewer* and the *Named Viewer*. The first is unlimited, and is directed for the common use, this means that everyone can access them by an unique URL, of course it can be protected by a password if the client wishes. The second is directed for a specific viewer and is protected by a login and password, they are limited to 10 per license.

B

Use Case (Full)

B.1 Problem Description

We start by introducing our IT company, with 200 employees, most of them working in the only building the company owns. Taking into account the number of electronic devices constantly running in the building, the company knows the energy consumption is high, and is concerned about reducing it to a minimum, reducing their operational costs. In addition to the above concerns, the European Union directives point to a 20% cut on the energy consumption as well as encourage the use of energy meters to monitor consumption and efficiency [1].

Moreover, the constant increase in social awareness for green energy and environmental concerns, press the company to adopt these measures in order to maintain a "green image" and transparency standards to their consumers.

Since the building is full of electronic components there will be a huge amount of sensors to retrieve the data for energy consumption and management. There are many different approaches stated in [63], [50], [1], [119], [62]. With this many devices the amount of data to collect, process and predict is huge, which leads to a fair capability of storage and data processing. The company currently has none of this capabilities to dedicate to their Energy Monitoring and Efficiency System (EMES), and so it is willing to outsource these capabilities. Using the same arguments from Section 1.3, choosing the cloud seems a good option.

The cloud will allow the company to surpass the storage and processing of all the data, without building their own infrastructure. However, they also want to make this data public through a website, that can be accessed by anyone. Also, due to the European Union above mentioned concerns, a monthly report should be delivered, for this matter the EMES needs to automatically create and send reports.

To facilitate the usability and ease of use, an app that allows employees and managers to monitor and control the system should be developed. For eliminating the "in loco" controlling approach the company wishes to adopt an SMS system to control the EMES, as well as to alert the right person if something of relevance happens. The goal is to make the system as controllable as possible without the need of a permanent manager in the building.

[1] http://ec.europa.eu/energy/efficiency/index_en.htm

B.2 System Usage

In this section we present the actors and their interactions with the EMES, The purpose is to better understand the system concept and the reason for some of the choices for potential cloud services solutions.

Interacting with the EMES we can define 4 major actors, each of them with their own functionality, and different access to information generated by the EMES. The first identified actor is the sensor that will collect data from the correspondent device and transmit it to a wireless receiver. Note that hundreds of sensors will be collecting data at the same time and transmitting it to the main receiver.

The second actor is the building manager, or the person in charge of maintenance or controlling the energy system. The three main interactions with the system are:

- View data (readings, predictions, reports, efficiency,...)
- Create patterns (prediction patterns, SMS alerts, report patterns,...)
- Control the system (SMS commands, App Control, resolve issues,...)

The third actor is the common employee, that only has access to some of the features available to the building manager. They can view most of the information available (average comsumption, ratios, etc), and can even be allowed to make some extra simple interactions through the mobile app or website, such has turn on or off their equipment, or communicate a malfunction to the building manager.

At last the general population. This actor has access to public information, allowed by the company, such as overall readings, efficiency, public reports, overall statistics, etc. This information will be available through the website.

Note that multiple levels of interaction can be specified for each of this actors. For example, in the website, while the general public can only view public information, an employee can see other kinds of information, alerts for example, or energy consumption by department.

B.3 Contracting a Composite Service Solution

For convenience reasons we bring back the image from Section 1.4. Our company wishes to contract several cloud services that can be integrated to fulfill their EMES requirements. To achieve such a system several steps have to be preform as shown in Figure B.1.

The first step corresponds to the identification of the company needs. Those needs are the requirements for the system captured by analysing the company needs as we did in Section 3.1 and 3.2. Other specification could have been collected such as non-functional requirements, dependencies between components or metrics. However, they are not important for the scope of this use case.

The second step, is to compile all the information collected in the previous step and create an architecture view. It could be for example an UML component diagram or any other tool the company wishes. Something that formalizes the requirements information. In this step everything that could influence a decision towards

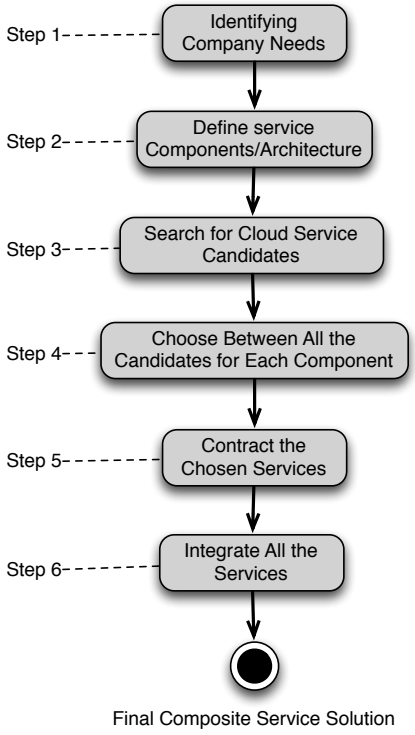


Figure B.1: Steps for contracting a composite service solution.

a specific service must be included. The purpose of this process is to gather and maintain the important information for later steps, this way the company never looses track of what they need, however the architecture can change with time, if no good solutions are found. This process is no different than the one used in software development.

In the third step the search for potential cloud services to fulfill the documented requirements starts. The company knows the importance of the EMES they want to implement and so it decides to allocate a team solely dedicated to this task. They search for all potential candidates without excluding any, except for those who do not fulfill the basic functional requirements. All those services who comply with these basic requirements are cataloged and all the relevant service information is collected and stored for later use in the decision process. This information is mainly extracted manually from the service provider website, making this entire task long and painful.

After collecting the information from enough sources, the fourth step begins. It is now time to reach a decision on which cloud service to use for each EMES component specified in step 2. Several meetings are scheduled with the team responsible for finding the services and the team responsible for implementing the EMES, even members of the administration are present for approval. All the cloud services are discussed and pros and cons are debated. Due to the complexity of the system to be implemented one meeting is not enough, several factors are critical, besides the price, availability and security are of the up most importance. The system must also

comply with the amount of data collected from the sensors, which is not an easy task.

After reaching a decision on which services to use, the board starts the contacts to contract the chosen services, this is the step 5.

Once the cloud services are contracted and running the final step belongs to the development team who has to integrate all the contracted services, and implement some parts of the system such has the website, or the mobile app. This step can be outsourced, however since the company is an IT specialized company they decide to do it themselves.

After a few months the EMES is finally up and running and the first report with the energy consumption and efficiency is sent to both the administration and the EU. The next months the company starts to reduce energy usage, and begins a marketing campaign in their new website for customers to prove their "green image".

B.4 Requirements

Because the smart building subject is in vogue, there are several examples ([62], [1], [63]) of systems similar to the one presented here, mostly for smart homes but that can easily be generalized to bigger buildings through the use, for example, of the cloud. Some use a client server approach like [119], which is interesting for a multi-building problem. More examples of different approaches for the smart buildings can be found in [33].

Since we are facing a building with high energy consumption and some special constraints from both cost reduction and EU directives, we require a different set of requirements from those specified in the above examples.

In this section we will list a series of requirements and give a brief explanation why they are really requirements.

B.4.1 Functional Requirements

The Functional requirements are a set of functions indispensable to the system. In our case the EMES functionality is going to be outsourced and so these requirements are strongly linked to the cloud components to be used later. Bellow we provide a list of the more relevant functional requirements :

- **Cloud Storage** - For an IT company with around 200 hundred employees, the building should be composed of several floors, each of them with several rooms full equipped with computers, monitors, etc. Even if the company has a smart approach for some resources like printers for example, this still means a lot of devices to monitor constantly, which obviously mean a huge amount of data. Since we are talking about a monitoring and efficiency system all the data must be stored, and this occupies a huge amount of storage capacity, capacity that the company currently does not possess;

- **Processing capability** - as explained in the previous requirement the amount of data is huge so, the system must have a processing capability to process

all the data, not only to read, but also to understand and produce new data, predictions and efficiency, or even suggestions for enhancing the energy results. This is a good example where the cloud can help;

- **Analysis capability** - The system should have a fair amount of tools to analyse and predict data, this collected or produced data can then be displayed to the end-user;

- **Manage incoming and outgoing messages** - The messages from and to the system should be managed properly. All the requests for data must be directed to a Webserver that decides what to do with them, and then sends the information. This way we ensure that both inside and outside people can access the data and control the system with proper privileges;

- **Ease of access** - since one of the company concerns is transparency, and ensuring they have a "green image", the most relevant reports and information should be available to everyone. A web-site or a mobile app can be introduced to ensure these goals by providing the information to general public;

- **Control** - It must be possible to send commands from multiple platforms, and devices in order to facilitate the system control for the company employees;

- **Report Management** - Since monthly reports to the UE must be sent, and for management reasons, the reports should be generated and sent automatically to the requesting entity or person. This way there is no need to waste time producing these reports manually, accelerating the process;

- **Alert System** - Because sometimes something might go wrong in the building (ex: someone left the lights on, or a specific device is consuming more energy then it should), the system should send alerts to someone in charge of supervising the system, in order to the problem can be dealt with as quickly as possible.

Note: above we have a list of high-level requirements and not development requirements.

B.4.2 Non-Functional Requirements

Basically the non-functional requirements are focused in scalability, we expect the web-site to be highly requested, availability, all the system should be running 24/7, and reliability, since some data is going to be predicted it must be accurate and based in accurate readings or else the reports would have no meaning.

Also important, but on another level, usability and maintainability, specially on the presentation module, it should be possible and easy to add new features as well as being easy to use by the end-user.

B.5 System Model

In this section we give an overview of the system model according to the specification given above. The system architecture is composed by three modules as shown by Figure B.2,the *Sensor Module*, the *Building Gateway* and the *Presentation Module*. While the first two can be perfectly transported to the cloud the *Presentation Module*, for physical reasons (sensors), cannot, and this is the reason why we decided to put it apart from the rest of the system. A more detailed explanation on this module is given in B.5.1.

The other two modules are perfectly easy to justify, the second module, the *Building Gateway*, is the backbone of the system, while the third module, *Presentation Module*, concerns about the presentation of information to the user. More detail about this two modules in B.5.2 and B.5.3.

B.5.1 Sensor Module

This is the only module that cannot be transported to the cloud, this happens because it is a highly physical part of the system. The sensors must interact directly with the devices to collect data.

The only requested function is to collect data about the consumption of each device and communicate it to a central wireless receiver that communicates with the database to store the data.

Since it is not the scope of this project to present a data collection system, or communication, we will not detail further this module.

B.5.2 Building Gateway

The so called core of the system, this module has three main concerns: the storage of the collected data in the database; the data processing to infer new pieces of data, and in the end send the result to the the end-user as requested.

To achieve this three goals we need the following components:

- **Web Server** - The responsibility of the Webserver is to act as a gateway. It manages all the communication with the outside applications that wish to view the available information, as well as control commands sent to the system, it should then process them and act as requested. Besides outside request, it also communicates with the database to retrieve information, or to store new one. The data retrieved from the database is then sent for processing if needed. This component controls the entire flow of the information in the system. Can obviously be distributed to more than one machine; Ex: Amazon Web Services [2], etc...

- **Data Base** - the database will receive data from the Sensor Module and will store it in a proper way to be retrieved by the Webserver. it will also sometimes

[2] `http://aws.amazon.com/`

receive already processed data directly from the Webserver.
Ex: Google Cloud Storage[3], etc...

- **Data Processing** - This unit acts only as a processing and data analysis station. His purpose is to get the data from the database and process it accordingly. The purpose for this component is obvious, as stated in B.4, the system requires a high processing capability, and so a dedicated unit is required to make up to this need. A SaaS can and should be used to analyse the data and make predictions.
Ex: Bime[4], etc...

B.5.3 Presentation Module

This module is responsible for presenting all the data to the end-user (building manager, employee, security, general public...), as well as provide the proper tools for controlling the system from both inside and outside the building. It is also responsible for the alerts to the end-user, this alerts can be customised. Many new applications can be performed in this module, the only constraint is to follow the communication protocol with the Webserver.

- **Web-Site** - The purpose for the web-site is to allow users to access all the data, probably with some constraints, but the goal is to open the collected data to anyone who wishes to see them. This goes in line with the company concerns stated in B.1, for the EU directives, and public image;

- **App** - The app acts both as a system controller an information viewer. It is in this app that the manager or anyone who was granted access can change energy consumption parameters, or simply analyse them. Could be available to all the building employees or general public, but can also be restricted to some, or to have different levels of usability, and information access. This app should also be multi-platform, and multi-device, for usability purposes.
Ex: Google App Engine[5], etc...

- **Mail Server** - Since a report on the energy activity is regularly created and can be sent to any entity by E-mail, it is important to have a mail server capable of doing so. Besides all the reports are stored in the server and it should be possible to review them later.

- **SMS Service** - Despite the fact that the system can send an sms to a specific mobile number alerting to some events previously configured, it is also possible to control the system through some special commands, given by a code in the sms, to activate a feature for example.
Ex: Clickatell[6], etc...

[3] https://cloud.google.com/products/cloud-storage
[4] http://bimeanalytics.com/
[5] https://developers.google.com/appengine/
[6] http://www.clickatell.com/

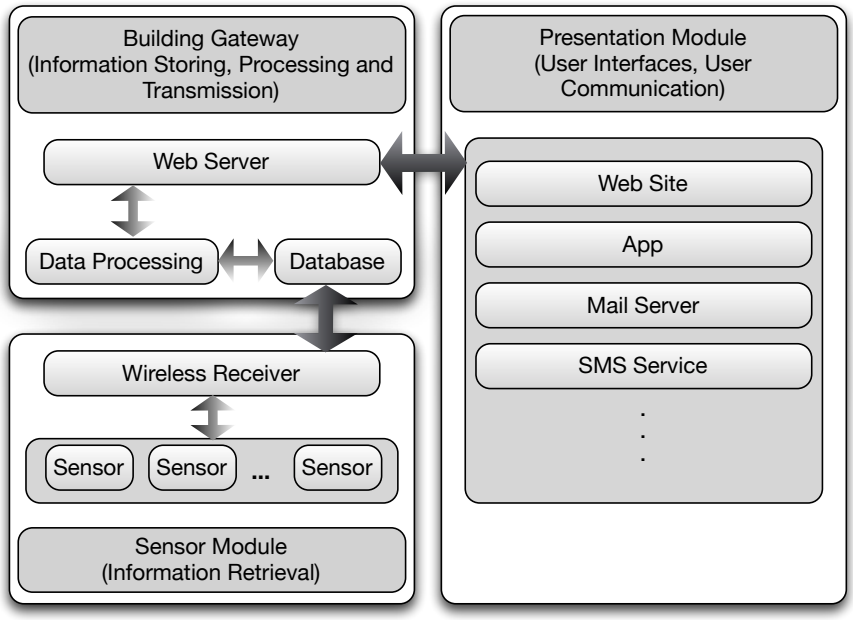


Figure B.2: EMES architecture model

C

Requirements List

This document lists all the CoudAid Prototype defined requirements according to the FURPS+ methodology [42] and prioritized with the MoSCoW method [28]. The lists are organized into two section functional requirements Section C.1 and non-functional requirements in Section C.2. This last section holds all the URPS+ as in Usability, Reliability, Performance, Supportability and the + for Design, Implementation and Interface.

