

# **Cloud Computing Service Discovery Framework for IaaS and PaaS Models**

**Farzad Firozbakht**

Thesis submitted to the  
Faculty of Graduate and Postdoctoral Studies  
in partial fulfillment of the requirements for the degree of

**Master of Science in Systems Science**



uOttawa

University of Ottawa  
Ottawa, Ontario, Canada

November 2016

*In The Name Of GOD*

# Abstract

---

Cloud service discovery is a new challenge which requires a dedicated framework in order to approach it. Over the past few years, several methods and frameworks have been developed for cloud service discovery but they are mostly designed for all cloud computing models in general which are not optimal. The three cloud computing models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), each computing model has its own set of resources. Having one single discovery framework for all three is not very efficient and the implementation of such a framework is complex with lots of overhead.

The existing frameworks for cloud service discovery are mostly semantic-based and there are a few syntax-based frameworks that are using the filter by Attribute method as their solution. This research proposes a cloud service discovery framework focusing on IaaS and PaaS cloud computing models. Our framework is using a syntax-based query engine at its core and uses Extensible Markup Language (XML) for storing cloud service information. We eventually test the framework from the user point of view with IaaS and PaaS cloud services from real cloud service providers.

Such a framework could be a good solution for IaaS and PaaS since it is accurate enough for service discovery and easy to update.

# Acknowledgment

---

I would like to thank my supervisor, Dr. Bijan Raahemi for his supervision and guidance which motivated me throughout the time that I spent on this research. I also like to thank my good friend Waeal Obidallah, for his friendship, support, and supervision of this work.

I dedicate this thesis to my family who believed in me and supported me throughout my entire life.

# Table of Contents

---

<b>Abstract.....</b>	<b>iii</b>
<b>Acknowledgment.....</b>	<b>iv</b>
<b>Table of Contents .....</b>	<b>v</b>
<b>List of Figures.....</b>	<b>vii</b>
<b>List of Tables .....</b>	<b>viii</b>
<b>List of Acronyms .....</b>	<b>ix</b>
<b>Chapter 1. Introduction .....</b>	<b>1</b>
1.1. <i>Motivation</i> .....	1
1.2. <i>Thesis Objectives and Research Questions</i> .....	2
1.3. <i>Methodology</i> .....	3
1.3.1    Design Science Research methodology for Information Systems .....	3
1.3.2    Research Method and Steps.....	4
1.4. <i>Thesis Contributions</i> .....	4
1.5. <i>Thesis Outline</i> .....	5
<b>Chapter 2. Background and Literature Review .....</b>	<b>7</b>
2.1. <i>Background Study</i> .....	7
2.1.1    Web Service Discovery .....	7
2.1.2    Cloud Computing Definition .....	8
2.1.3    Introduction to Cloud Architecture.....	8
2.1.4    Cloud Service Models.....	9
2.1.5    Taxonomy of Cloud Services .....	11
2.2. <i>Related Work (Literature Survey)</i> .....	12
2.2.1    Semantic-based Discovery Framework .....	12
2.2.2    Semantic-based Cloud Service Discovery framework using query Processing Agent (QPA) .....	15
2.2.3    Similarity Measurement.....	17
2.2.4    Search-Based cloud Service Discovery .....	18
2.2.5    Algorithms for IaaS and PaaS Service Discovery .....	20
2.2.6    Concept-based Querying .....	22
2.2.7    Context-aware and syntactic matching methods .....	23
2.2.8    Syntax-Based query using XML and Ajax .....	23
2.2.9    Filter by Attribute .....	24

<b>Chapter 3. The Proposed Cloud Service Discovery Framework .....</b>	<b>28</b>
3.1. <i>IaaS and PaaS Service Discovery.....</i>	28
3.2. <i>Behavioral Design of IaaS and PaaS Cloud Service Discovery Framework ...</i>	29
3.3. <i>Structural Design of IaaS and PaaS Cloud Service Discovery framework.....</i>	31
3.3.1   Data Flow .....	32
3.3.2   End System .....	32
3.3.3   Query Engine .....	32
3.3.4   XML Files (Information Storage).....	33
3.3.5   Relational Database .....	33
3.3.6   Cloud Client and Cloud Service Providers .....	33
<b>Chapter 4. Implementation and Experiments .....</b>	<b>35</b>
4.1. <i>Work Flow Diagram of IaaS and PaaS Cloud Service Discovery Framework</i>	35
4.2. <i>Implementation Method .....</i>	40
4.3. <i>Generating Output .....</i>	41
4.3.1   Query Engine Testing .....	41
4.3.2   Experimenting from the user point of view .....	43
4.4. <i>Comparison.....</i>	46
4.4.1   Semantic-based method .....	46
4.4.2   Filter by Attribute method .....	47
4.5. <i>Summary of the Comparison.....</i>	48
<b>Chapter 5. Conclusions .....</b>	<b>49</b>
5.1. <i>A Summary of the Research .....</i>	49
5.2. <i>Contributions of the thesis .....</i>	49
5.3. <i>Publication.....</i>	50
5.4. <i>Limitations and Future Works .....</i>	50
<b>Appendix A: Source Code .....</b>	<b>51</b>
<b>References .....</b>	<b>57</b>

## List of Figures

---

<b>Figure 1</b>	DSR methodology in IS discipline [5, 2] .....	3
<b>Figure 2</b>	Simplified Cloud Infrastructure [15].....	9
<b>Figure 3</b>	Cloud Computing Models [16] .....	10
<b>Figure 4</b>	AWS Cloud Infrastructure and Cloud Services [17] .....	11
<b>Figure 5</b>	Cloud Services Taxonomy [18] .....	12
<b>Figure 6</b>	Architecture of Semantic Computing [22].....	13
<b>Figure 7</b>	Architecture of SSE [20].....	14
<b>Figure 8</b>	sample Semantic-based cloud service discovery system [24] .....	15
<b>Figure 9</b>	Cloud Service Discovery framework using QPA [25].....	16
<b>Figure 10</b>	Specificity cost property: implies that $Sim(C,D) > Sim(A,B)$ [26] .....	18
<b>Figure 11</b>	Architecture of Cloud Service Crawler Engine [28].....	19
<b>Figure 12</b>	PaaS Ontology for operating systems [30] .....	21
<b>Figure 13</b>	Technical coverage of semantic computing [22] .....	22
<b>Figure 14</b>	The architecture of an Ajax search engine [39]. .....	24
<b>Figure 15</b>	Sample XML used for Filter by Attribute.....	25
<b>Figure 16</b>	An example of Filter by Attribute method [40] .....	26
<b>Figure 17</b>	IaaS and PaaS core features [41].....	29
<b>Figure 18</b>	Behavioral Diagram of IaaS and PaaS Service Discovery Framework .....	30
<b>Figure 19</b>	Structural Design of IaaS and PaaS Cloud Service Discovery framework .....	31
<b>Figure 20</b>	Cloud service models and their main access tools [42] .....	34
<b>Figure 21</b>	IaaS and PaaS Cloud Service Discovery Workflow Diagram .....	36
<b>Figure 22</b>	Transition from XML Data to DOM Tree .....	37
<b>Figure 23</b>	Using AJAX without DOM API [43] .....	38
<b>Figure 24</b>	Using AJAX with DOM API [44] .....	39
<b>Figure 25</b>	Similarity far from an exact match .....	41
<b>Figure 26</b>	Similarity close to match the query from the user .....	41
<b>Figure 27</b>	Input getting an exact match from the query engine.....	42
<b>Figure 28</b>	Input with no suggestions from the query engine .....	42
<b>Figure 29</b>	Using keyword “Data” as our generic query .....	43
<b>Figure 30</b>	Using keyword “CPU” as our generic query .....	44
<b>Figure 32</b>	is an example of searching for a specific service, accurate query results for accurate suggestions. ....	44
<b>Figure 31</b>	Searching for an actual IaaS cloud resource .....	44
<b>Figure 32</b>	Searching for a specific service Provider, Example 1 .....	45
<b>Figure 33</b>	Searching for a specific provider, Example 2 .....	45