C.1 Functional Requirements

The F in FURPS+ refers to the functional requirements which describes the overall functionality and capability of the system. Therefore, Table C.1 shows the full list of the CloudAid Application functional requirements. Note that some Requirements are specializations of others. This means that the "father", or top requirement, can be subdivided in smaller requirements. Note also that to facilitate the list comprehension the requirements are grouped according to the target application module. These modules were later used in the application architecture and are explained in Section 4.2 of the main document.

Table C.1: CloudAid: Functional Requirements List

ID	Name	Priority
CSA Data		
FR1	Add a ServiceTemplate to the CSA (Composite Service Architecture)	**MUST**
	Description: The user should be able to insert data about a ServiceTemplate and add it to the his Composite Service Architecture. These ServiceTemplates will be the building blocks of the aggregated solution.	
FR1.1	Add a Requirement to a ServiceTemplate	**MUST**

Continued on Next Page...

Table C.1 CloudAid: Functional Requirements List – Continued

ID	Name	Priority
	Description: The user should be able to insert requirements for a specific ServiceTemplate in order to focus the search.	
FR1.2	Add a Criterion to a ServiceTemplate	**MUST**
	Description: The user should be able to insert criteria for a specific ServiceTemplate. These Criteria are the characteristics to be evaluated and on which is based the decision process.	
FR2	Add a Requirement to the CSA	**MUST**
	Description: It should be possible to the user to insert general requirements that are not specific to a particular ServiceTemplate but apply to all the Composite Service Architecture. An overall price limit is a good example.	
FR3	Add a Criterion to the CSA	**MUST**
	Description: It should be possible to the user to insert general criteria that are not specific to a particular ServiceTemplate but will be used by all the Composite Service Architecture ServiceTemplates. The price characteristic is a good example.	
Service Set		
FR4	Crawl the Web for Linked USDL service descriptions	**WON'T**
	Description: The service set on which the search engine will be used should be built upon the crawl of the web for Linked USDL service descriptions. These Service description are provided directly by the service provider and published in the web for later retrieval. Due to the fact that Linked USDL is a new tool and is still under development there was no point in the development of this requirement for this version. However it should be considered in later versions.	
Search Engine		
FR5	Search for services that fulfil the ServiceTemplate Requirement list	**MUST**
	Description: The search mechanism should be able to return services that fulfill the user defined requirements list for a given ServiceTemplate. There can be more than one requirement and all should be fulfilled in order to the service to be considered an alternative and presented to the user.	
FR5.1	Search for services that do not have a certain functionality	**SHOULD**
	Description:	

Continued on Next Page...

Table C.1 CloudAid: Functional Requirements List – Continued

ID	Name	Priority
	The search mechanism should allow "negation searches". This is, allow the user to find services that do not have a certain functionality or a value for a specific characteristic. Ex: "I want Databases that are not MySQL".	
FR5.2	Search for services with a minimum value requirement	**SHOULD**
	Description: The search mechanism should allow the the search for requirements with a minimum value defined. Ex: "I want to store more that 500Gb of data."	
FR5.3	Search for services with a maximum value requirement	**SHOULD**
	Description: The search mechanism should allow the the search for requirements with a maximum value defined. Ex: "I want less than 1TB of data storage."	
FR5.4	Search for services with a specific value requirement	**SHOULD**
	Description: The search mechanism should allow the the search for requirements with a specific value defined. Ex: "I want MySQL databases".	
FR6	Search for services that fulfill price requirements	**MUST**
	Description: The search mechanism should allow the search for price requirements with a minimum, maximum or specific value defined. The price is usually the most common requirement while searching for a service and therefore this requirement is a priority.	
FR7	Get the service values for the defined Criteria	**MUST**
	Description: The system should be capable of extracting the service values for the user defined criteria. The Service features are stored in the Linked USDL Service Description and therefore they should be extracted and the needed values filtered for the Decision Process.	
FR8	Allow the usage of different units of measurements for the requirements	**COULD**
	Description: The requirements can be described in a different unit of measurement of the one described in the service (Ex: Database with 500GB, and the service set has a Database with 0.5TB). This difference should be invisible to the user.	
FR9	Read Linked USDL service descriptions	**MUST**
	Description:	

Continued on Next Page...

Table C.1 CloudAid: Functional Requirements List – Continued

ID	Name	Priority
	Since the Services are described in Linked USDL, the system should be able to read this format and convert its data for all the necessary decision related calculations.	
FR9.1	Extract the Service Qualitative & Quantitative Values	**MUST**
	Description: The system should be able to extract from the service description the Qualitative and Quantitative Service Properties. These properties are the service features, that will either be related to a requirement or a criterion for the decision process.	
FR9.2	Calculate the service price when defined with the price:hasPrice property	**MUST**
	Description: The system should, from the Linked USDL description, be able to automatically calculate the service price define with the Linked USDL property: hasPrice. This Price should be stored in the Service Data.	
FR9.3	Calculate the service price when defined with the price:hasPriceFunction property	**WON'T**
	Description: The system should, from the Linked USDL description, be able to automatically calculate the service price defined with the Linked USDL property: hasPriceFunction. This is the Linked USDL option for describing dynamic pricing. Due to its complexity it was decided to leave this requirement for later versions since it would only delay the application prototype with no major benefits for the immediate prototype objective: to provide a workable proof of concept.	
FR10	Group Similar Alternatives	**WON'T**
	Description: The system should be capable of grouping alternatives that have the same values for the defined criteria and only vary in non-criteria characteristics. These alternatives should be considered as a single alternative and be dealt as such. Since these requirement would affect mainly the ease of use for the user allowing the reduction of user interaction, it was considered of less importance for the immediate prototype objective: to provide a workable proof of concept. However, it should be considered in later version.	
FR10.1	Use a indifference threshold	**WON'T**
	Description:	

Continued on Next Page...

Table C.1 CloudAid: Functional Requirements List – Continued

ID	Name	Priority
	The system could allow the user to define a indifference threshold that would allow the filtering and grouping of similar alternatives. This requirement was viewed as less important for the immediate prototype objective: to provide a workable proof of concept. Therefore its development was postponed to a later version.	
Decision Engine		
FR11	Import/Export XMCDA decision data	**MUST**
	Description: The decision module should be capable to export the decision problem to XMCDA [38] and import the decision results from XMCDA. XMCDA is a standard for Decision Methods data transactions and therefore has an important role in requirement SR2.	
FR12	Rank the services according to their decision value	**MUST**
	Description: The alternatives for each ServiceTemplate should be ranked according to their decision value. This will produce an ordered list of alternatives based on the Decision Method result.	
FR13	Normalization of Decision characteristics	**MUST**
	Description: The system should be able to normalize the characteristics of the service defined defined as criteria in order to weight and decide upon them.	
FR13.1	Normalization of Decision Numerical characteristics	**MUST**
	Description: The system should be able to normalize numerical characteristics. Eg: 500Gb, 2CPUs, etc.	
FR13.2	Normalization of Decision Non-Numerical characteristics	**MUST**
	Description: The system should be able to normalize non-numerical characteristics. Eg: High replication is defined as being the best option (preference), if a service has the value "Low Replication", the user should define a distance to the best option.	
FR13.3	Normalization of Decision Binary characteristics	**MUST**
	Description: The system should be able to normalize binary characteristics. Eg: SSH/NO SSH, YES/NO, etc.	
FR13.4	Normalization based on User preferences	**SHOULD**
	Description:	

Continued on Next Page...

Table C.1 CloudAid: Functional Requirements List – Continued

ID	Name	Priority
	The normalization should be done based on a preference introduced by the user. If no preference is defined the direction of the preference should be defined, maximizing or minimizing the criterion.	
Aggregation Engine		
FR14	Generate Aggregated alternatives	**MUST**
	Description: The system should generate aggregated solutions composed of at least one alternative for each existent ServiceTemplate. The aggregated solution is the best combination of alternatives for the user Composite Service Architecture based on the user defined data (requirements, criteria, weights, preferences).	
FR15	Calculate the admissible aggregated solutions	**MUST**
	Description: The system should be capable of deciding which aggregated solutions are admissible or not. The admissibility of an aggregated solution is based on the Composite Service Architecture user requirements and intrinsic alternative restrictions. These alternative restrictions are already defined in the Service Description and are not controlled by the user, but should be respected when aggregating different alternatives. Eg: A service that only allows the import of data from Oracle formats cannot work with a PostgreSQL database.	
FR15.1	Calculate the admissible aggregated solutions based on price requirements	**MUST**
	Description: The system should be capable of deciding which aggregated solution meet the overall price requirements. Eg: "I don't want to pay more than €200 for my aggregated solution".	
FR15.2	Calculate the admissible aggregated solutions based on service restrictions	**COULD**
	Description: The application should be capable of deciding which aggregated solution meet the intrinsic alternative restrictions.	
FR16	Decide which is the best admissible solution from the admissible solutions list	**MUST**
	Description: The system should be capable of deciding which is the best admissible solution if more than one exists. This decision should be based in the user data. In the end only one aggregated solution is presented to the user, if any is found.	

C.2 Non-Functional Requirements

The rest of FURPS+ specification describes non-functional requirements. Non-functional requirements deal with system properties rather than functions performed by the system. Usability, Reliability, Performance and Suportability are the categories described by FURPS. The "+" in FURPS+ refers to four extra categories: Design, Implementation, Interface and Physical. These requirements are usually referenced as system constraints. The following sections list the requirements for each of the FURPS+ specified categories.

C.2.1 Usability Requirements

Usability requirements relate to user interface aesthetics and data presentation to the user. Although this is not a priority as already stated, some concerns are still considered and are expressed in Table C.2.

Table C.2: CloudAid: Usability Requirements List

ID	Name	Priority
UR1	Adapt the Decision Method to the User Preferences	**MUST**
	Description: The decision method used is based on the user information. This information should be collected through a set of questions.	
UR2	Show the CSA information to the user	**MUST**
	Description: The user interface should show the Composite Service Architecture information inserted by the user. i.e: Requirements, Criteria, Service Templates, etc.	
UR3	Show the Search results for a ServiceTemplate to the user	**MUST**
	Description: The user interface should show each ServiceTemplate search results based on the defined requirements.	
UR4	Show the Aggregation results to the user	**MUST**
	Description: The user interface should show the final aggregation solution to the user.	
UR5	Understandable sentences	**MUST**
	Description: The system should interact with the user in a easy to understand language without ambiguity.	
UR6	Prompt user for extra information	**SHOULD**
	Description: The system should ask the user for extra information if the decision method in use requires it or if at any point required by the system.	

Continued on Next Page...

Table C.2 CloudAid: Usability Requirements List – Continued

ID	Name	Priority
UR7	Error Messages and Flow of execution status	**SHOULD**
	Description: The system should show understandable error messages and explain what should be changed in order to progress.	
UR8	Allow to change requirements if no service is found but keep previous decisions	**WON'T**
	Description: The system should be able to keep the state if no service is found for the user defined requirements. The user should be prompted for changing its data. Since this requirement would only improve the user experience and has no major impact in the overall prototype concept it was postponed to further versions development.	

C.2.2 Reliability Requirements

Focused in Availability, system accuracy and Failover mechanisms, Reliability Requirements were used to group all the CloudAid application calculation accuracy related requirements. The accuracy of the system data is of great importance for it to properly present a final solution to the user, hence, all the calculations should, at all time, be correct. Table C.3 lists all the reliability requirements.

Table C.3: CloudAid: Reliability Requirements List

ID	Name	Priority
RR1	Correct normalization of Service Template Weights	**MUST**
	Description: The ServiceTemplate weights, if used, should be correctly normalized at all times. This normalization allows the consistency and comparability of the decision data.	
RR2	Correct normalization of Criteria weights	**MUST**
	Description: The Criteria weights, if used, should be correctly normalized at all times. This normalization allows the consistency and comparability of the decision data.	
RR3	Correct Decision Data Export/Import to XMCDA	**MUST**
	Description: The decision data should be correctly exported and imported to and from the XMCDA standard. The decision methods will read the data exported to XMCDA and therefore, if any misleading conversion occurs the decision process is corrupted. The same happens with the decision results in the opposite direction.	

Continued on Next Page...

Table C.3 CloudAid: Reliability Requirements List – Continued

ID	Name	Priority
RR4	Correct service price calculation	**MUST**
	Description: The Service Price information should be correctly extracted from the Linked USDL service description and then correctly calculated. Any incorrect conversion or calculation will corrupt the search and later the decision process.	
RR5	Correct decision on the Admissible solutions algorithm	**MUST**
	Description: The system should correctly test and return the admissible solutions.	
RR6	Correct calculation of the overall admissible solution decision value	**MUST**
	Description: The admissible solutions decision value should be correctly calculated based on the ServiceTemplates weights which are defined either directly by the user or through a decision method.	
RR7	Correct mapping of the Linked USDL service description	**COULD**
	Description: All the Linked USDL concepts should be correctly mapped to the "in-memory" service description.	
RR8	Correct normalization of Decision characterisitcs	**MUST**
	Description: The decision characteristics of all types (numerica, bynary and non-numerical), should be correct at all times. This normalization allows the consistency and comparability of the decision data.	

C.2.3 Performance Requirements

Performance requirements capture the system metrics such as throughput, response time, etc. These are not a top priority, mainly because of the project research scope. As previously stated, the goal is to prove that the concept is valid and not to develop an high performance application. Nevertheless a few metrics are of interest and are listed in Table C.4.

Table C.4: CloudAid: Performance Requirements List

ID	Name	Priority
PR1	Admissible Algorithm performance with high amount of alternatives	**SHOULD**

Continued on Next Page...

Table C.4 CloudAid: Performance Requirements List – Continued

ID	Name	Priority
	Description: The algorithm for admissible solutions decision should be able to deal with high amount of combinations since we can have many ServiceTemplates and many alternatives for each of them. This requirement will serve as a metric to test the algorithm but no minimum value is pre-defined	
PR2	Allow the search of a high number of service descriptions	**SHOULD**
	Description: The system should be capable of searching high amounts of services descriptions in a few seconds time, around 3 to 5 seconds.	

C.2.4 Supportability Requirements

The suportability requirements category concerns about testability, modifiability and extensibility among other characteristics. These three characteristics however, are quite important in the CloudAid application, mostly because this is a prototype research application hence, the possibility of future development is high enough to turn extensibility/modifiability into one big concern. Moreover the research oriented perspective makes the testability characteristic also of great importance. It should be possible to test and prove that the achieved results are valid. Table C.5 lists the Suportability Requirements.

Table C.5: CloudAid: Supportability Requirements List

ID	Name	Priority
SR1	Testable System	**MUST**
	Description: It should be possible to test the different modules of the system. It should be possible to test each and every requirement in the requirements list.	
SR2	Allow the use of different Decision Methods	**SHOULD**
	Description: It should be possible to change the Decision Method if wanted or needed. The system should easily accept the use of new decision methods. The idea is to provide to other researchers the capability to use the environment and test their own methods.	
SR3	Allow the ease of functionality extension	**MUST**
	Description:	

Continued on Next Page...

Table C.5 CloudAid: Supportability Requirements List – Continued

ID	Name	Priority
	The system should allow the easy extension or modification of functionalities in each of the independent modules. This makes possible the future development of further or more complex functionalities. We should not forget the research perspective of the application.	

C.2.5 Design Requirements

Also named as Design constraints, these requirements give an overview of what are the design time requirements, usually data models or databases if needed. The CloudAid application has some special data needs as Table C.6 shows.

Table C.6: CloudAid: Design Requirements List

ID	Name	Priority
DR1	CSA Data Model	**MUST**
	Description: It should be defined a data structure to store the ServiceTemplates, Requirements and Criteria defined by the user.	
DR2	Use of a Triple Store	**COULD**
	Description: If possible a triple store can be used for storing the Service set data. This Triple store should be considered for large scale storage.	
DR3	Use of an external Cloud Concepts Ontology	**SHOULD**
	Description: In order to facilitate the user requirements analysis by the system and later the search engine capability the use of an external ontology with cloud concepts is important. These concepts should act as "tags" for the search engine.	
DR3.1	Import a list of concepts from the CloudTaxonomy	**WON'T**
	Description: Since the external ontology can be modified, it should be important to ensure that the system copes the this modifications and imports all the new concepts when needed without changing is normal execution.	

C.2.6 Implementation Requirements

The implementation requirements define system implementation constraints such as standards to use, architectural constraints or coding constraints for the system development. As stated before, there were no big implementation constraints, since we had full freedom in the prototype development. However, it was decided, for sake of Suportability, to use the MVC model. Note that besides requirement IMPR1 the other are related to previous requirements. Table C.7 shows the Implementation Requirements list.

Table C.7: CloudAid: Implementation Requirements List

ID	Name	Priority
IMPR1	MVC Architecture	**MUST**
	Description: The System should implement the MVC model for easier modularity and later extensibility/modifiability.	
IMPR2	Modularity capability	**MUST**
	Description: All the modules (UI, Search, Decision, Aggregation), should be extensible, if new functionalities are to be added, without the need for changing the others. This requirement is strongly linked to requirement SR3	
IMPR3	Interface for Decision Methods	**WON'T**
	Description: All the Decision method should implement an interface that would facilitate the inclusion of new decision methods. This would benefit the SR1 requirement. However since a deeper study of how and if it is possible to implement such general interface capable of wrapping all the decision methods is mandatory, it was decided to postpone the development for later versions.	

C.2.7 Interface Requirements

Interface requirements specify constraints about external items with which the system has to interact. In our case two items are of the upmost importance: Linked USDL, which is used to describe the services and XMCDA [38], a standard for describing decision problems. Table C.8 shows the requirements for the two above external interfaces.

Table C.8: CloudAid: Interface Requirements List

ID	Name	Priority
INTR1	Service Set based on Linked USDL service Descriptions	**MUST**
	Description: The services are described in Linked USDL, an RDF model.Therefore it should be possible to deal with RDF files.	
INTR2	Decision problem expressed in XMCDA	**MUST**
	Description: The decision problem should be described in the XMCDA (which is a XML Schema) standard for SR1 purposes and the results should also be imported from XMCDA.	
INTR3	Service information compliant with Linked USDL	**SHOULD**
	Description: The service data should be stored and managed in a similar way to the Linked USDL service description.	

D

Model-View-Controller Overview

As stated in section 4.1.3 one of the main objectives was the modularity of the prototype and the subsequent independence of its modules. With this in mind a decision for using the Model-View-Controller Model (MVC Model) [70] was made.

The MVC Model as the name implies is divided in three major components which are depicted in [70]:

- **Model** - "is the domain-specific software simulation or implementation of the application's central structure.". These are usually referenced to as the data model, that can be a database or objects in memory.
- **View** - "views deal with everything graphical; they request data from their model, and display the data. They contain not only the components needed for displaying but can also contain subviews and be contained within superviews.". Usually reference as user interfaces.
- **Controller** - "Controllers contain the interface between their associated models and views and the input devices (keyboard, pointing device, time). Controllers also deal with scheduling interactions with other view-controller pairs". It is the connector between model and views, responsible for the flow of execution.

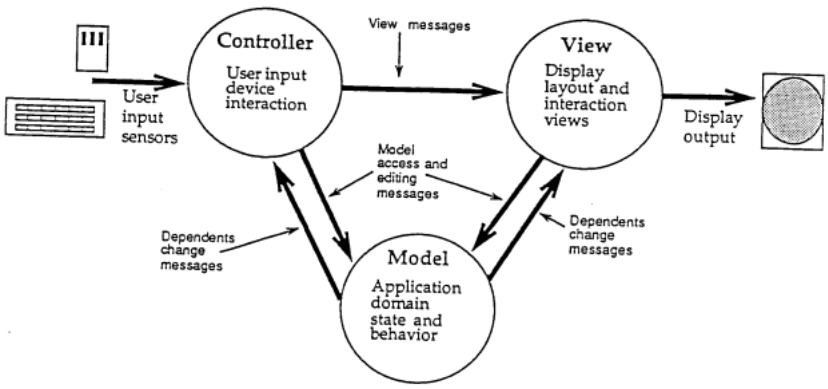


Figure D.1: Model View Controller State and Message Sending. From [70]

Using this model we were able to detach the user interfaces from the application logic and the service data. Going back to the objectives from Section 4.1.3, we were able to focus on the application development without any concerns with the usability issues, that could be developed later on if wanted. Also the modularity was achieved since each Model and Controller could be developed independently from the others.

E

Simulation Scenarios

In order to facilitate the testing phase several simulation scenarios were built. This document presents the used simulation scenarions for the CloudAid Prototype testing. These try to exemplify possible usage scenarios for the application. The obvious scenario is the Use Case defined in Appendix B. However, due to its complexity some testes were easier to execute with less amounts of information as is the case of the Heroku Scenario. These less complex scenarios aim at validating data calculations (reliability) or some functional requirements such as the Search module related requirements. Bellow is presented the full specification of the used scenarios.

Each scenario is composed of a CSA Data structure with Service Templates, Requirements and Criteria. Although all the bellow scenarios were used as simulations, there is no difference in the application execution and results if we manually define the same set of data. Hence, these scenarios simulate the user interaction with the system.

UseCase Scenario: The UseCase Scenario represents a real world problem to be solved by the CloudAid application. It is the most complex of the four scenarios used in the testing phase. However, it is the most important because it brings together all the concepts discussed throughout this thesis and applies them to our real world use case.

The CSA Data is composed of a list of Service Templates, each with their own requirements and criteria, a list or global requirements and a list of global criteria:

- **Service Templates**:
  - Database
    - Requirements:
      - "StorageCapacity > 200Tb"
      - "Must have some Encryption feature"
      - "Location = 'EU'"
      - "Must not have the 'MySQL' platform"
      - "DataTransferOUT per month > 200Tb"
    - Criteria:

            - StorageCapacity
            - DataTransferOUT
    - Data Analytics
        - Requirements:
            - "Must have 24/7 support feature"
            - "Must have SSL feature"
            - "Must have some Backup feature"
            - "API calls per minute > 2000000"
        - Criteria:
            - API Calls
    - WebServer
        - Requirements:
            - "Must have the 'Apache' platform"
            - "Must have 'autoscalability' feature"
            - "Location = 'EU'"
            - "CPU cores > 4"
            - "Must have some network performance metric"
            - "StorageCapacity > 500Gb"
            - "MainMemory < 16Gb"
            - "Must support 'PHP' language"
            - "Must be a Unix machine"
        - Criteria:
            - CPU Cores
            - Performance (network)
            - StorageCapacity
            - MainMemory
    - App Platform
        - Requirements:
            - "Must have the 'Apache' platform"
            - "Must have 'autoscalability' feature"
            - "Location = 'EU'"
            - "CPU cores > 8"
            - "Must have some network performance metric"
            - "StorageCapacity > 500Gb"
            - "MainMemory < 8Gb"
            - "Must support 'Ruby' language"
        - Criteria:
            - CPU Cores
            - Performance (network)

            - StorageCapacity
            - MainMemory
    - Mail Service
        - Requirements:
            - "Must send 'E-mail' messages"
            - "Must use 'SMTP' protocol"
            - "Messages per month > 5000"
            - "Number of dedicated IPs > 1"
            - "Send files > 100Mb"
        - Criteria:
            - Number of Messages
    - SMS Service
        - Requirements:
            - "Must send 'SMS' messages"
            - "Messages per month > 1000"
            - "Number of Users > 4"
        - Criteria:
            - Number of Messages

- **Global Requirements**:
    - "Price < €300"
    - "Availability > 99.9" (All Service Templates must fulfill these requirement)
- **Global Criteria**:
    - Price
    - Availability

Heroku Scenario The Heroku Scenario was the first to be used. This scenario was design for low complexity but still reliable enough for the prototype testing. Hence, although it has only a few data, it is based on a real case and real service Heroku databases [1]. The objective was to be able to manually calculate all the data and results in order to validate the prototype calculations and results. The example is based in a potential cloud solution that requires 3 storage services with different needs.

This is the scenario used in all reliability tests (except TRR8), and most of the functionality tests has well. The CSA Data:

- **Service Templates**:
    - Database 1

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

        - Requirements:
            - "StorageCapacity > 500Gb"
            - "Must have some Backup feature"
            - "Must not have the 'MySQL' platform"
            - "Must support the 'PostegreSQL' platform"
        - Criteria:
            - StorageCapacity
    - Database 2
        - Requirements:
            - "StorageCapacity > 500Gb"
            - "Availability > 99%"
            - "CacheSize > 16Gb"
        - Criteria:
            - StorageCapacity
            - CacheSize
    - Database 3
        - Requirements:
            - "CacheSize > 5Gb"
        - Criteria:
            - CacheSize

- **Global Requirements**:
    - "Price < €4000"
- **Global Criteria**:
    - Price

Normalization Scenario This scenario was designed with the sole purpose of testing the normalization of the decision characteristics (TRR8). A decision characteristic can be numerical, non-numerical or binary, this scenario has at least one of each type in order to test the normalization process. The CSA Data:

- **Service Templates**:
    - Server
        - Requirements:
            - "StorageCapacity > 150Gb"
            - "DataOutExternal > 40Tb"
            - "Must have the 'Apache' platform"
            - "Must be locates in 'EU'"
            - "Must have '64bit' CPU architecture"
            - "Must support 'Java' language"

        - "Availability > 99.9%"
        - "Must have some network performance metric"
    - Criteria:
        - StorageCapacity (Numerical characteristic)
        - DataOutExternal (Numerical characteristic)
        - Performance (Non-Numerical characteristic, can assume "Low", "Medium", "High" or "Very High")
        - StorageType (Binary characteristic, either "HardDisk" or "SSD")

- **Global Requirements**:
    - "Price < €5000"
- **Global Criteria**:
    - Price (Numerical characteristic)

Requirements Scenario The Requirements scenario was designed for testing the SearchEngine module execution time with different number of requirements. It was created with a total of 7 Service Templates each with an increasing amount of Requirements but no criteria (Only for testing the search mechanism). There is no point in fully describing all the requirements in this scenario since their data is not important, different requirements could have been defined without major effects in the performance test to be performed.

F

Cloud Taxonomy

This document holds the full list of concepts present in the Cloud Taxonomy as well as its description.

F.1 Top Level Concepts

There are three top level concepts in the Cloud Taxonomy as Figure F.1 shows. These concepts wrap completely different things. While Deployment model and Service Model describe the service as a whole the Property on the other sides describes the service features or characteristics:

- **Deployment Model** - Describes what type of cloud is the service targeted for: Private, Public, Hybrid, Community cloud as described in [84] by the NIST.
- **Property** - A service property, wraps all the cloud service properties.
- **Service Model** - Describes the cloud service models as defined in [84] by the NIST: IaaS, PaaS, SaaS and added the BPaaS (Business Process as a Service) from the mOSAIC ontology [89].

F.2 Property

The most important class. Groups all the service characteristics or features also called Service Properties. In Figure F.2 we can see the 5 different categories of properties:

- **Feature** - Describes all the service features that do not belong in any of the other categories but are still features or service properties. Mainly used for SaaS since they have the highest degree of differentiation which makes hard to clearly state which type of service property the SaaS feature is related to.
- **Functional Property** - Groups all the cloud service functional properties. See Section F.2.1.

Figure F.1: Cloud Taxonomy: Top Level

- **Interface** - Groups all the types of interfaces or points of interaction between actors and the Cloud Service. See Section F.2.2.
- **Non-Functional Property** - Groups all the cloud service non-functional properties (Eg: availability, scalability,...). See Section F.2.3.
- **SupportProperties** - Groups all the cloud service support properties or types of support available in the service. See Section F.2.4.

Figure F.2: Cloud Taxonomy: Property

F.2.1 FunctionalProperty

A functional property of a service is a specific feature or resource provided by the service. These are the most important properties in a service as they define what exactly the service offers or does. In order to distinguish all the functional properties it was decided to group them according to their usage in different Service Model. Therefore, for example all the IaaS related properties such as CPU Speed or Operating system are grouped under the *ComputingFuntionalProperty* class. This decision was solely for organization process, nothing obliges a IaaS service to use all the properties defined in the *Computing Functional Property* category and nothing prevents the same service to use properties of any other category. It should be normal for a cloud service to use properties of several different categories. Figure F.3 shows the list of functional property categories:

- **Computing Functional Property** - Groups all the computational (IaaS) resources related characteristics.

Figure F.3: Cloud Taxonomy: Functional Properties

- **Data Functional Property** - Groups all the Storage as a Service and data transfer related characteristics.
- **Encryption** - Describes cloud service encryption characteristics.
- **License** - Describes cloud service license types. Typically can be open source or proprietary.
- **Location** - Describes location characteristics, for example region of access or hosting location.
- **Monitoring** - Describes the available service monitoring related characteristics.
- **Network Functional Property** - Groups all the network related characteristics.
- **Platform** - Describes any kind of platforms used or allowed by the service. Eg: Oracle, MySQL, or Apache.
- **Platform Functional Property** - Groups all the Platform as a Service and Software as a Service related characteristics.
- **Replication** - Describes the service or its data replication characteristics.

F.2.1.1 Computing Functional Property

The computing functional properties are all the IaaS related characteristics and are typically related to computing resources as CPU, memory or operating systems used by the instance to be contracted. Good examples of services that use this characteristics are Amazon EC2, Google App engine, or GoGrid. Figure F.4 show the tree of characteristics:

- **CPU Architecture** - CPU architecture type. Eg: 32 or 64 bits.
- **CPU Cores** - for number of CPU cores related features.
- **CPU Flops** - for CPU FLOPS related features
- **CPU Speed** - for CPU speed related features. Eg: 1.2Ghz.