# List of Tables

---

**Table 1** Comparison Summary.....48



## List of Acronyms

---

<b>Acronym</b>	<b>Definition</b>
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
SaaS	Software as a Service
DSRM	Design science research methodology
IS	Information System
UDDI	Universal Description, Discovery, and Integration
WSDL	Web Service Definition Language
OWL	Web Ontology Language
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
AWS	Amazon Web Services
EC	Elastic Computing
SQL	Service Query Description Language
SCDL	Service Capability Description Language
SNL	Structured Natural Language
SSE	Semantic Search Engine
QPA	Query Processing Agent
CSCE	Cloud Service Crawler Engine
WADL	Web Application Description Language
XML	Extensible Markup Language
DTD	Document Type Definition
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
DOM	Document Object Model
XSD	XML Schema Definition
WAMP	Windows, Apache, MySQL, and PHP

# Chapter 1. Introduction

---

Cloud service is a set of applications that are delivered as hardware and software to a remote user; it also enables virtualization, distributed storage, distributed databases, and monitoring systems. In the cloud architecture every software application or hardware becomes a service or part of a service and to add a new service to the cloud some adjustments are need to be done in order to make the new service compatible with the architecture. Several protocols are used in order to provide these services to the user depending on the type of the service. The straight forward way is to have ontology for all three categories of services however; in the existing methods ontology is used for semantics which works best with SaaS since the number of services grow much faster than PaaS and IaaS over time and there are more dependencies between the services in SaaS. This thesis contributes a Syntax-based cloud service discovery framework which is optimized for IaaS and PaaS.

This chapter presents research motivation, objectives, and methodology.

## 1.1. Motivation

There are many IT infrastructures that are providing services on the Internet, however, the difference between a cloud and a traditional IT infrastructure is the way that these services are provided. In the cloud, all the services are provided over a cloud management interface layer. In this architecture, it is important to assign the user to the right services based on the queries. There are many ways to assign these services and some algorithms have been developed, however, the service allocation is different based on the types of services that are provided in a cloud system. In the case of SaaS, the same methods that are applied in web service discovery can also be applied. For IaaS and PaaS the method changes since the range of the services that are being provided are much more limited and they take more advantage of the cloud architecture [1].

The purpose of this research is to provide an optimized framework for IaaS and PaaS which has the least overhead and the most accuracy in cloud service discovery based on the user query and resource allocation.

## 1.2. Thesis Objectives and Research Questions

The objective of this thesis is to develop a service discovery framework specialized in discovering the resources for IaaS and PaaS in the cloud. We are looking for an optimal way to discover these services and give suggestions to the user based on the queries. It is important for us to have a framework that has enough accuracy in cloud service discovery for IaaS and PaaS models while being easy to update the services over time. The currently used mechanisms in cloud service discovery are too complex for IaaS and PaaS and have a lot of overhead since they are designed for all three computing models. This research is targeting towards a framework that can take benefit from a better discovery solution for IaaS and PaaS cloud computing models.

In this context, this thesis investigates the following research questions:

**RQ1:** Could the proposed framework be a better solution than the existing ones and having the potential to be a replacement?

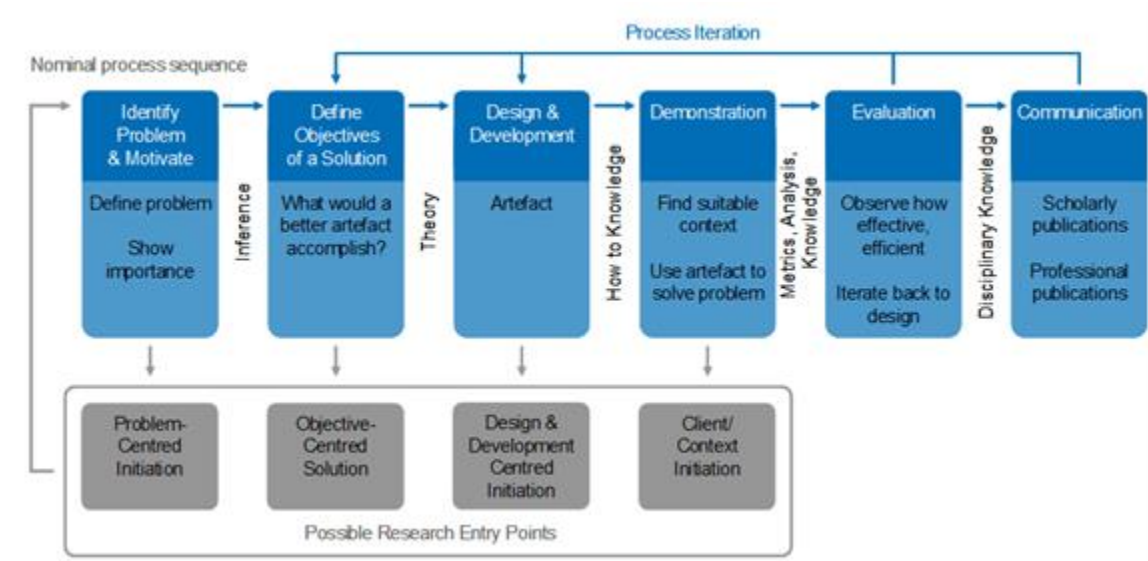
**RQ2:** What type of query engine can be used for finding IaaS and PaaS resources so that the framework can be accurate but less complex than the existing cloud service discovery frameworks?

To answer these questions, we take the implementation technology for each component of the framework seriously while developing the actual framework component so that we can build a prototype and run some tests. For our testing, we use scenarios that are close to reality as our proof of concept.

## 1.3. Methodology

### 1.3.1 Design Science Research methodology for Information Systems

The Design Science Research methodology for Information Systems “creates and evaluates IT artifacts intended to solve identified organizational problems [2].” In this methodology, it is important to consider the relationship between the nature of the artifact, objective and problem as something separate from the contribution of the DSR study [3]. The development of an artifact within a DSR research should be a search process that draws from existing theories to come up with a solution [2]. The output of this methodology is in the form of a framework which serves as a support to perform a specific task [4]. Figure 1 is a diagram based on [2].



**Figure 1** DSR methodology in IS discipline [5, 2]

This method is in six phases, the first phase defines the problem in order to justify the value of the proposed solution. The second phase defines the objectives for a solution. These objectives could be a replacement for an existing solution or something that adds to it. The third phase identifies the artifact, its architecture and how it functions. The fourth phase demonstrate the artifact in different scenarios to see if it actually solves one or more instances of the problem. The fifth phase focuses on how well the artifact contributes to solving the problem. This can be done by testing the proposed artifact and

comparing with some existing solutions introduced in phase two. The sixth phase is for summarizing the results and publishing the contribution [2, 4].