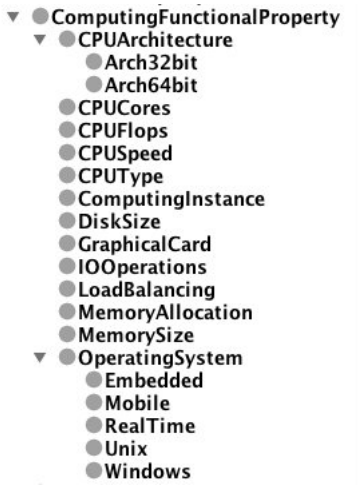


Figure F.4: Cloud Taxonomy: Computing Properties

- **CPU Type** - for specific types of processor. Eg: Xeon, AMD, or other type of information.
- **Computing Instance** - for features that perform as computing instances but are described by the service provider as a single unit. Eg: Worker Daemons that process data.
- **DiskSize** - for hard disk storage size related characteristics.
- **GraphicalCard** - for graphical processing resources or features.
- **IO Operations** - for Input/Output operations related characteristics. Could be the number of operations per second or any other kind of feature related to I/O operations.
- **Load Balancing** - for load balancing features allowed by the service. Could be capacity or number of load balancers or any other features provided by the cloud service.
- **Memory Allocation** - for memory allocation policies, algorithms or strategies.
- **Memory Size** - for main memory or RAM related characteristics.
- **Operating System** - Groups all the Operating system related characteristics. It describes which are provided or supported by the cloud service. They can be of the types: Embedded, Mobile, Real Time, Unix or Windows.

F.2.1.2 Data Functional Property

Mainly composed of database characteristics the *DataFunctionalProperty* class groups all the data related characteristics including data storage, transfer and processing but also database requests or backup and redundancy features. We can see the full list of properties in Figure F.5 as listed bellow:

- **Backup & Recovery** - for backup policies and recovery features.
- **Cache Size** - for the cache size provided by the service.
- **Data IN External** - for the amount of data transferred from external sources to the service in question.
- **Data OUT External** - for the amount of data transferred from the service to outside locations.
- **Data IN Internal** - for the amount of data transferred from other service in the same cloud to the service in question.
- **Data OUT Internal** - for the amount of data transferred from the service to other service in the same cloud.
- **Data Processed** - for the amount of data processed by the service.
- **Data Request** - groups all the database related requests:
  - COPY Request
  - DELETE Request
  - GET Request
  - LIST Request
  - POST Request
  - PUT Request
  - Read
  - Write
- **File Size** - for characteristics related to the file size allowed either for transferring or to storage or any other purpose.
- **Queries** - for any characteristics related to database queries. Can be either query amounts or specific query types allowed.
- **Records** - for any characteristics related to number of records, rows, or specific constraints or features to data elements.
- **Redundancy** - for describing redundancy policies or strategies.
- **Storage Capacity** - for storage resources amount capacity. The amount of storage space.
- **Storage Type** - for describing the type of storage used. Eg: SSD, HardDisk,...
- **Transactions** - for transactional databases or transaction policies related features.

Figure F.5: Cloud Taxonomy: Data Properties

F.2.1.3 Network Functional Property

The *NetworkFunctionalProperty* class groups the network characteristics of the service. They are usually related to IP address or network metrics such as latency or bandwidth. Figure F.6 shows the tree of network characteristics:

- **IP Address** - for any characteristics related to IP addresses. Eg: IPv4 or IPv6.
- **Network Delay** - for network delay characteristics. Eg: 200ms.
- **Network Internal Bandwidth** - for characteristics related to network bandwidth inside the provider cloud.
- **Network Latency** - for network latency characteristics. Eg: Low latency, 200ms...
- **Network Protocol** - for describing the network protocols used or allowed by the service.
- **Network Public Bandwidth** - for characteristics related to network bandwidth for public domains.
- **Network Request** - Groups all types of requests and any characteristics related to them.
- **Transfer Rate** - for any features about transfer rates or transfer speeds. Could also describe limit amounts of transfer rates.

F.2.1.4 Platform Functional Property

The platform functional properties are those properties usually seen in SaaS and PaaS services. Characteristics such as the programming language or the amount of

Figure F.6: Cloud Taxonomy: Network Properties

Figure F.7: Cloud Taxonomy: Platform Properties

users allowed to access are grouped under this category. The fact that SaaS offers are highly differentiated from one another makes it difficult to wrap their characteristics. However, for such situation the *Feature* class can be used. In Figure F.7 we can see the tree for the *PlatformFunctionalProperty* Class

- **API Calls** - for the amount of API calls or accesses to the service.
- **Applications** - for describing any constraints or features about the amount of application allowed.
- **Language** - for specifying programming languages supported by the service.
- **Messages** - Groups all the message related characteristics. These messages can be of a certain type (Eg: E-mail, SMS, etc) and use a certain protocol (HTTP, SOAP, SMTP, etc).
  - Message Number
  - Message Protocol
  - Message Type
- **Users** - for the amount o users or any restrictions to the user with granted access to the service.
- **Websites** - for services that deal with web hosting or any website activity. Eg: Number of hosted websites allowed.

F.2.2 Interface

The *Interface* class can be used to describe any type of interaction point between the cloud service and the consumer. Although the major interfaces have been describes as shown in Figure F.8 other types can be wrapped by the top class *Interface*.

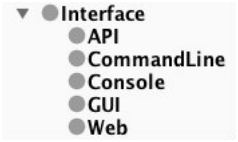


Figure F.8: Cloud Taxonomy: Interfaces

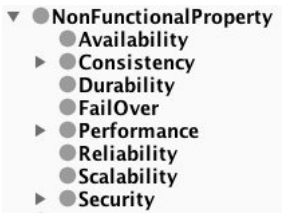


Figure F.9: Cloud Taxonomy: Non-Functional Properties

- **API** - for API like interfaces.
- **Command Line** - for interfaces that use a command line to issue operations.
- **Console** - for services that provide a special console for user interaction or management.
- **GUI** - for services that provide a graphical user interface.
- **Web** - for services that use a web interface or website for user interaction or management.

F.2.3 NonFunctionalProperty

This class groups all the non-functional properties of the services. These characteristics are usually related to performance or quality of service. Figure F.9 shows the tree of categories for the non-functional properties:

- **Availability** - for availability related features or characteristics.
- **Consistency** - for consistency policies or other related features. Can describe degrees of data or results consistency.
- **Durability** - for features that deal with persistence of data.
- **Fail Over** - for fail mechanisms or policies implemented or used in case of any failure.
- **Performance** - for any metric or characteristic of the service that describes its performance. Eg: High Network Bandwidth, requests processed per second.
- **Reliability** - for describing features provided by the service to ensure the process or data reliability.
- **Scalability** - for describing any policies or scalability strategies.
- **Security** - for security policies or protocols used or allowed.

Figure F.10: Cloud Taxonomy: Support Properties

F.2.4 SupportProperties

From the services we analysed we found out that support was a common feature in almost all the cloud services. Hence, this class tries to capture the most common types of support features. Figure F.10 shows the sub categories of the *SupportProperties* class:

- **Developer Centre** - for services that provide some development knowledge base.
- **Forum** - for services that have a specific forum dedicated to topics related to the cloud service in question.
- **Manual** - for services that provide any kind of manual for its usage.
- **Support Team** - for services that provide a specialized support team to its users.
- **Support 24x7** - for services that provide 24 hours a day 7 days a week support of any kind.
- **Videos** - for services that provide videos either of tutorials or other kind of support to their users.

G

Application Example

This Chapter aims at presenting an example of the CloudAid Prototype execution for the Use Case defined in Appendix B.

In this example it is used the data in the Use Case Scenario depicted in Appendix E. In order to execute the CloudAid Prototype already with this scenario one can use the command in Listing G.1 from the CloudAid installation folder. The first argument specifies the scenario to use, in this case the use case. The second argument the Decision Method to be used, SAW. The third specifies the type of Admissible Solutions Algorithm to use, no incomparability support was the one used in this example.

Listing G.1: Command to run the CloudAid Prototype with this example setup

```
1 java -jar CloudAid.jar usecase saw noinc
```

Note also that it was used the partial service set which is available in the CloudAid Public repository [7].

Once the application is started the first step is to load the service set. It loads all the turtle or RDF/XML files in the *Services* folder. Figure G.1 shows the files loaded as well as the status for the components being initialized. From the partial service set there are 13 service descriptions each of them with dozens of service offering.

After initializing the service set and all the application components (Search, Decision and Aggregation Engines), in a normal execution the user would be prompted for information about the CSA he wishes to build. However, this example runs on the Use Case Scenario, thus all the data is already automatically inserted (see Appendix B for the Use Case CSA data).

Skipping the CSA data insertion by the user, the next step is the CSA evaluation (CSAEvaluator module). It may happen that the user is required to insert extra information. Since we are using the SAW (Simple Additive Weighting) method there is the need to specify weights for the criteria defined. Figure G.2 shows these questions. For each criterion defined in the CSA the user has to insert a numerical value for its weight and the preference direction. There are a total of 14 criteriain this example, and one is global: the price. The Price is also the only criterion which

```
SYSTEM: Initializing CloudAid Components...
STATUS: ----Initializing ServiceSet
SYSTEM: Loading ServiceSet...
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_medium_c1_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_medium_c1_1.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_xlarge_c1_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_xlarge_c1_1.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_xlarge_m1_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_xlarge_m1_1.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_xlarge_m2_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonEC2_xlarge_m2_1.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonS3_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_AmazonS3_1.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_Analytics_service_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_Heroku_Addons_Mail_Service_0.ttl
SYSTEM: Reading file: ./Services/Partial_5625/Service_SMS_Service_0.ttl
SYSTEM: Total triples in ServiceSet: 610105
SYSTEM: Successfully loaded 13 Service Descriptions.
STATUS: ----Initializing DecisionEngine
STATUS: ----Initializing AggregationEngine
```

Figure G.1: CloudAid Example: Service Set Startup

```
STATUS: Evaluating CSA Data
Please specify the decision weight of criterion: price in Service Template: General
2
Do you want to maximize the value of criterion: price in Service Template: General? (y/n)
n
Please specify the decision weight of criterion: StorageCapacity in Service Template: database
3
Do you want to maximize the value of criterion: StorageCapacity in Service Template: database? (y/n)
y
```

Figure G.2: CloudAid Example: Inserting criteria weights (SAW method)

has a minimum preference direction as shown in Figure G.2. This means that the smaller the value of the price the better. With a total of 14 criteria means that there are 28 questions to answer, 14 with the weight values and 14 with the preference directions. Table G.1 lists the values inserted in this example.

Table G.1: Use Case Example: Service Templates Criteria Weights

Service Template	Criterion	Weight
General	Price	2
Database	StorageCapacity	3
Database	DataOutExternal	5
Data Analytics	APICalls	4
WebServer	CPUCores	1.5
WebServer	Performance	4
WebServer	StorageCapacity	1
WebServer	MemorySize	1.3
App	CPUCores	2.5
App	Performance	2
App	StorageCapacity	1
App	MemorySize	1.3

Continued on Next Page...

Table G.1 – Continued

Service Template	Criterion	Weight
Mail Service	MessageNumber	4
SMS Service	MessageNumber	4

If we were using the AHP (Analytic Hierarchic Process) method there would be no need for the criteria weight information. However, the user still needs to insert the preference direction as shown in Figure G.3.

Once all the weights are inserted they must be normalized. Figure G.4 shows this normalization. Note that this is a simple weights normalization to intervals between [0,1] and does not involve the full normalization process explained in Section 6.7.1.

After all the data as been evaluated by the CSAEvaluator module, the Search Engine can start. The first Service Template in the Use Case CSA is the Database, which has the following requirements:

- "StorageCapacity > 200Tb"
- "Must have some Encryption feature"
- "Location = 'EU'"
- "Must not have the 'MySQL' platform"
- "DataTransferOUT per month > 200Tb"

In Figure G.5 we see these requirements mapped as exclusive requirements. In fact, in this particular case, all the requirements defined are exclusive, however, there might be cases where this does not happen. An important remark goes to the extra requirement. In the list above we only see 5 requirements, however, in Figure G.5 we see 6. The last requirement in the figure represents the global price requirement that was generalized according to the Generalization process explained in Section 6.5.1.

With this list of requirements and the service set used, it was possible to discover 8 service offerings to fit the Database Service Template. These are the alternatives. The last lines in Figure G.5 show precisely these 8 service offerings and their prices, also captured and calculated during the search process. Note that for both decision methods (SAW and AHP) the search process is the same.

```
STATUS: Evaluating CSA Data
Do you want to maximize the value of criterion: price in Service Template: General? (y/n)
n
Do you want to maximize the value of criterion: StorageCapacity in Service Template: database? (y/n)
y
Do you want to maximize the value of criterion: DataOUTExternal in Service Template: database? (y/n)
y
```

Figure G.3: CloudAid Example: Inserting criteria preference direction (AHP method)

```
SYSTEM: Normalizing Service Template Weights
SYSTEM: Total: 16.0 || value: 4.0|| res: 0.25
SYSTEM: Total: 16.0 || value: 5.0|| res: 0.3125
SYSTEM: Total: 16.0 || value: 3.0|| res: 0.1875
SYSTEM: Total: 16.0 || value: 2.0|| res: 0.125
SYSTEM: Total: 16.0 || value: 1.0|| res: 0.0625
SYSTEM: Total: 16.0 || value: 1.0|| res: 0.0625
SYSTEM: Normalizing general criteria
SYSTEM: Total: 2.0 || value: 2.0|| res: 1.0
SYSTEM: Normalizing criteria of the serviceTemplate: Comp0
SYSTEM: Total: 10.0 || value: 3.0|| res: 0.3
SYSTEM: Total: 10.0 || value: 5.0|| res: 0.5
SYSTEM: Total: 10.0 || value: 2.0|| res: 0.2
SYSTEM: Normalizing criteria of the serviceTemplate: Comp1
SYSTEM: Total: 6.0 || value: 4.0|| res: 0.6666666666666666
SYSTEM: Total: 6.0 || value: 2.0|| res: 0.3333333333333333
SYSTEM: Normalizing criteria of the serviceTemplate: Comp2
SYSTEM: Total: 9.799999952316284 || value: 1.5|| res: 0.15306122523454366
SYSTEM: Total: 9.799999952316284 || value: 4.0|| res: 0.40816326729211644
SYSTEM: Total: 9.799999952316284 || value: 1.0|| res: 0.10204081682302911
SYSTEM: Total: 9.799999952316284 || value: 1.29999995231628421| res: 0.13265305700425253
SYSTEM: Total: 9.799999952316284 || value: 2.0|| res: 0.20408163364605822
SYSTEM: Normalizing criteria of the serviceTemplate: Comp3
SYSTEM: Total: 8.799999952316284 || value: 2.5|| res: 0.28409091063028524
SYSTEM: Total: 8.799999952316284 || value: 2.0|| res: 0.2272727285042282
SYSTEM: Total: 8.799999952316284 || value: 1.0|| res: 0.1136363642521141
SYSTEM: Total: 8.799999952316284 || value: 1.29999995231628421| res: 0.14772726810914424
SYSTEM: Total: 8.799999952316284 || value: 2.0|| res: 0.2272727285042282
SYSTEM: Normalizing criteria of the serviceTemplate: Comp4
SYSTEM: Total: 6.0 || value: 4.0|| res: 0.6666666666666666
SYSTEM: Total: 6.0 || value: 2.0|| res: 0.3333333333333333
SYSTEM: Normalizing criteria of the serviceTemplate: Comp5
SYSTEM: Total: 6.0 || value: 4.0|| res: 0.6666666666666666
SYSTEM: Total: 6.0 || value: 2.0|| res: 0.3333333333333333
```

Figure G.4: CloudAid Example: Service Templates and Criteria weights normalization

```
    STATUS: Search for Service Template: database
    SYSTEM: Exclusive Requirements:
    SYSTEM: Requirement [id=Req3, priority=0, description=null, qualType=null, quantType=STORAGECAPACITY,
exclusivityMax=false, min=204800.0, max=0.0, needed=true, qualValue=null, criterion=true, exclusive=true]
    SYSTEM: Requirement [id=Req4, priority=0, description=null, qualType=ENCRYPTION, quantType=null,
exclusivityMax=false, min=0.0, max=0.0, needed=true, qualValue=null, criterion=false, exclusive=false]
    SYSTEM: Requirement [id=Req5, priority=0, description=null, qualType=LOCATION, quantType=null,
exclusivityMax=false, min=0.0, max=0.0, needed=true, qualValue=EU, criterion=false, exclusive=false]
    SYSTEM: Requirement [id=Req6, priority=0, description=null, qualType=PLATFORM, quantType=null,
exclusivityMax=false, min=0.0, max=0.0, needed=false, qualValue=MySQL, criterion=false, exclusive=false]
    SYSTEM: Requirement [id=Req7, priority=0, description=null, qualType=null, quantType=DATAOUTEXTERNAL,
exclusivityMax=false, min=204800.0, max=0.0, needed=true, qualValue=null, criterion=true, exclusive=true]
    SYSTEM: Requirement [id=Req37, priority=0, description=The overall System price must be bellow 5000€,
qualType=null, quantType=null, exclusivityMax=true, min=0.0, max=1000.0, needed=true, qualValue=null,
criterion=true, exclusive=true]
    Number of alternatives found:8
    SYSTEM - ServiceOffering_AmazonS3_6159 = 0.864
    SYSTEM - ServiceOffering_AmazonS3_6189 = 1.079
    SYSTEM - ServiceOffering_AmazonS3_6219 = 1.0639999
    SYSTEM - ServiceOffering_AmazonS3_6149 = 0.869
    SYSTEM - ServiceOffering_AmazonS3_6199 = 1.0739999
    SYSTEM - ServiceOffering_AmazonS3_6139 = 0.874
    SYSTEM - ServiceOffering_AmazonS3_6209 = 1.0689999
    SYSTEM - ServiceOffering_AmazonS3_6129 = 0.879
```

Figure G.5: CloudAid Example: Exclusive requirements and alternatives found

The application then converts all the Linked USDL *Service Offerings* found from RDF to Java Objects according to the Service Data Model depicted in Section 4.2.4.2. This conversion is performed by the ResourceConverter (see Section 6.6.3). To the already converted offerings is added the attributes data. These attributes are the values of each Service Offering for each criterion defined for the Service Template.

An example of an alternative is shown in Figure G.6. We can see three divisions: the qualitative features, the quantitative features and the attributes. The first two correspond directly to the service description and are tagged with the CloudTaxonomy concepts (See Section 5.1). The third part are the attribute values extracted from the other two parts. Note that for the Database Service Template two criteria are defined plus the global price criterion. Thus, as we see in Figure G.6 three attributes exist: Price, StorageCapacity and DataOutExternal.

The next step to be executed is the decision process, still for the Database Service Template. With the data collected in the Search Engine (alternatives and their attributes), the first task is to normalize all the data. This process is explained in detail in Section 6.7.1. Once again the user is requested for further information, in this case for any preferable value. In Figure G.7 we are defining, in the first 5 lines, that the storage capacity is going to have a preferable value of 5120000 (in this case the unit is Gb). This means that all the alternatives attribute values have to be normalized according to this preference. If none as been specified the preference direction is used. Figure G.7 shows the results obtained for the normalization process. The first step is to calculate the distances to the preferable value (5120000), the second is to normalize this distances. Line 6 to 13 show the distances results and the remaining lines show the normalization of these distances. The closer the value to the preferable value the better. If we add all the normalized attributes the result must be 1.

This example shows how the process goes for the numerical attributes. However, some Service Templates may have non-numerical attributes as is the case of the WebServer (that will be processed later on). As an example figure G.8 shows how they are treated. In this case the value is not binary since the Performance can assume Low, Medium, High and Very High (at least this is the perspective of the user). When asked to define the preferable value (which is mandatory for non-numerical attributes) the user defined "high" as the value. Then it is required to compare each alternative attribute value with this preferable value (distance). This distance is stored for each different attribute value found in the list of alternatives. This way we achieve a similar distances table to the one in Figure G.7 for numerical attributes. The last step, the actual normalization is performed in the same way as for the numerical attributes.

Once all the attributes have been normalized the decision problem can be formalized. We have all the alternatives and their attribute values for each criterion. Figure G.9 shows the final normalized values for all the alternatives found for the Database Service Template. All the data is ready for the XMCDA file which is the next step. Examples of these XMCDA files can be seen in the CloudAid Repository

```
- ALTERNATIVE: AmazonS3_6159
    - Qualitative Features
       - QualitativeFeature [name=Reduced Redundancy Storage, description=acTAO1MIIzvQYjiIR0oDbpfzdZ6DCH,
type=REDUNDANCY]
       - QualitativeFeature [name=EU Ireland, description=vVoSEJLAyeYjQLtLVj4HATP2M4Afwm, type=LOCATION]
       - QualitativeFeature [name=PostgreSQL, description=qG80kJHPHlA8E6_c5GpKbPeJP1o1Gp, type=PLATFORM]
       - QualitativeFeature [name=Authentication Mechanisms, description=7V4NHB8H76e6A81DmcfSCaaQ_DofMN,
type=SECURITY]
       - QualitativeFeature [name=REST interface, description=9_H4jlmkha936yVH_1I1ZZi0XYjlUw, type=API]
       - QualitativeFeature [name=SOAP interface, description=lPePaKFjse7JiFSPK37DZL3fFbB0Vj, type=API]
       - QualitativeFeature [name=HTTP, description=6KdVNX4r7OEVvpDt1TgGbtLMqQ7R7K, type=PROTOCOL]
       - QualitativeFeature [name=Secure data Upload/Download Encryption,
description=9ZO8agGJ5P8uiWgpsKgtEi6yumEa4K, type=ENCRYPTION]
       - QualitativeFeature [name=BitTorrent, description=zX4Wh9sxNaeI_dR4e7os1LC3HEUwE6, type=PROTOCOL]
       - QualitativeFeature [name=AmazonS3 Management Console, description=Ev6rcx2KRiPZii3rjbQu_NVj14TipL,
type=CONSOLE]
       - QualitativeFeature [name=Monitoring and controlling, description=YMkLCasQxNVIYETVpVH6LETo67OzfS,
type=MONITORING]
    - Quantitative Features
       - QuantitativeFeature [name=Resource_DATAOUTEXTERNAL_64838,
description=mQjMYSNlIfn5QyZgzmgBaGgJWtb1db, valueFloat=512000.0, unitOfMeasurement=GB,
type=DATAOUTEXTERNAL]
       - QuantitativeFeature [name=Resource_PUTREQUESTS_64838, description=45I_fHfcg_PL74sUAV7Hc4VLlBFkku,
valueFloat=1000.0, unitOfMeasurement=null, type=PUTREQUESTS]
       - QuantitativeFeature [name=Resource_DATAININTERNAL_64838,
description=aMqHUkKVAuJ6m1wgWRun_kniscmMqx, valueFloat=1.0, unitOfMeasurement=TB, type=DATAININTERNAL]
       - QuantitativeFeature [name=Resource_LISTREQUESTS_64838,
description=yy8hFPoqNdH3shSFfQLiP7ukpU46Yo, valueFloat=1000.0, unitOfMeasurement=null, type=LISTREQUESTS]
       - QuantitativeFeature [name=Resource_COPYREQUESTS_64838,
description=Xyu2OOiSPN3SpNZKysfu6GZPsFNFQl, valueFloat=1000.0, unitOfMeasurement=null, type=COPYREQUESTS]
       - QuantitativeFeature [name=Resource_GETREQUESTS_64838, description=yulCvEB_edM0CQSNcESZs9BGKgDHiT,
valueFloat=10000.0, unitOfMeasurement=null, type=GETREQUESTS]
       - QuantitativeFeature [name=Resource_DATAINEXTERNAL_64838,
description=lJGKvcF_OtScFWHHYMepNtchLsKQzE, valueFloat=1.0, unitOfMeasurement=TB, type=DATAINEXTERNAL]
       - QuantitativeFeature [name=Resource_DELETEREQUESTS_64838,
description=mfHQ9yHOZLeZKeSo357njbvHdfJheS, valueFloat=1000.0, unitOfMeasurement=null,
type=DELETEREQUESTS]
       - QuantitativeFeature [name=Resource_POSTREQUESTS_64838,
description=5beiJSH2SF0VXUqOZX80ZnYzONAp_s, valueFloat=1000.0, unitOfMeasurement=null, type=POSTREQUESTS]
       - QuantitativeFeature [name=Resource_DATAOUTINTERNAL_64838,
description=S9IiJro5JhT1PUuCvI8_6QxBzEooP9, valueFloat=1.0, unitOfMeasurement=TB, type=DATAOUTINTERNAL]
       - QuantitativeFeature [name=Resource_STORAGECAPACITY_64838,
description=a9ZLzFVJsm4k80x995BdHJgja7l4Uo, valueFloat=1.024E7, unitOfMeasurement=GB,
type=STORAGECAPACITY]
       - QuantitativeFeature [name=Resource_FILESIZE_63578, description=cmhMBZmXWCNECIgJ7cqbPhtU0uwV6u,
valueFloat=5120.0, unitOfMeasurement=GB, type=FILESIZE]
       - QuantitativeFeature [name=Resource_AVAILABILITY_63578,
description=3ZxHj4ObBqzDJVizLJBitcgcdND2y7, valueFloat=99.99, unitOfMeasurement=null, type=AVAILABILITY]
    - Attributes
      - price = 0.864
      - StorageCapacity = 1.024E7
      - DataOUTExternal = 512000.0
```

Figure G.6: CloudAid Example: Alternative Data

```
STATUS: Decision for Service Template: database
Does criterion StorageCapacity have a preferable value? (Y/N)
y
Please insert the preferable value for criterion: StorageCapacity:
5120000
SYSTEM: AmazonS3_6159 : StorageCapacity=5120000.0
SYSTEM: AmazonS3_6189 : StorageCapacity=4608000.0
SYSTEM: AmazonS3_6219 : StorageCapacity=5120000.0
SYSTEM: AmazonS3_6149 : StorageCapacity=0.0
SYSTEM: AmazonS3_6199 : StorageCapacity=4096000.0
SYSTEM: AmazonS3_6139 : StorageCapacity=4096000.0
SYSTEM: AmazonS3_6209 : StorageCapacity=0.0
SYSTEM: AmazonS3_6129 : StorageCapacity=4608000.0
SYSTEM: NORMALIZING---
SYSTEM: min: 0.0|| max: 5120000.0|| value: 5120000.0|| res: 0.0
SYSTEM: min: 0.0|| max: 5120000.0|| value: 4608000.0|| res: 0.09999999999999998
SYSTEM: min: 0.0|| max: 5120000.0|| value: 5120000.0|| res: 0.0
SYSTEM: min: 0.0|| max: 5120000.0|| value: 0.0|| res: 1.0
SYSTEM: min: 0.0|| max: 5120000.0|| value: 4096000.0|| res: 0.19999999999999996
SYSTEM: min: 0.0|| max: 5120000.0|| value: 4096000.0|| res: 0.19999999999999996
SYSTEM: min: 0.0|| max: 5120000.0|| value: 0.0|| res: 1.0
SYSTEM: min: 0.0|| max: 5120000.0|| value: 4608000.0|| res: 0.09999999999999998
```

Figure G.7: CloudAid Example: Insert preferable value and attribute normalization process

```
The criterion Performance is not numerical.
Is this a 2 option criterion? (Y/N)
n
Please insert the preferable value for criterion: Performance:
High
What is a distance between the value Network High(a) and the preferable value High(b) ?
0
```

Figure G.8: CloudAid Example: Insert data about a non-numerical attribute

```
AmazonS3_6159
SYSTEM: price = 1.0    SYSTEM: StorageCapacity = 0.0    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6189
SYSTEM: price = 0.0    SYSTEM: StorageCapacity = 0.09999999999999998    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6219
SYSTEM: price = 0.06976790697674407    SYSTEM: StorageCapacity = 0.0    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6149
SYSTEM: price = 0.9767441860465116    SYSTEM: StorageCapacity = 1.0    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6199
SYSTEM: price = 0.02325627906976724    SYSTEM: StorageCapacity = 0.19999999999999996    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6139
SYSTEM: price = 0.9534883720930232    SYSTEM: StorageCapacity = 0.19999999999999996    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6209
SYSTEM: price = 0.046512093023256096    SYSTEM: StorageCapacity = 1.0    SYSTEM: DataOUTExternal = 1.0
AmazonS3_6129
SYSTEM: price = 0.9302325581395348    SYSTEM: StorageCapacity = 0.09999999999999998    SYSTEM: DataOUTExternal = 1.0
```

Figure G.9: CloudAid Example: Decision Problem Data

[7]. Note also that the CloudAid Prototype execution halts until it receives the decision results from the Decision Method.