### **1.3.2 Research Method and Steps**

With the help of DSR methodology in IS, we are performing our research activities in six phases, the details of these phases are as follows:

- Phase1 is where we do the literature survey and go through the fundamentals of the cloud computing and cloud service discovery system, in this phase we introduce the available techniques for finding cloud computing resources and point out the area in cloud service discovery that can be improved.
- Phase2 introduces the components of our proposed framework and what they can achieve.
- Phase3 goes through the technologies that are needed to develop the framework considering the existing challenges from the server-side of a service discovery framework.
- Phase4 is where we model our prototype and implement the query engine based on syntax. We also store real world data for our evaluation and build a virtual server on our local device in order to run the server-side code.
- Phase5 focuses on building scenarios that are close to reality for testing purposes. We run a bunch of tests and check our results to see if the framework is working properly and it is accurate enough otherwise, we make some improvement to the code running on the client-side which forces us to go back to phase two. Eventually, we do a comparison and evaluate our framework.
- Phase6 is the summarization and publishing the resulted knowledge in conferences.

## **1.4. Thesis Contributions**

In this research, we introduce a framework which is based on IaaS and PaaS cloud computing models and a syntax-based query engine which is implemented in order to give

user either the exact match or a similar result, based on the inputs. This method will be prototyped and demonstrated to see if the framework is working properly. There are many syntax-based functions that can be used for the query engine but we will use and modify the one that can take advantage of every component of the proposed framework properly.

## 1.5. Thesis Outline

The structure of the thesis is as follows:

- Chapter 2 presents the literature review that is related to research questions. First, the fundamentals of web service discovery are being studied and then the algorithms that are used for service discovery are introduced. Since the cloud architecture is derived from Grid computing architecture, the similarities and differences between them will be discussed. After that, we move on to cloud architecture, similarities between discovering web services and cloud services and discussing which methods in web service discovery can also be applied to cloud services.
- Chapter 3 introduces the proposed framework and the relationship between its components, it also explains the query engine, its dependencies and how it takes advantage of the proposed framework. Assumptions and requirements for the framework are also presented in this chapter.
- Chapter 4 focuses on the implementation of the query engine on the proposed framework, the tools, and standards that were used to make this implementation possible and building the prototype. In this chapter, we also compare our solution to an existing method and point out the pros and cons of each solution in different scenarios.
- Chapter 5 (conclusion) is the final chapter of the thesis which explains the contribution of the thesis and the results of the comparisons and experimentations of different methods in cloud service discovery for IaaS and PaaS. We

also go through a brief discussion about the research limitations and future works.

## Chapter 2. Background and Literature Review

---

This chapter introduces the cloud service discovery concepts, methods, and algorithms that have been previously implemented and are related to this research. First, we start this chapter with a brief introduction to web services and the methods that are used for web service discovery and then we go through fundamentals of cloud and methods for cloud service discovery.

### 2.1. Background Study

#### 2.1.1 Web Service Discovery

Every service that requires machine-to-machine interaction over a network is considered as a web service [6]. Policies help to deliver these services, based on W3C a policy is a constraint on the behavior of agents or people or organizations [7]. The agent in this context is a piece of software or hardware that sends and receives messages and the services are the resources that are provided by the service providers.

Currently, there is no general method for web service discovery but there are some that are widely used based on the type of service that is being provided. The three major methods that are used in web service discovery are as follows:

- The semantic-based method which requires web services to have associated semantic descriptions. Using ontology is a common approach towards semantic based discovery method [8].
- The syntax-based method which relies on the user query. These queries are organized into terms and operators [9].
- The hybrid method which is a combination of both semantic and syntax based with or without ontology. This method is widely used in search engines since the generated results are based on web crawling algorithms [10].



### **2.1.2 Cloud Computing Definition**

There are several definitions of cloud computing, the briefest definition according to Reference [11] is as follows:

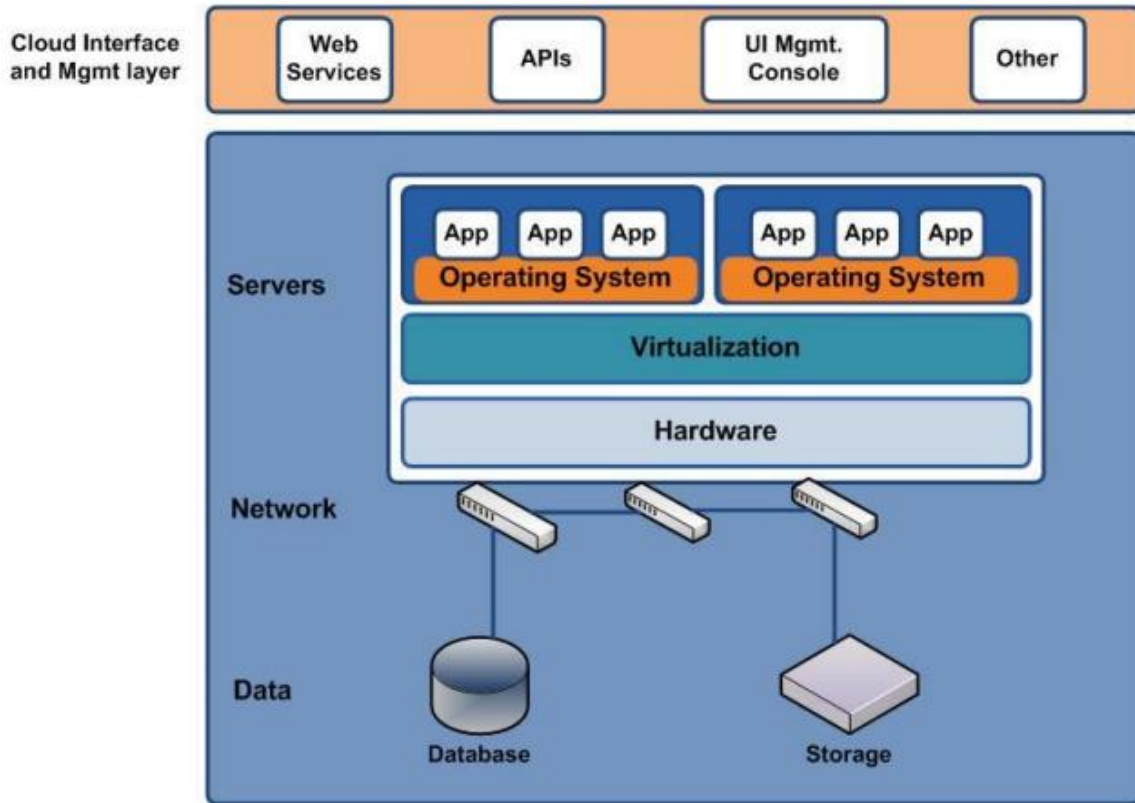
“Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services.”

With help of cloud computing, resources can be delivered to users in a manner similar to traditional utilities [12]. There are three types of computing models, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [11, 13]. The range of services for SaaS is much broader than IaaS and PaaS, for that reason, IaaS and PaaS are considered together [11].

### **2.1.3 Introduction to Cloud Architecture**

The idea of cloud computing comes from Grid which was originally designed for heavy computational tasks within organizations and was not available as a service. Cloud architecture has an elasticity which means it can be scaled up on demand, however, Grid was built with many resources up front.