Depending on the method being executed the next steps in the interaction are different. If using the SAW method nothing needs to be done besides waiting for the results to arrive. However, when using the AHP method extra information is required. This information is inserted in the JAHP application (external to the CloudAid Prototype). Figure G.10 shows a screen of the application where the user is comparing the three criteria of the Database Service Template being evaluated. Figure G.11 on the other hand shows the screen used for comparing the alternatives regarding the StorageCapacity criterion. These comparisons are made by sliding up or down each slider according to the level of preference. When the user is satisfied with the data inserted, he can press "Save Alternatives" and the data will be sent to the CloudAid application. Note that a detailed explanation can be found in Section 6.7.4.2

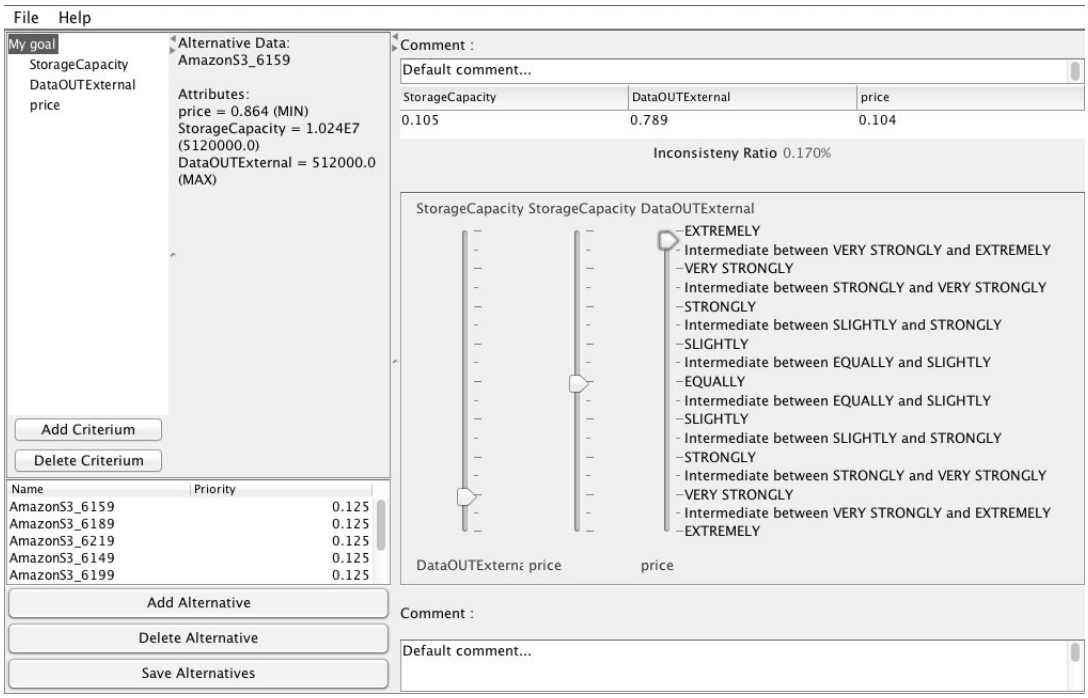


Figure G.10: JAHP Interface: Comparing criteria of the use case scenarios database Service Template

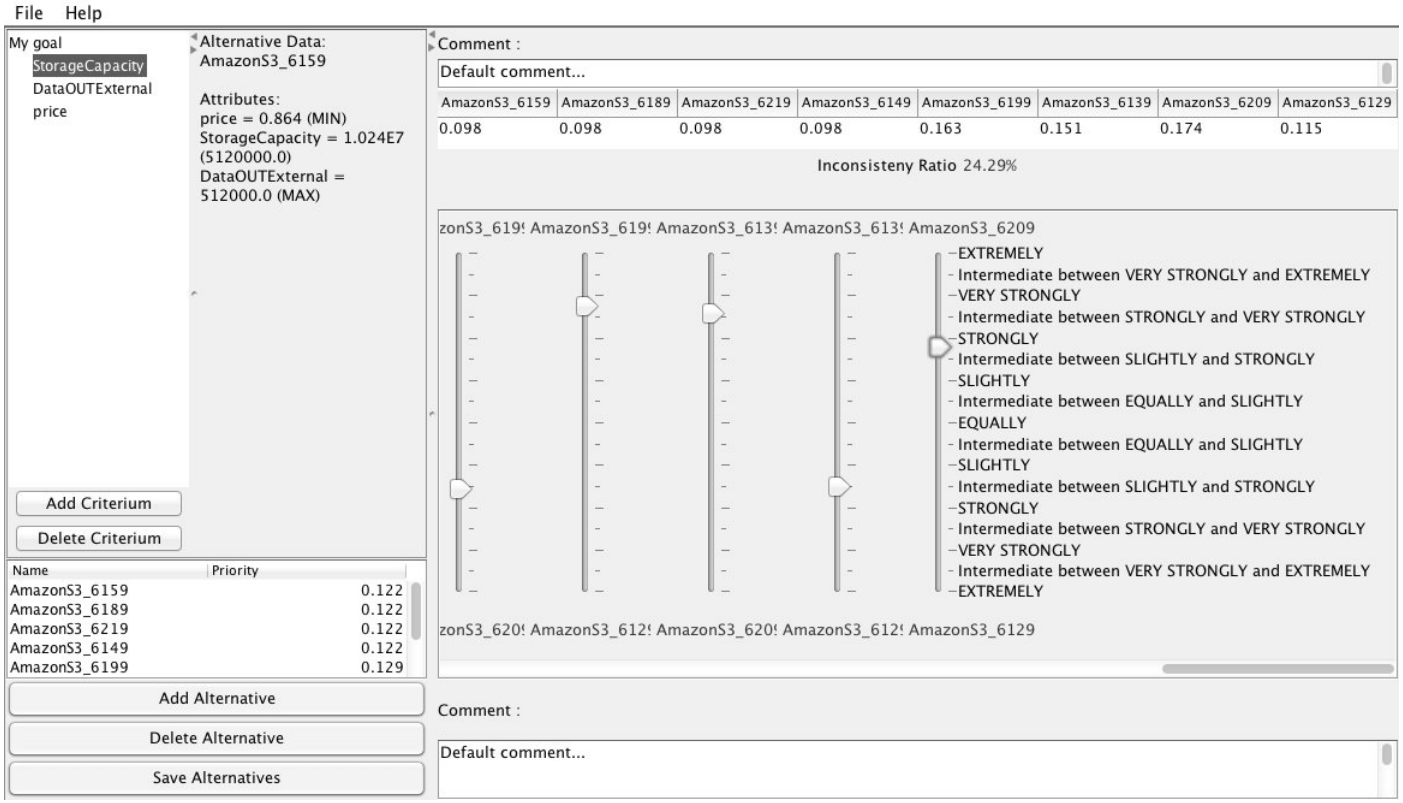


Figure G.11: JAHP Interface: Comparing alternatives of the Use Case Scenarios Database Service Template regarding the StorageCapacity criterion

```
Decision Results for Service Template: Comp0
    ALTERNATIVE1:AmazonS3_6159--->0.699999988079071
    ALTERNATIVE2:AmazonS3_6189--->0.5299999713897705
    ALTERNATIVE3:AmazonS3_6219--->0.5139535665512085
    ALTERNATIVE4:AmazonS3_6149--->0.9953488707542419
    ALTERNATIVE5:AmazonS3_6199--->0.5646512508392334
    ALTERNATIVE6:AmazonS3_6139--->0.750697672367096
    ALTERNATIVE7:AmazonS3_6209--->0.8093024492263794
    ALTERNATIVE8:AmazonS3_6129--->0.7160465121269226
Ordered Decision Results for Service Template: Comp0
    ALTERNATIVE1:AmazonS3_6149--->0.9953488707542419
    ALTERNATIVE2:AmazonS3_6209--->0.8093024492263794
    ALTERNATIVE3:AmazonS3_6139--->0.750697672367096
    ALTERNATIVE4:AmazonS3_6129--->0.7160465121269226
    ALTERNATIVE5:AmazonS3_6159--->0.699999988079071
    ALTERNATIVE6:AmazonS3_6199--->0.5646512508392334
    ALTERNATIVE7:AmazonS3_6189--->0.5299999713897705
    ALTERNATIVE8:AmazonS3_6219--->0.5139535665512085
```

Figure G.12: CloudAid Example: Decision Results extracted from the Decision Method

When the Decision Method finishes with its processing, it sends back the results as a XMCDA file. This file must be read and its data extracted. As seen in Figure G.12 each alternative is assigned a performance value. Note that the first 9 lines present the results as they come from the Decision Method, unsorted. Thus the final task is to sort these results from the highest to the lowest. This way we achieve our Sorted Ranked List of Alternatives.

This process is repeated for each Service Template in the CSA, thus, in the Use Case Scenario, 6 times. After all the Search and Decision mechanisms have been executed, the last step of the process can start, the Aggregation Engine.

As explained in Section 6.8 the Aggregation Engine's job is to combine one alternative of each Service Template in the CSA and check if this Aggregated Solution is Admissible or not. Finally when all the Admissible Aggregated Solutions are found (using one of the algorithms depicted in Section 6.8.1) the final step is to apply the Service Template weights in order to find the best solution. However, once again there is a difference between the Decision Method being executed. Using the SAW, nothing is required since the user has already inserted the Service Template Weights. But if using the AHP an extra step is required. A new request to the Decision Method must be sent (creating a new XMCDA file) to compare the Service Templates. Figure G.13 shows this step. In this case the user must compare the 6 Service Templates with each other. The final result is returned in the same way as before for the alternatives decision.

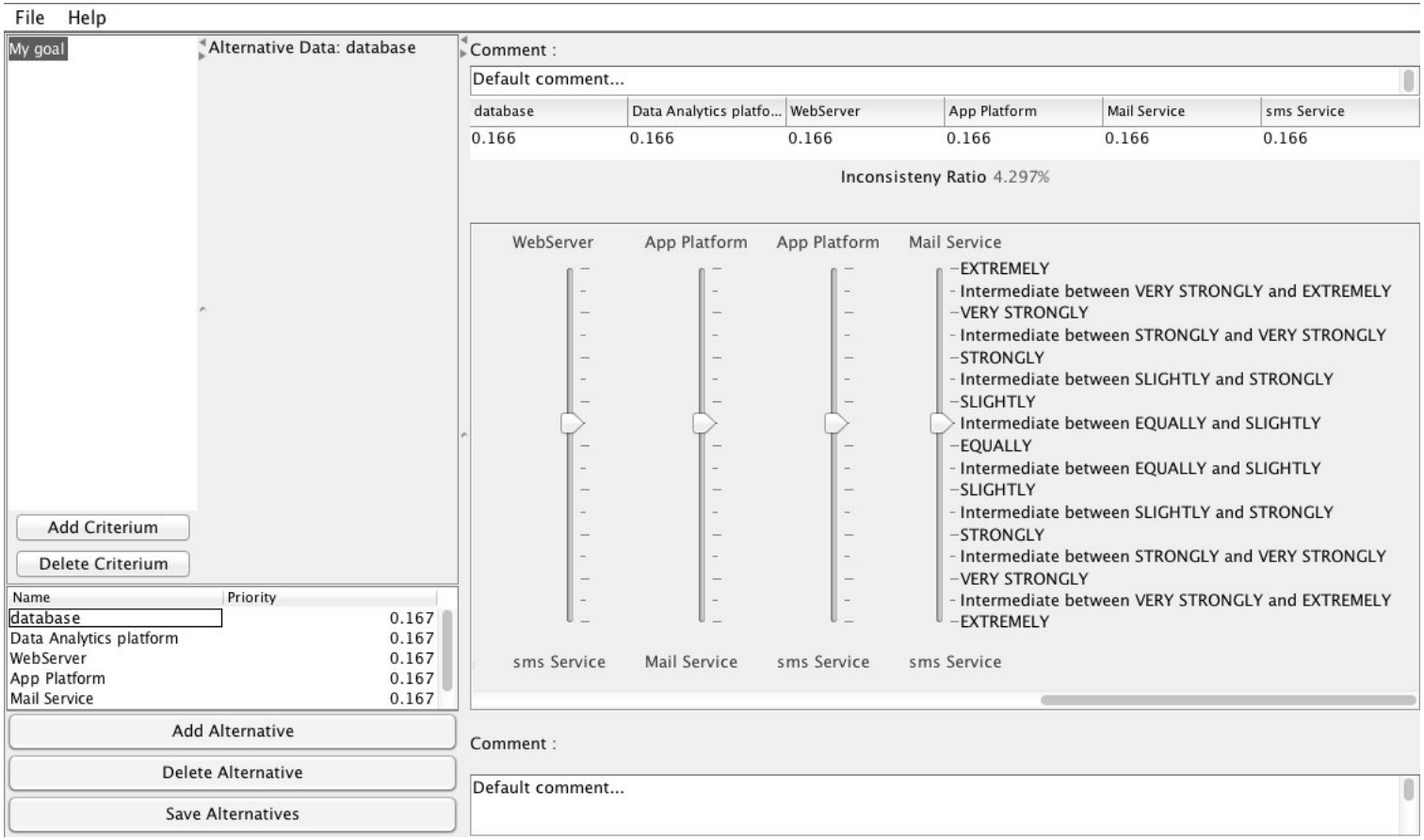


Figure G.13: JAHP Interface: Comparing Service Templates of the Use Case Scenario

```
STATUS: Starting Altenatives aggregation...
Computing Aggregation Solutions...
Computing alternative combinations...
SYSTEM: Using default algorithm.
Indexes:
000000
CSA RESULTS:
SERVICE TEMPLATE: Comp0
    ALTERNATIVE1:AmazonS3_6149--->0.9953488707542419
SERVICE TEMPLATE: Comp1
    ALTERNATIVE1:Analytics_service_5560--->0.6835443377494812
SERVICE TEMPLATE: Comp2
    ALTERNATIVE1:AmazonEC2_xlarge_m1_4284--->0.8469387888908386
SERVICE TEMPLATE: Comp3
    ALTERNATIVE1:AmazonEC2_xlarge_c1_2360--->1.0
SERVICE TEMPLATE: Comp4
    ALTERNATIVE1:Heroku_Addons_Mail_Service_35--->0.6666666865348816
SERVICE TEMPLATE: Comp5
    ALTERNATIVE1:SMS_Service_7441--->0.8533333539962769
Do you want to exit? (Y/N)
```

Figure G.14: CloudAid Example: Aggregation Results with Global Price limit of €5000 (SAW method)

The final results are then extracted and the values of each Service Template are used for calculating the overall performance of each Admissible Aggregated Solution. Finally the result is presented to the user as shown in Figure G.14.

From the Use Case Scenario we can see that the global price defined is €5000. This means that an Aggregates Solution is considered Admissible if its total price is bellow this value. Comparing the data collected during the execution, showed in Figure G.15, with the final results in Figure G.14 we conclude that the first alternative in the Sorted Ranked List of each Service Template was the one chosen to be part of the best Admissible Aggregated Solution. However, it might happen that the first alternative in each Sorted Ranked List when aggregated does not fulfill all the admissibility requirements, thus not being and Admissible Aggregated Solution. In fact, with the current example if we change the Price limit from €5000 to €1000 we see this different. Figure G.16 shows these results, where it is possible to see that the last Service Template does not use the first alternative in its Sorted Ranked List, it uses its second. The reason is because the total price of the Aggregated Solution composed by the first alternative in each Sorted Ranked List is above €1000 and therefore not Admissible. By the algorithm (in this example it is used the algorithm without incomparability support), the best Admissible Aggregated Solution is the one presented in Figure G.16.

Note also that with different Decision Methods also the final results may be different. Mainly because they calculate the performances for each alternative in different ways. Figure G.17 shows some results using the AHP method.

```
SERVICE TEMPLATE: Comp0
    ALTERNATIVE1:AmazonS3_6149--->0.9953488707542419
    ALTERNATIVE2:AmazonS3_6209--->0.8093024492263794
    ALTERNATIVE3:AmazonS3_6139--->0.750697672367096
    ALTERNATIVE4:AmazonS3_6129--->0.7160465121269226
    ALTERNATIVE5:AmazonS3_6159--->0.699999988079071
    ALTERNATIVE6:AmazonS3_6199--->0.5646512508392334
    ALTERNATIVE7:AmazonS3_6189--->0.529999713897705
    ALTERNATIVE8:AmazonS3_6219--->0.5139535665512085
SERVICE TEMPLATE: Comp1
    ALTERNATIVE1:Analytics_service_5560--->0.6835443377494812
    ALTERNATIVE2:Analytics_service_5561--->0.6814346313476562
    ALTERNATIVE3:Analytics_service_5562--->0.6772152185440063
    ALTERNATIVE4:Analytics_service_5563--->0.671940227371216
    ALTERNATIVE5:Analytics_service_5564--->0.6666666865348816
    ALTERNATIVE6:Analytics_service_5555--->0.3333333432674408
    ALTERNATIVE7:Analytics_service_5556--->0.33122363686561584
    ALTERNATIVE8:Analytics_service_5557--->0.32700422406196594
    ALTERNATIVE9:Analytics_service_5558--->0.32172995805740356
    ALTERNATIVE10:Analytics_service_5559--->0.3164556920528412
SERVICE TEMPLATE: Comp2
    ALTERNATIVE1:AmazonEC2_xlarge_m1_4284--->0.8469387888908386
    ALTERNATIVE2:AmazonEC2_xlarge_m1_4316--->0.8005566000938416
    ALTERNATIVE3:AmazonEC2_xlarge_m1_4308--->0.7820037007331848
    ALTERNATIVE4:AmazonEC2_xlarge_c1_2364--->0.77458256483078
    ALTERNATIVE5:AmazonEC2_xlarge_m1_4300--->0.7634508609771729
    ALTERNATIVE6:AmazonEC2_xlarge_m1_4292--->0.7356215119361877
    ALTERNATIVE7:AmazonEC2_xlarge_c1_2396--->0.728200376033783
    ALTERNATIVE8:AmazonEC2_xlarge_c1_2388--->0.709647536277771
    ALTERNATIVE9:AmazonEC2_xlarge_c1_2380--->0.691094696521759
    ALTERNATIVE10:AmazonEC2_xlarge_c1_2372--->0.6632653474807739
SERVICE TEMPLATE: Comp3
    ALTERNATIVE1:AmazonEC2_xlarge_c1_2360--->1.0
    ALTERNATIVE2:AmazonEC2_xlarge_c1_2392--->0.9741735458374023
    ALTERNATIVE3:AmazonEC2_xlarge_c1_2384--->0.9638429880142212
    ALTERNATIVE4:AmazonEC2_xlarge_c1_2376--->0.95351243019104
    ALTERNATIVE5:AmazonEC2_xlarge_c1_2368--->0.9380165338516235
    ALTERNATIVE6:AmazonEC2_xlarge_c1_2361--->0.8347107172012329
    ALTERNATIVE7:AmazonEC2_xlarge_c1_2393--->0.8088842630386353
    ALTERNATIVE8:AmazonEC2_xlarge_c1_2385--->0.7985536456108093
    ALTERNATIVE9:AmazonEC2_xlarge_c1_2377--->0.7882230877876282
    ALTERNATIVE10:AmazonEC2_xlarge_c1_2369--->0.7727272510528564
SERVICE TEMPLATE: Comp4
    ALTERNATIVE1:Heroku_Addons_Mail_Service_35--->0.6666666865348816
    ALTERNATIVE2:Heroku_Addons_Mail_Service_34--->0.4278438687324524
    ALTERNATIVE3:Heroku_Addons_Mail_Service_32--->0.37364038825035095
    ALTERNATIVE4:Heroku_Addons_Mail_Service_30--->0.3412638008594513
    ALTERNATIVE5:Heroku_Addons_Mail_Service_28--->0.3333333432674408
    ALTERNATIVE6:Heroku_Addons_Mail_Service_29--->0.3251037001609802
    ALTERNATIVE7:Heroku_Addons_Mail_Service_31--->0.2954372763633728
    ALTERNATIVE8:Heroku_Addons_Mail_Service_33--->0.2497098829271698
SERVICE TEMPLATE: Comp5
    ALTERNATIVE1:SMS_Service_7441--->0.8533333539962769
    ALTERNATIVE2:SMS_Service_7442--->0.8453333377838135
    ALTERNATIVE3:SMS_Service_7443--->0.8133333325386047
    ALTERNATIVE4:SMS_Service_7444--->0.6666666865348816
    ALTERNATIVE5:SMS_Service_7436--->0.3333333432674408
    ALTERNATIVE6:SMS_Service_7437--->0.3253333568572998
    ALTERNATIVE7:SMS_Service_7438--->0.29333335161209106
    ALTERNATIVE8:SMS_Service_7439--->0.14666667580604553
```

Figure G.15: CloudAid Example: Sorted Ranked Lists for each Service Template (SAW method)

```
CSA RESULTS:
SERVICE TEMPLATE: Comp0
    ALTERNATIVE1:AmazonS3_6149--->0.9953488707542419
SERVICE TEMPLATE: Comp1
    ALTERNATIVE1:Analytics_service_5560--->0.6835443377494812
SERVICE TEMPLATE: Comp2
    ALTERNATIVE1:AmazonEC2_xlarge_m1_4284--->0.8469387888908386
SERVICE TEMPLATE: Comp3
    ALTERNATIVE1:AmazonEC2_xlarge_c1_2360--->1.0
SERVICE TEMPLATE: Comp4
    ALTERNATIVE1:Heroku_Addons_Mail_Service_34--->0.4278438687324524
SERVICE TEMPLATE: Comp5
    ALTERNATIVE1:SMS_Service_7441--->0.8533333539962769
```

Figure G.16: CloudAid Example: Aggregation Results with Global Price limit of €1000 (SAW method)

```
CSA RESULTS:
SERVICE TEMPLATE: Comp0
    ALTERNATIVE1:AmazonS3_6129--->0.125
SERVICE TEMPLATE: Comp1
    ALTERNATIVE1:Analytics_service_5555--->0.10000000149011612
SERVICE TEMPLATE: Comp2
    ALTERNATIVE1:AmazonEC2_xlarge_m1_4292--->0.10000000149011612
SERVICE TEMPLATE: Comp3
    ALTERNATIVE1:AmazonEC2_xlarge_c1_2385--->0.10000000149011612
SERVICE TEMPLATE: Comp4
    ALTERNATIVE1:Heroku_Addons_Mail_Service_35--->0.125
SERVICE TEMPLATE: Comp5
    ALTERNATIVE1:SMS_Service_7436--->0.125
```

Figure G.17: CloudAid Example: Aggregation Results with Global Price limit of €1000 (AHP method)

Bibliography

[1] Al-Ali, A., El-Hag, A., Bahadiri, M., Harbaji, M., and Ali El Haj, Y. Smart home renewable energy management system. *Energy Procedia 12* (2011), 120–126.

[2] Apache. Apache jena homepage. `http://jena.apache.org/`. Accessed: 5/5/2013.

[3] Apfelbacher, R., and Curth, A. Fmc and uml. `http://www.fmc-modeling.org/fmc-and-uml`. Accessed: 5/5/2012.

[4] Araújo, J. CloudAid Algorithm Comparison Test Inputs. `https://github.com/jorgearj/CloudAid/blob/master/CloudAid/PrototypeTesting/AlgorithmTests/ComparisonTestInputs.txt`. Accessed: 26/5/2013.

[5] Araújo, J. CloudAid Algorithm Tests. `https://github.com/jorgearj/CloudAid/tree/master/CloudAid/PrototypeTesting/AlgorithmTests`. Accessed: 26/5/2013.

[6] Araújo, J. CloudAid Architecture Diagrams. `https://github.com/jorgearj/CloudAid/tree/master/CloudAid/Architecture`. Accessed: 27/6/2013.

[7] Araújo, J. CloudAid Public Repository. `https://github.com/jorgearj/CloudAid/tree/master/CloudAid`. Accessed: 22/6/2013.

[8] Araújo, J. CloudAid Reliability Tests Document. `https://github.com/jorgearj/CloudAid/blob/master/CloudAid/PrototypeTesting/ReliabilityTestSheet.xlsx`. Accessed: 22/6/2013.

[9] Araújo, J. CloudAid Scenarion3 Document. `https://github.com/jorgearj/CloudAid/blob/master/CloudAid/PrototypeTesting/Scenario3_NoInc.pdf`. Accessed: 26/5/2013.

[10] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Commun. ACM 53*, 4 (Apr. 2010), 50–58.

[11] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., et al. Semantic web services framework (swsf) overview. *World Wide Web Consortium, Member Submission SUBM-SWSF-20050909* (2005).

[12] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D.,

ET AL. Semantic web services ontology (swso). *Member submission, W3C* (2005).

[13] BICHIER, M., AND LIN, K.-J. Service-oriented computing. *Computer 39*, 3 (march 2006), 99 – 101.

[14] BISDORFF, R., MEYER, P., AND VENEZIANO, T. Quick dive into xmcda-2.0. *Decision Deck* (2009).

[15] BIZER, C., HEATH, T., AND BERNERS-LEE, T. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems 5*, 3 (2009), 1–22.

[16] BLAKE, M., TAN, W., AND ROSENBERG, F. Composition as a service [web-scale workflow]. *Internet Computing, IEEE 14*, 1 (jan.-feb. 2010), 78 –82.

[17] BOOCH, G., RUMBAUGH, J., AND JACOBSON, I. *The unified modeling language user guide*. Pearson Education India, 1999.

[18] BOOTH, D., HAAS, H., MCCABE, F., NEWCOMER, E., CHAMPION, M., FERRIS, C., AND ORCHARD, D. Web Services Architecture, Oct. 2004.

[19] BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSOHN, N., NIELSEN, H., THATTE, S., AND WINER, D. Simple object access protocol (soap) 1.1, 2000.

[20] CARDOSO, J., BARROS, A., MAY, N., AND KYLAU, U. Towards a unified service description language for the internet of services: Requirements and first developments. In *Services Computing (SCC), 2010 IEEE International Conference on* (july 2010), pp. 602 –609.

[21] CARDOSO, J., PEDRINACI, C., LEIDIG, T., RUPINO, P., AND DE LEENHEER, P. Open semantic service networks. In *International Symposium on Services Science (ISSS?12), Leipzig, Germany* (2012).

[22] CARDOSO, J., AND SHETH, A. Semantic e-workflow composition. *Journal of Intelligent Information Systems 21*, 3 (2003), 191–225.

[23] CARDOSO, J., WINKLER, M., AND VOIGT, K. A service description language for the internet of services. In *Proceedings of ISSS* (2009).

[24] CASATI, F., ILNICKI, S., JIN, L., KRISHNAMOORTHY, V., AND SHAN, M.-C. Adaptive and dynamic service composition in eflow. In *Advanced Information Systems Engineering*, B. Wangler and L. Bergman, Eds., vol. 1789 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2000, pp. 13–31.

[25] CHEN, M.-F., TZENG, G.-H., AND DING, C. Fuzzy mcdm approach to select service provider. In *Fuzzy Systems, 2003. FUZZ '03. The 12th IEEE International Conference on* (may 2003), vol. 1, pp. 572 – 577 vol.1.

[26] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al. Web services description language (wsdl) 1.1, 2001.

[27] Cisco. Cisco cloudwatch summer 2012. http://www.cisco.com/cisco/web/UK/assets/cisco_cloudwatch_2012_2606.pdf. Accessed: 27/12/2012.

[28] Clegg, D., and Barker, R. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[29] Cloud, W. A. Bime analytics. http://www.bimeanalytics.com/. Accessed: 15/11/2012.

[30] Comission, E. Energy efficiency standards. http://ec.europa.eu/energy/efficiency/index_en.htm. Accessed: 12/12/2012.

[31] Comission, E. Unleashing the potential of cloud computing in europe. http://ec.europa.eu/information_society/activities/cloudcomputing/docs/com/com_cloud.pdf. Accessed: 27/12/2012.

[32] de Bruijn, J., Lausen, H., Polleres, A., and Fensel, D. The web service modeling language wsml: An overview. In *The Semantic Web: Research and Applications*, Y. Sure and J. Domingue, Eds., vol. 4011 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 590–604.

[33] De Silva, L., Morikawa, C., and Petra, I. State of the art of smart homes. *Engineering Applications of Artificial Intelligence* (2012).

[34] Deck, D. Decision deck - xmcda. http://www.decision-deck.org/xmcda/index.html. Accessed: 5/5/2013.

[35] Deck, D. The decision deck project. http://www.decision-deck.org. Last Accessed: 5/5/2013.

[36] Deck, D. The decision deck project manifesto. http://www.decision-deck.org/_static/D2manifesto.pdf. Last Accessed: 13/01/2013.

[37] Deck, D. J-XMCDA project. http://sourceforge.net/apps/mediawiki/j-mcda/index.php?title=Main_Page. Accessed: 5/5/2013.

[38] Deck, D. Xmcda 2.2.0 documentation. http://www.decision-deck.org/xmcda/_static/html-doc/2.2.0/XMCDA-2.2.0.html. Accessed: 5/5/2013.

[39] Deck, D. Xmcda Schema. http://www.decision-deck.org/xmcda/_downloads/XMCDA-2.2.0.xsd. Accessed: 26/5/2013.

[40] Dietze, S., Liu, D., Yu, H., and Pedrinaci, C. Semantic web-driven development of service-oriented systems-exploiting linked data for service annotation and discovery.

[41] Dustdar, S., and Schreiner, W. A survey on web services composition. *International Journal of Web and Grid Services 1*, 1 (2005), 1–30.

[42] Eeles, P. Capturing architectural requirements,. *Technical report IBM* (November 2005).

[43] Erl, T. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl).* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[44] Fehling, C., Leymann, F., Mietzner, R., and Schupeck, W. A collection of patterns for cloud types, cloud service models, and cloud-based application architectures, May 2011.

[45] Fensel, D., and Bussler, C. The web service modeling framework wsmf. *Electronic Commerce Research and Applications 1*, 2 (2002), 113 – 137.

[46] Figueira, J., Greco, S., and Ehrgott, M. *Multiple Criteria Decision Analysis: State of the Art Surveys.* Springer Verlag, Boston, Dordrecht, London, 2005.

[47] Fortis, T.-F., Munteanu, V. I., and Negru, V. Towards an ontology for cloud services. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on* (2012), IEEE, pp. 787–792.

[48] Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., and Vaquero, L. M. Service specification in cloud environments based on extensions to open standards. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE* (New York, NY, USA, 2009), COMSWARE '09, ACM, pp. 19:1–19:12.

[49] Gartner. Gartner says cloud consumers need brokerages to unlock the potential of cloud services. `http://www.gartner.com/newsroom/id/1064712`, 2009. Accessed: 19/6/2013.

[50] Gill, K., Yang, S., Yao, F., and Lu, X. A zigbee-based home automation system. *Consumer Electronics, IEEE Transactions on 55*, 2 (2009), 422–430.

[51] Guan, Y., Ghose, A., and Lu, Z. Using constraint hierarchies to support qos-guided service composition. In *Web Services, 2006. ICWS '06. International Conference on* (sept. 2006), pp. 743 –752.

[52] Han, T., and Sim, K. M. An ontology-enhanced cloud service discovery system. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (2010), vol. 1.

[53] Haslhofer, B., Momeni Roochi, E., Schandl, B., and Zander, S. Europeana rdf store report.

[54] Heberle, F. Comparison of service offerings in the future internet, spanning over multiple providers and stores, 2012.

[55] Hepp, M. Goodrelations: An ontology for describing products and services offers on the web. In *Knowledge Engineering: Practice and Patterns*, A. Gangemi and J. Euzenat, Eds., vol. 5268 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008, pp. 329–346.

[56] Hill, T. On goods and services. *Review of income and wealth 23*, 4 (2005), 315–338.

[57] IDC. Quantitative estimates of the demand for cloud computing in europe and the likely barriers to uptake. `http://ec.europa.eu/information_society/activities/cloudcomputing/docs/quantitative_estimates.pdf`. Accessed: 27/12/2012.

[58] Kang, J., and Sim, K. M. Cloudle: A multi-criteria cloud service search engine. In *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific* (2010), IEEE, pp. 339–346.