The core of the cloud architecture is the database, network and hardware resources. In order to make the available services accessible to the user, virtualization is required, major benefits of virtualization are energy efficiency and improved resource utilization [14]. The cloud architecture provides a centralized computing service over a network [15]. PaaS delivers a prebuild application or development platform to the client; it scales up with respect to its infrastructure. IaaS provides infrastructures to the client such as virtual machines, storage, networks, firewalls and load balancers [11]. For IaaS virtualization is not necessary since upscale and downscaling it does not depend on it but for PaaS and SaaS virtualization is required since it provides elasticity and organizations can quickly upscale and downscale their resources. Simplified cloud architecture is shown in Figure 2.



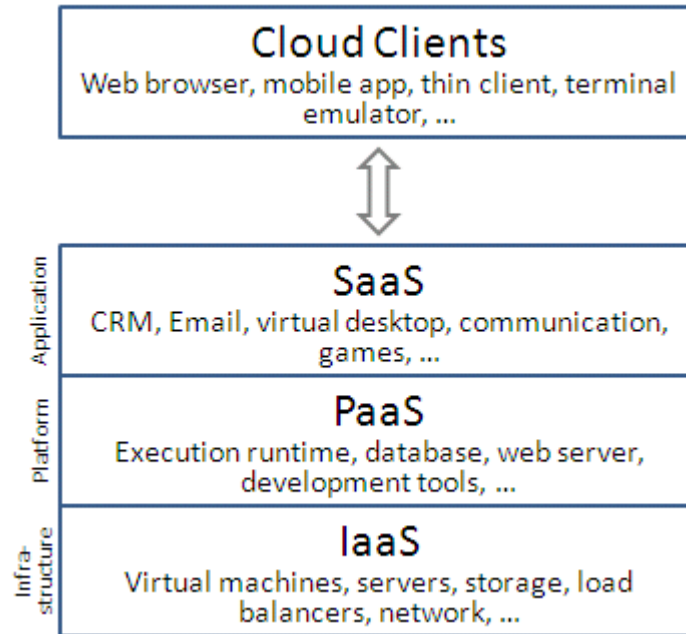
**Figure 2** Simplified Cloud Infrastructure [15]

As it is shown in Figure 2 there are two layers in typical cloud architectures: the first layer is the service layer which contains the Data, Network, Servers and virtualization on top of all of that in order to achieve scalability. The virtualization is managed by the internal system administrators and by default it does not provide the abstraction layer that enables cloud services. The abstraction layer as it is shown in Figure 2 is the layer on top, this layer is essential in any cloud architecture since it hides the complexity of the infrastructure and provides users to manage their cloud service lifecycle, as well as providing the interface so that users can have access to the services [15].

#### 2.1.4 Cloud Service Models

The cloud provides the same technologies as any other IT infrastructure, the difference is that everything in the cloud is provided as a service. A cloud service model describes how cloud services are available to the users. There are dependencies between the models; platforms are operating systems and development environments that are installed on

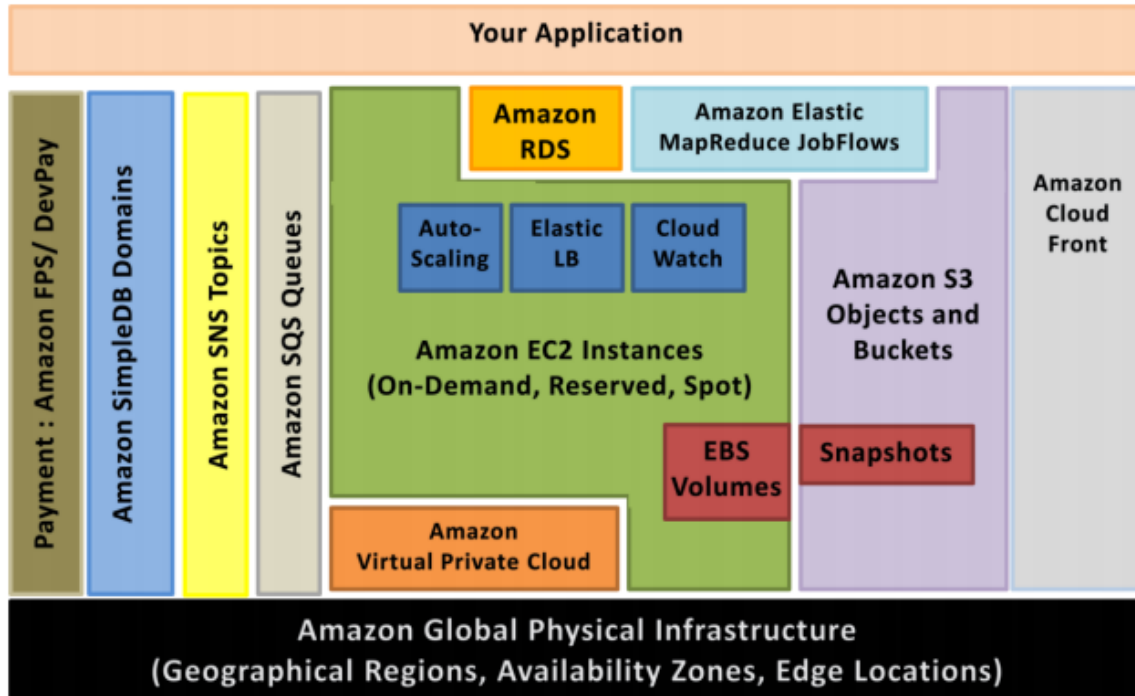
hardware resources which make PaaS dependent on IaaS. Applications that are provided as a service are dependent on the platforms; this makes SaaS depend on PaaS. The diagram below shows how these three cloud service models interact with each other:



**Figure 3** Cloud Computing Models [16]

As it is shown in Figure 3 cloud services in terms of layers are dependent. Services in SaaS are dependent on their platforms and the platforms are dependent on the hardware resources that are available.

Amazon is one of the largest cloud service providers; Figure 4 shows the cloud infrastructure and the available cloud services that are being provided by Amazon Web Services (AWS).

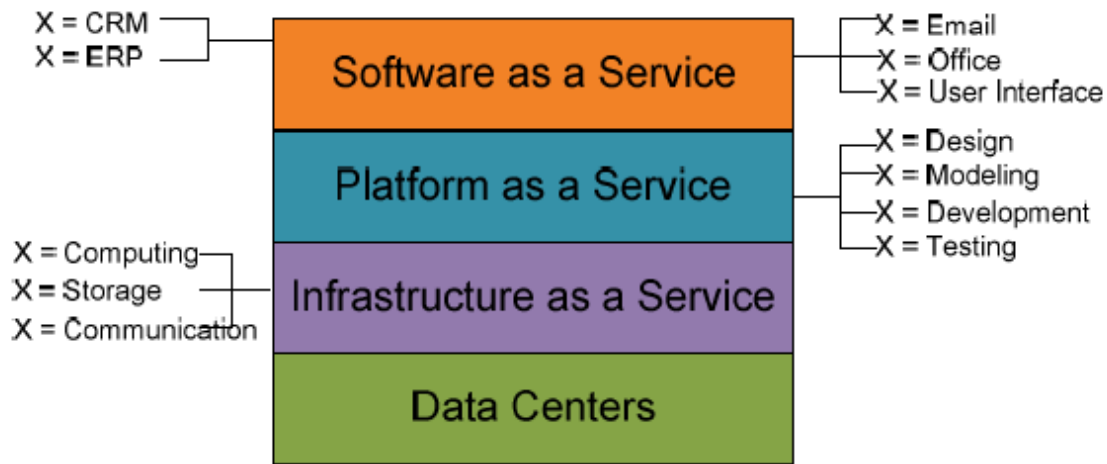


**Figure 4** AWS Cloud Infrastructure and Cloud Services [17]

One of the services in AWS is Amazon EC2 Instances; it allows users to manage virtual machines that are running operating systems. Amazon Relational Database Service (RDS) provides users with MySQL and Oracle database services [17]. Amazon S3 is a fast cloud storage that allows users to store large amounts of data as objects from anywhere using HTTP. There are more services that AWS is providing; all of them have one thing in common which is the infrastructure, it needs to be compatible with the services that are being provided. In the case of AWS services are using a single infrastructure called Amazon Global Physical Infrastructure which is using geographical regions, availability zones, and edge locations as its parameters to provide the services.

### 2.1.5 Taxonomy of Cloud Services

As mentioned previously there are three major cloud service models (SaaS, IaaS, and PaaS). Of course, other service models can be found including the major three however, that depends on the provider to extend the model if they wish to. The three major models form the top levels of taxonomy since they are the fundamentals of cloud services. Figure 5 shows the cloud services taxonomy.



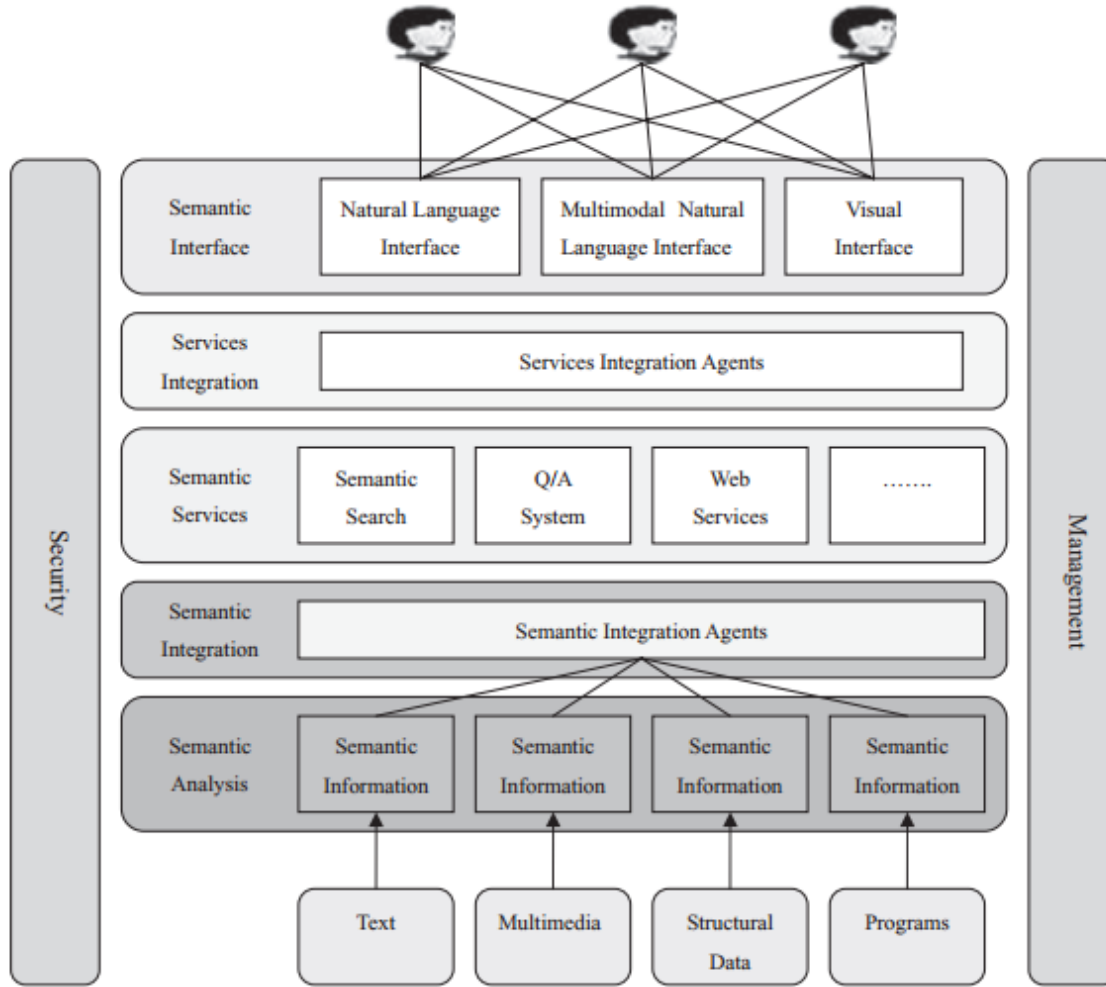
**Figure 5** Cloud Services Taxonomy [18]

Public cloud services, as it is shown in Figure 5, is a set of computing resources provided by third-party organizations like google or Amazon. There might be other cloud service taxonomies which are being used only inside organizations but we are not covering them since the framework provided in this thesis is based on Public cloud services [19].

## 2.2. Related Work (Literature Survey)

### 2.2.1 Semantic-based Discovery Framework

Semantic computing is a combination of software engineering, artificial intelligence, database and computational linguistics. Unlike traditional computing, semantic computing is being used to extract or process the content and semantics of passive data and active processes, search engines that are using this method are very effective with resource-intensive services [20, 21]. Figure 6 is the general architecture of semantic computing, which is being used as the fundamental model to develop a service discovery framework similar to a search engine.

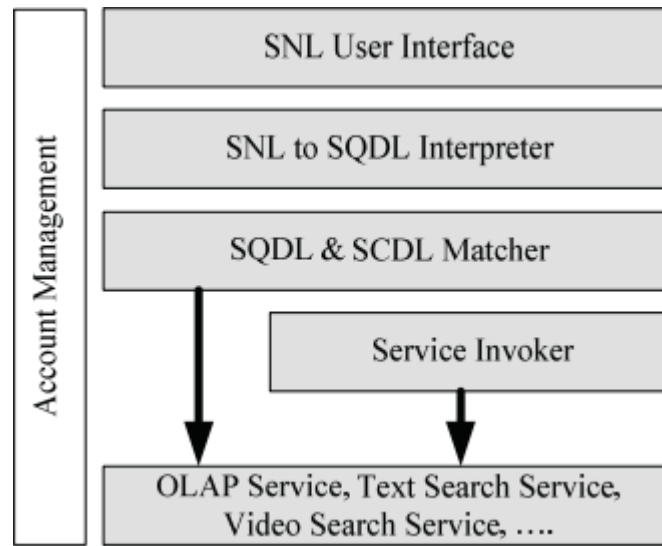


**Figure 6** Architecture of Semantic Computing [22]

As it is shown in the architecture, the search engine analyzes and converts signals such as pixels and words (content) to meanings (semantics); it also provides the information resources for Semantic Integration and Semantic Applications [20]. With the help of this architecture, a semantic service search engine can be developed. It works similar to a regular search engine however; the focus is on cloud services only. The model provides the user with an interface in order to search for services, the query is translated into service query description language (SQDL) which processes the query and enables it for further processes [20]. There is also another language used in this model called service capability description language (SCDL), which is for describing the capability of each service [20]. A matcher is being used in order to do the comparison between the SQDL and SCDL, then if the user finds the desired service it uses the Service Invoker that gets the

final results from the matching of SQDL and SCDL. There is an account management layer that has control over all steps since it provides access control and charge function which is responsible for the payment system.

Using semantic computing users are provided with an interface such as a search engine or a query engine, with the help of SQDL and SCDL users can find their desired services. One of the simplest forms of semantic computing in a service discovery framework is Semantic Search Engine (SSE). Figure 7 shows the architecture of SSE.