[59] Kang, J., and Sim, K. M. Cloudle: An agent-based cloud search engine that consults a cloud ontology. In *Cloud Computing and Virtualization Conference, CCV* (2010).

[60] Kapitsaki, G., Kateros, D., Foukarakis, I., Prezerakos, G., Kaklamani, D., and Venieris, I. Service composition: State of the art and future challenges. In *Mobile and Wireless Communications Summit, 2007. 16th IST* (july 2007), pp. 1 –5.

[61] Keller, F., and Wendt, S. Fmc: an approach towards architecture-centric system development. In *Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the* (april 2003), pp. 173 – 182.

[62] Kim, H., Lee, S., Kim, H., and Kim, H. Implementing home energy management system with upnp and mobile applications. *Computer Communications* (2012).

[63] Kim, W., Lee, S., and Hwang, J. Real-time energy monitoring and controlling system based on zigbee sensor networks. *Procedia Computer Science 5* (2011), 794–797.

[64] Klusch, M. Semantic web service description. In *CASCOM: Intelligent Service Coordination in the Semantic Web*, M. Schumacher, H. Schuldt, H. Helin, M. Walliser, S. Brantschen, M. Calisti, and T. Hempfling, Eds., Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Basel, 2008, pp. 31–57.

[65] Knöpfel, A., Gröne, B., and Tabeling, P. *Fundamental modeling concepts*. Wiley, West Sussex UK, 2005.

[66] Knublauch, H. The Object-Oriented Semantic Web with SPIN. `http://composing-the-semantic-web.blogspot.pt/2009/01/`

object-oriented-semantic-web-with-spin.html. Accessed: 21/5/2013.

[67] Knublauch, H. Spin - SPARQL Inferencing Notation. http://spinrdf.org/. Accessed: 21/5/2013.

[68] Knublauch, H. Spin - SPARQL Syntax. http://www.w3.org/Submission/spin-sparql/. Accessed: 21/5/2013.

[69] Kopecky, J., Vitvar, T., Bournez, C., and Farrell, J. Sawsdl: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE 11*, 6 (nov.-dec. 2007), 60 –67.

[70] Krasner, G. E., Pope, S. T., et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming 1*, 3 (1988), 26–49.

[71] Lara, R., Lausen, H., Arroyo, S., Bruijn, J. D., Fensel, D., and Innsbruck, U. Semantic web services: description requirements and current technologies. In *In Proceedings of the International Workshop on Electronic Commerce, Agents, and Semantic Web Services held in conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003* (2003).

[72] Lawler, C. M. Cloud service broker. http://www.hitachiconsulting.com/files/pdfRepository/Cloud-Service-Broker-Presentation-Green-IT-Cloud-Summit-2012.pdf. Accessed: 19/6/2013.

[73] Lee, B. S., Yan, S., Ma, D., and Zhao, G. Aggregating iaas service. In *SRII Global Conference (SRII), 2011 Annual* (2011), IEEE, pp. 335–338.

[74] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. Whats inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing* (Washington, DC, USA, 2009), CLOUD '09, IEEE Computer Society, pp. 23–31.

[75] Lheureux, B., Petrov, P., and Thurai, A. The rise of cloud service brokerage featuring gartner and bcbs. http://www.slideshare.net/Intel-ASIP/the-rise-of-cloud-service-brokerage-featuring-gartner-and-bcbs, 2012. Accessed: 19/6/2013.

[76] Liu, D., Li, N., Pedrinaci, C., Kopecký, J., Maleshkova, M., and Domingue, J. An approach to construct dynamic service mashups using lightweight semantics. In *Current Trends in Web Engineering*, A. Harth and N. Koch, Eds., vol. 7059 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2012, pp. 13–24.

[77] Manifesto, O. C. Open cloud manifesto. *Availabe online: www. opencloudmanifesto. org/Open 20* (2009).

[78] Markowitz, H. Portfolio selection*. *The Journal of Finance 7*, 1 (1952), 77–91.

[79] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al. Owl-s: Semantic markup for web services. *W3C Member submission 22* (2004), 2007–04.

[80] Martin, J. *Rapid application development*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1991.

[81] Marton, A., Piccinelli, G., and Turfin, C. Service provision and composition in virtual business communities. In *Reliable Distributed Systems, 1999. Proceedings of the 18th IEEE Symposium on* (1999), pp. 336 –341.

[82] McGuinness, D., Van Harmelen, F., et al. Owl web ontology language overview. *W3C recommendation 10*, 2004-03 (2004), 10.

[83] Mcilraith, S. Adapting golog for composition of semantic web services. pp. 482–493.

[84] Mell, P., and Grance, T. The nist definition of cloud computing (draft). *NIST special publication 800* (2011), 145.

[85] Meyer, P., and Bigaret, S. Diviz: a software for modeling, processing and sharing algorithmic workflows in mcda. *Intelligent Decision Technologies: an International Journal* (2011).

[86] Milanovic, N., and Malek, M. Current solutions for web service composition. *Internet Computing, IEEE 8*, 6 (nov.-dec. 2004), 51 – 59.

[87] Morge, M. Jahp - java analytic hierarchy process. `http://www.di.unipi.it/~morge/software/JAHP.html`. Accessed: 3/3/2013.

[88] Mortensen, D. T., and Pissarides, C. A. Job creation and job destruction in the theory of unemployment. *The Review of Economic Studies 61*, 3 (1994), 397–415.

[89] Moscato, F., Aversa, R., Di Martino, B., Fortis, T., and Munteanu, V. An analysis of mosaic ontology for cloud resources annotation. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on* (2011), IEEE, pp. 973–980.

[90] Nair, S., Porwal, S., Dimitrakos, T., Ferrer, A., Tordsson, J., Sharif, T., Sheridan, C., Rajarajan, M., and Khan, A. Towards secure cloud bursting, brokerage and aggregation. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on* (2010), pp. 189–196.

[91] Narayanan, S., and McIlraith, S. A. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web* (New York, NY, USA, 2002), WWW '02, ACM, pp. 77–88.

[92] Nguyen, D., Lelli, F., Taher, Y., Parkin, M., Papazoglou, M., and van den Heuvel, W.-J. Blueprint template support for engineering cloud-based services. In *Towards a Service-Based Internet*, W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssière, Eds., vol. 6994 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011, pp. 26–37.

[93] Nguyen, D. K., Lelli, F., Papazoglou, M., and van den Heuvel, W.-J. Issue in automatic combination of cloud services. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on* (july 2012), pp. 487 –493.

[94] OpenCrowd. Cloud taxonomy. `http://cloudtaxonomy.opencrowd.com/taxonomy/`. Accessed: 20/03/2013.

[95] Orriëns, B., Yang, J., and Papazoglou, M. Model driven service composition. In *Service-Oriented Computing - ICSOC 2003*, M. Orlowska, S. Weerawarana, M. Papazoglou, and J. Yang, Eds., vol. 2910 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 75–90.

[96] O'Sullivan, J., Edmond, D., and ter Hofstede, A. Formal description of non-functional service properties. *Centre for Information Technology, Queensland University of Technology, Tech. Rep* (2005).

[97] Pedrinaci, C., da Silva, C. F., Araújo, J., Cardoso, J., and Leidig, T. Linked USDL Pricing Repository. `https://github.com/linked-usdl/usdl-price`. Accessed: 21/5/2013.

[98] Pedrinaci, C., and Domingue, J. Toward the next wave of services: linked services for the web of data. *Journal of Universal Computer Science 16*, 13 (2010), 1694–1719.

[99] Pedrinaci, C., Kopeckỳ, J., Maleshkova, M., Liu, D., Li, N., and Domingue, J. Unified lightweight semantic descriptions of web apis and web services.

[100] Pedrinaci, C., and Leidig, T. Linked usdl core. `http://linked-usdl.org/ns/usdl-core`. Accessed: 5/05/2013.

[101] Pedrinaci, C., and Leidig, T. Linked-usdl homepage. `http://www.linked-usdl.org/`. Accessed: 19/11/2012.

[102] Pedrinaci, C., Leidig, T., Cardoso, J., and Araújo, J. Linked usdl pricing repository. `https://github.com/linked-usdl/usdl-price/blob/master/usdl-price_v2.ttl`. Accessed: 28/5/2013.

[103] Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., and Domingue, J. iserve: a linked services publishing platform. In *CEUR Workshop Proceedings* (2010), vol. 596.

[104] Prud'hommeaux, E., and Seaborne, A. SPARQL Query Language for RDF. Tech. rep.

[105] Rahm, E., and Bernstein, P. A. A survey of approaches to automatic schema matching. *The VLDB Journal 10*, 4 (Dec. 2001), 334–350.

[106] Rao, J., and Su, X. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition*, J. Cardoso and A. Sheth, Eds., vol. 3387 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 43–54.

[107] Reiter, R. On closed world data bases. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Springer US, 1978, pp. 55–76.

[108] Reseach, S. Internet of services - service delivery framework. `http://www.internet-of-services.com/index.php?id=265&L=0&tx_ttnews[backpid]=475&tx_ttnews[tt_news]=236&tx_ttnews[pointer]=1`. Accessed: 19/6/2013.

[109] Rimal, B. P., Choi, E., and Lumb, I. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on* (2009), Ieee, pp. 44–51.

[110] Robbins, S., Judge, T., et al. Organizational behaviour, 1996.

[111] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D., et al. Web service modeling ontology. *Applied Ontology 1*, 1 (2005), 77–106.

[112] Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., and Dustdar, S. An end-to-end approach for qos-aware service composition. In *Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International* (sept. 2009), pp. 151 –160.

[113] Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., and Dustdar, S. Towards composition as a service - a quality of service driven approach. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on* (29 2009-april 2 2009), pp. 1733 –1740.

[114] Saaty, T. L. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research 48*, 1 (1990), 9 – 26. ¡ce:title¿Desicion making by the analytic hierarchy process: Theory and applications¡/ce:title¿.

[115] Schroth, C., and Janner, T. Web 2.0 and soa: Converging concepts enabling the internet of services. *IT Professional 9*, 3 (may-june 2007), 36 –41.

[116] Schwaber, K. *Agile project management with Scrum*. Microsoft Press, 2004.

[117] September, A. Ieee standard glossary of software engineering terminology, 1990.

[118] Sharpe, W. A simplified model for portfolio analysis. *Management science 9*, 2 (1963), 277–293.

[119] Shin, J., and Hwang, J. Intelligent energy information service based on a multi-home environment. *Procedia Computer Science 10* (2012), 197–204.

[120] Sirin, E., Hendler, J., and Parsia, B. Semi-automatic composition of web services using semantic descriptions. In *In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003* (2002), pp. 17–24.

[121] Spohrer, J., Maglio, P. P., Bailey, J., and Gruhl, D. Steps toward a science of service systems. *Computer 40*, 1 (jan. 2007), 71 –77.

[122] Spohrer, J., Vargo, S., Caswell, N., and Maglio, P. The service system is the basic abstraction of service science. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual* (jan. 2008), p. 104.

[123] Srivastava, B., and Koehler, J. Web service composition-current solutions and open problems. In *ICAPS 2003 Workshop on Planning for Web Services* (2003), vol. 35.

[124] Sun, H., Wang, X., Zhou, B., and Zou, P. Research and implementation of dynamic web services composition. In *Advanced Parallel Processing Technologies*, X. Zhou, M. Xu, S. Jähnichen, and J. Cao, Eds., vol. 2834 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 457–466.

[125] Toma, I., Roman, D., Fensel, D., Sapkota, B., and Gomez, J. A multi-criteria service ranking approach based on non-functional properties rules evaluation. In *Service-Oriented Computing – ICSOC 2007*, B. Krämer, K.-J. Lin, and P. Narasimhan, Eds., vol. 4749 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007, pp. 435–441.

[126] Tran, V., and Tsuji, H. A survey of fuzzy-based approaches for web service ranking. *International Journal of Web Services Practices 3*, 3-4 (2008), 121–128.

[127] Triantaphyllou, E., Shu, B., Sanchez, S. N., and Ray, T. Multi-criteria decision making: an operations research approach. *Encyclopedia of electrical and electronics engineering 15* (1998), 175–186.

[128] Tut, M., and Edmond, D. The use of patterns in service composition. In *Web Services, E-Business, and the Semantic Web*, C. Bussler, R. Hull, S. McIlraith, M. Orlowska, B. Pernici, and J. Yang, Eds., vol. 2512 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002, pp. 28–40.

[129] Vitvar, T., Kopecky, J., Zaremba, M., and Fensel, D. Wsmo-lite: lightweight semantic descriptions for services on the web. In *Web Services, 2007. ECOWS '07. Fifth European Conference on* (nov. 2007), pp. 77 –86.

[130] Wright, P., Sun, Y. L., Harmer, T., Keenan, A., Stewart, A., and Perrott, R. A constraints-based resource discovery model for multi-provider cloud environments. *Journal of Cloud Computing 1*, 1 (2012), 1–14.

[131] Wu, D., Parsia, B., Sirin, E., Hendler, J., and Nau, D. Automating daml-s web services composition using shop2. In *The Semantic Web - ISWC 2003*, D. Fensel, K. Sycara, and J. Mylopoulos, Eds., vol. 2870 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 195–210.

[132] www.fmc modeling.org. Fundamental modeling concepts homepagel. `http://www.fmc-modeling.org/home`. Accessed: 5/5/2012.

[133] Xie, X., and Chen, K. An ahp-based evaluation model for service composition. In *Computational Science and Its Applications - ICCSA 2006*, M. Gavrilova, O. Gervasi, V. Kumar, C. Tan, D. Taniar, A. Laganá, Y. Mun, and H. Choo, Eds., vol. 3983 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 756–766.

[134] Youseff, L., Butrico, M., and Da Silva, D. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08* (2008), IEEE, pp. 1–10.

[135] Yu, H. Q., Liu, D., Dietze, S., and Domingue, J. Developing rdf-based web services for supporting runtime matchmaking and invocation. In *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on* (oct. 2011), pp. 392 –397.

[136] Yu, T., Zhang, Y., and Lin, K.-J. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web 1*, 1 (May 2007).

[137] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on 30*, 5 (may 2004), 311 – 327.

List of Tables

List of Figures

List of Acronyms

AI Artificial Intelligence

AHP Analytic Hierarchy Process

CaaS Composition as a Service

CSA Composite Service Architecture

DMTF Distributed Management Task Force

EMES Energy Monitoring and Efficiency System

IaaS Infrastructure as a Service

IoS internet of Services

MCDA Multi-Criteria Decision Aiding

MCDM Multi-Criteria Decision Making

NIST National Institute of Standards and Technology

OWL-S Semantic Markup for Web Services

PaaS Platform as a Service

QoS Quality of Service

RDF Resource Description Framework

SaaS Software as a Service

SAW Simple Additive Weighting

SOA Service Oriented Architecture

SOC Service Oriented Computing

SPARQL SPARQL Protocol and RDF Query Language

SWSF Semantic Web Service Framework

SWSL Semantic Web Service Language

SWSO Semantic Web Service Ontology

UML Unified Modeling Language

USDL Unified Service Description Language

WSDL Web Service Description Language

WSMF The Web Service Modeling Framework

WSML The Web Service Modeling Language

WSMO Web Service Modeling Ontology

XaaS Anything as a Service

Listings

List of Algorithms