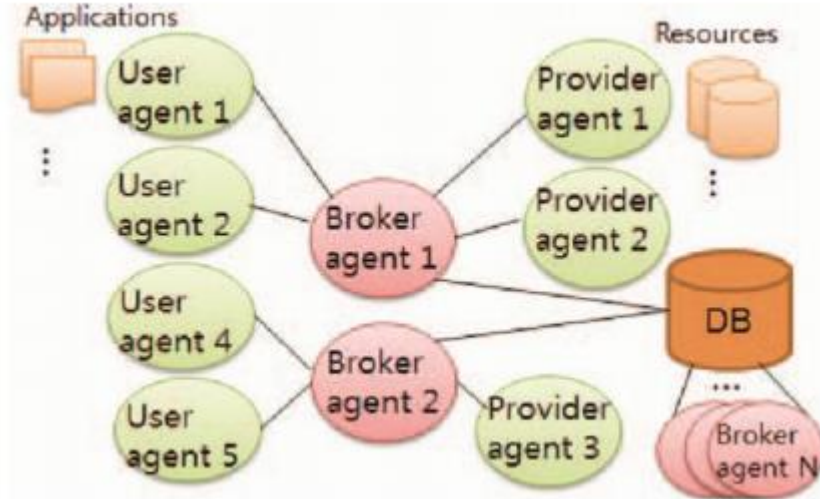


**Figure 7** Architecture of SSE [20]

The SNL user interface provides the user with a query interface. SNL to SQDL Interpreter translates SNL to machine processable query language in order to use it for SSE. The SQDL and SCDL matcher try to find the match between the user query and SCDL which has descriptions of the available services. If the matcher cannot find a match, it breaks down the SQDL query to simpler queries and tries to find several services that are close to the user request [20].

Service invoker invokes the matched service from the user side; this step in SSE is specifically designed for small services. The account management deals with the payment system and provides a charge function [23].

A search engine or query engine works best in a cloud discovery framework if it is integrated into a framework. Figure 8 is an example of a semantic-based cloud service discovery framework.



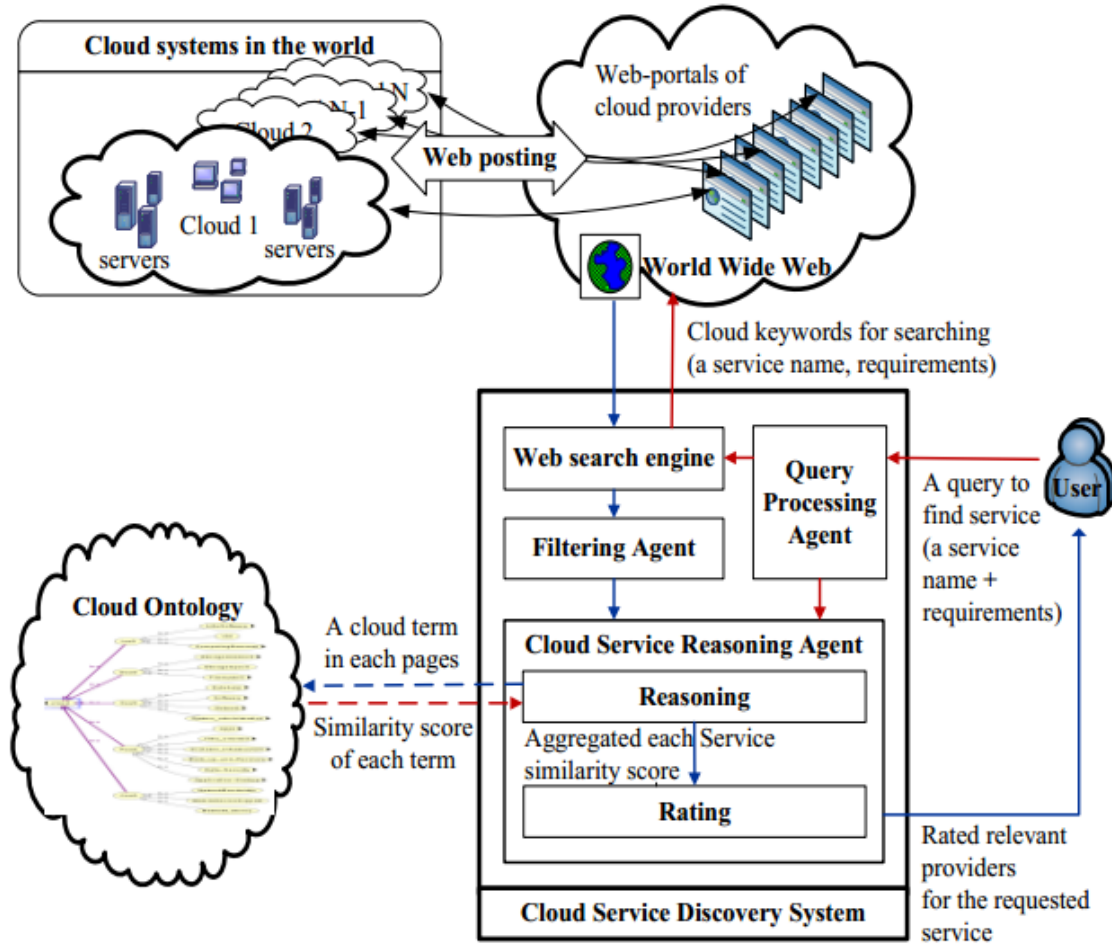
**Figure 8** sample Semantic-based cloud service discovery system [24]

The framework uses brokers in order to allocate the services from the service providers and with matching and comparison which will be performed by SSE, presents the closest match to the user query. Every time a request is made; a broker agent is dedicated to that request in order to allocate the related services from the available service providers.

### 2.2.2 Semantic-based Cloud Service Discovery framework using query Processing Agent (QPA)

This service discovery framework consists of a search engine and three different agents, Query Processing Agent, Filtering Agent and Cloud Service Reasoning Agent CSRA [25]. Figure 9 shows how the model works.





**Figure 9** Cloud Service Discovery framework using QPA [25]

This model is more effective than the semantic computing solution since it uses an ontology which consists of the taxonomy of different types of cloud services. By applying ontology we can get more accurate results based on user queries. As it is shown in the figure:

- The Query Processing Unit QPA executes the query and generates alternate results if the number of generated search results is less than what the user requested.
- The filtering agent filters the contents based on the keywords entered by the user.
- The Reasoning agent is used to determine the similarities between the services that are available and user requests.

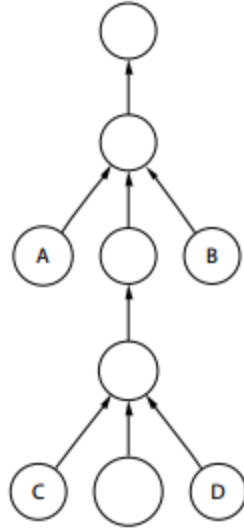
There are three methods that are being used by the Reasoning agent (Similarity reasoning, Equivalent reasoning, and Numerical reasoning). There is also a rating system which ranks the results with respect to user queries; the highest rated result is the best service for the user [25]. Agent-based method unlike the semantic computing method is focused much more on cloud services discovery than service discovery in general and therefore it is more optimized if we use a similar approach to develop a service discovery framework in the cloud.

### 2.2.3 Similarity Measurement

There is not a unique way for similarity measurement, however; we are looking for the one that is the most suitable for service discovery for PaaS and IaaS. The similarity is derived from a vaguely defined notion of how much concepts have in common [26]. The objective is to derive a function  $\text{sim}(x, y)$  to measure how much the concepts  $x$  and  $y$  share or how close they are [26]. A second approach also exists however it is optional since it is not implied by the semantic of the ontology, It measures the values of two functions  $\text{sim}(A, B)$ ,  $\text{sim}(C, D)$  and gives them priority based on their values. This property is the Specificity cost property which is defined as follows:

*“The intuition for this property is that the similarity between for instance siblings on low levels in the ontology as “alsatian” and “poodle” should be higher than the similarity between siblings close to the top as “Physical” and “Abstract”. This idea corresponds to the notion of information content described in (Resnik, 1998) [26].”*

The entire similarity operation falls into the category of query processing. A concept needs to be compared first and then derive a similarity between every other concept in the database. Cloud ontology is generative due to the scalable characteristic of the cloud; therefor the algorithm that is being used for similarities should also have this property in order to support the cloud infrastructure [26]. The defined ontology is the basis of the comparison, it is still possible to have a result based on the comparison in the absence of ontology but we will have to use a more general method which it will not be optimal. Figure 10 shows the similarity between four objects.



**Figure 10** Specificity cost property: implies that  $Sim(C,D) > Sim(A,B)$  [26]

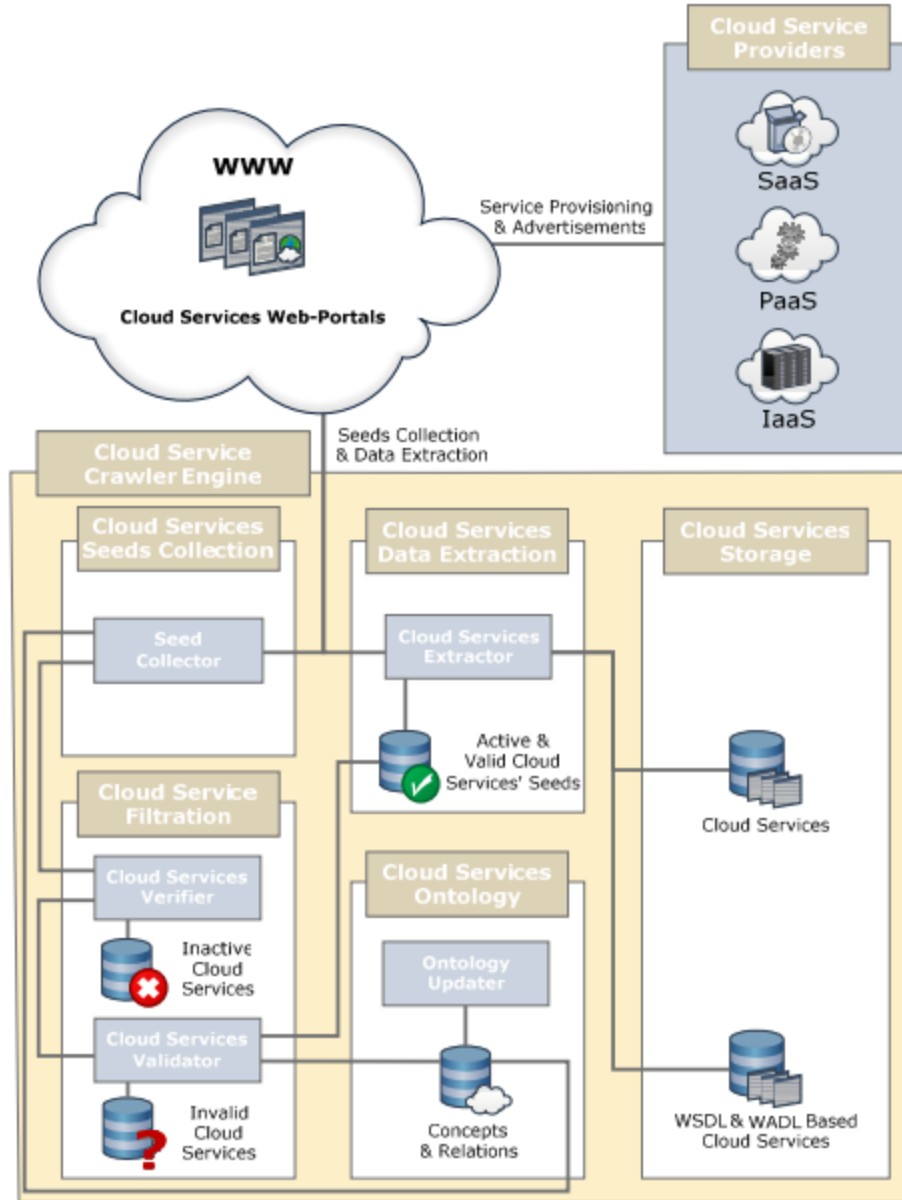
This method is suitable to compare queries and find similarities rather than finding an exact match between the query and available resources. Ontology causes more restrictions and less expression which are more suited when it comes to comparing queries and services in a specific category.

This best way for service to have associated semantics is using an ontology, however; there might be complex relationships between the services (entities). Usually, complex relationships are based on property sequences of the entities, these relationships are represented as paths spanning across multiple domains that are also represented by ontology since it is the only way to measure their similarities and helps to rank the entities. The ranking of the entities is based on relevance, specificity and the span of relationships [8, 27].

#### 2.2.4 Search-Based cloud Service Discovery

One of the methods to search for cloud data and later providing them as service is crawling. A crawler analyzes and collects relevant data from the web by parsing links [28].

A sample crawler engine with six layers is shown in Figure 11:



**Figure 11** Architecture of Cloud Service Crawler Engine [28]

As it is shown in the figure each layer is responsible for a specific task of analyzing the retrieved data.

- **Cloud Service Provider Layer** consists of cloud providers who are broadcasting their services; these are accessible from search engines.
- **Cloud service Ontology Layer** maintains the cloud service ontology, this ontology contains the relationships between the entities that help the crawler engine to discover and validate the available services.

- **Cloud Service Seeds Collection Layer** contains the URL of cloud services as well as WSDL and WADL documents.
- **Cloud Services Filtration Layer** filters the seeds collected from the seed collector; it uses some of the concepts in the ontology as keywords to collect data.
- **Cloud Services Data Extraction Layer** extracts the cloud service for active and valid services and later stores them in cloud service database for later analysis.
- **Cloud Service Ontology** provides the crawler engine with Meta information and describes the relationships and semantics of cloud services.

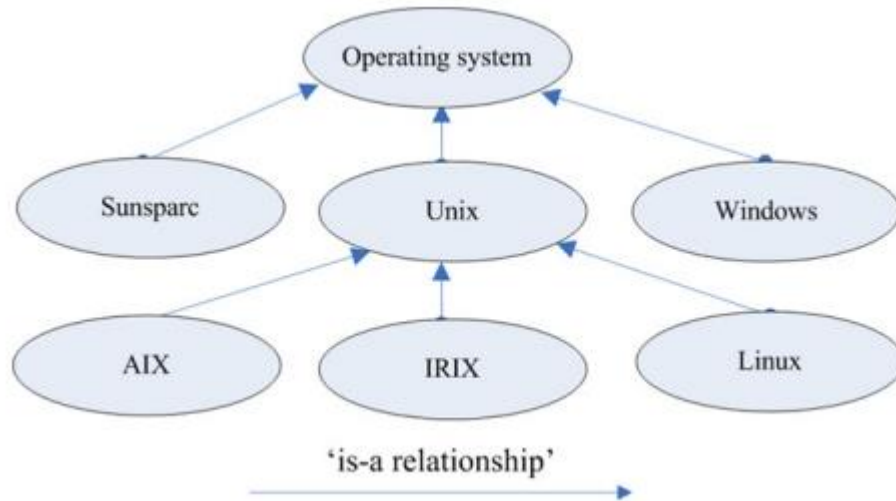
### 2.2.5 Algorithms for IaaS and PaaS Service Discovery

There are many algorithms that can allocate services available on the web, however, a few of them are suitable to be used in cloud architecture since the resources are scalable and change dynamically. One of the major issues in cloud computing is effective discovery and allocation of the resources.

As mentioned before ontology is one of the ways for semantic-based discovery frameworks. It is a set of concepts and the relation between these concepts which provides us with Metadata especially in the area of semantic web; it also specifies the communications between the broker agents and resources. Basically, ontology makes it easy to look for semantic relations since each service might be composed of several other services in the cloud [29]. The concepts can be either equivalent or inclusive. A similarity function is necessary to determine the relation between the user query and the available resource that is going to be broadcasted by the brokers. In a cloud architecture, ontologies are hierarchical and for that reason, the similarity function always takes two values  $\text{Sim}(x, y)$  [30, 31]. The similarity function is defined as  $\text{sim}(x, y) = \rho [|\alpha(x) \cap \alpha(y)| / |\alpha(x)|] + (1 - \rho)[|\alpha(x) \cap \alpha(y)| / |\alpha(y)|]$  where  $\rho$  determines the degree of influence of generalization, it also allows to tailor the similarity function, and can thereby comply with the generalization property [30].  $\alpha(x)$  is the set of nodes reachable by  $x$  and  $\alpha(y)$  is the set of nodes reachable by  $y$ ,  $\alpha(x) \cap \alpha(y)$  is the reachable nodes shared by  $x$  and  $y$  which indicates what is common between  $x$  and  $y$ .

In case of PaaS if we want to apply the similarity function first we have to define ontology for the available platforms, since most of the platforms have at least one similarity we can define a hierarchical ontology and form a semantic similarity matrix  $SIM(n \times n)$  where each row and column represents the semantic description of resources and each element represents the degree of semantic similarity based on predefined concepts of domain ontology between every pair of resources, the resource with the most similarity has the value of “1” and if the value is between “1” and “0” the resource value closer to “1” will be selected as the final result for the user query [30].

Since the service is in the cloud architecture, it is important to make sure that the algorithm can perform in a dynamic environment. Figure 12 shows how a similarity function can be used in case of PaaS.



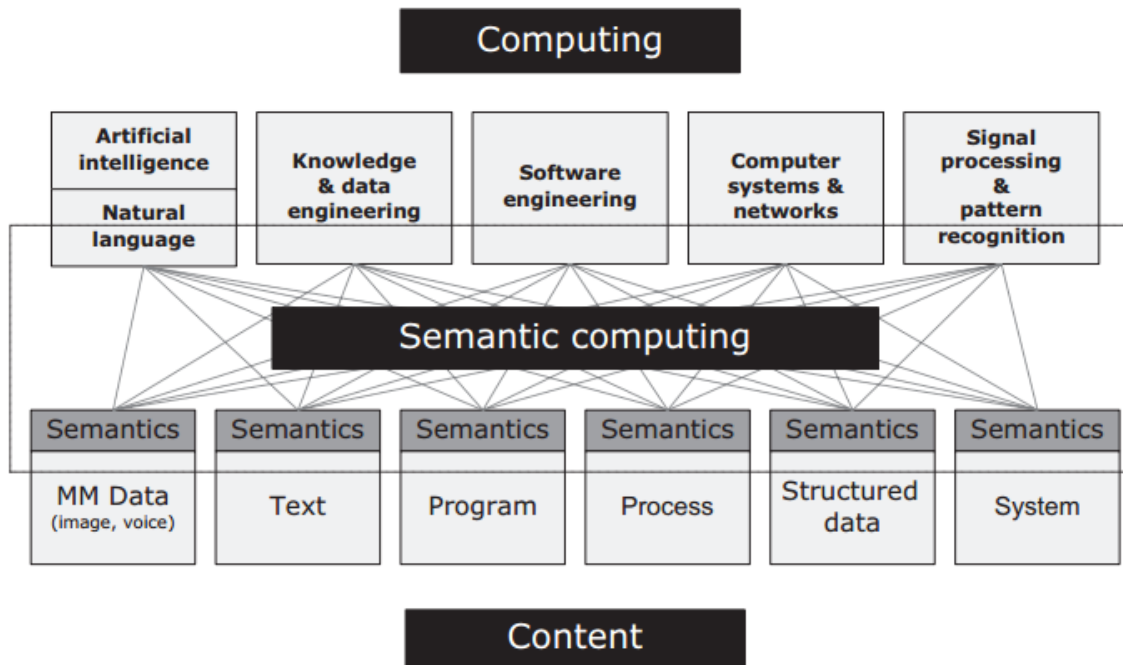
**Figure 12** PaaS Ontology for operating systems [30]

Each of the circles in the ontology is treated as nodes and the arrows represent the relationship between the nodes. The nodes Windows, UNIX and Sunsparc are all operating systems so they are connected to the node operating system. Linux, IRIX and AIX are sharing a kernel that is similar to UNIX and for that reason, they are connected to the node UNIX. Based on these relations we can create the similarity matrix and apply it to our similarity function. Using Ontology for similarity based functions has its disadvantages such as time-consuming development and having enough accuracy to have good performance [32].

### 2.2.6 Concept-based Querying

For service discovery on the web, ontology is one of the solutions to define major concepts. Later we use these concepts to describe the semantics of different objects; this is a typical approach in almost every service discovery algorithm.

This method like the previous algorithm is semantic-based however, it utilizes knowledge from a domain-specific ontology to obtain better answers on a semantical level and as a result, it gives us a better solution since concepts are the only thing that is being considered for comparison rather than actual words [31]. The ontology is being used to define and relate the concept and a concept language is used in order to express the semantics of queries and objects in the information base [31, 22, 33]. We also use algebra for integration, formalization, representation and reasoning with the semantics of natural language and ontologies [31]. Figure 14 represents the technical coverage of semantic computing.



**Figure 13** Technical coverage of semantic computing [22]

Another major issue in implementing ontology is the communication between different entities, not all entities can be related to each other and we cannot create the ontology that

contains all the entities and the relationships between them, instead we have multiple ontologies that are separate.

The mapping between ontologies is a possible solution since it is scalable and can be automated since cloud service change over time and need to be updated [22]. This solution also allows us to have more accurate query results since mapping has the advantage of creating relationships between ontologies from different cloud service models and also provides ranking between available services [27].

### **2.2.7 Context-aware and syntactic matching methods**

A search or query engine uses an algorithm that monitors the information and recommends the user with the best result based on the query [34]. Cloud services providers use their own service description which makes it hard to search for services due to non-uniform and non-standardized naming conventions, this approach makes it difficult for users to find the right service that they are looking for [35]. Syntactic matching is a straight forward solution that avoids some complexity simply by having good performance in most scenarios and being straightforward to develop. XML is used to store data, the user query is taken as a string and it is being compared with the XML file using hash table [36, 37]. The result is either an exact match or suggestions that are similar to the query.

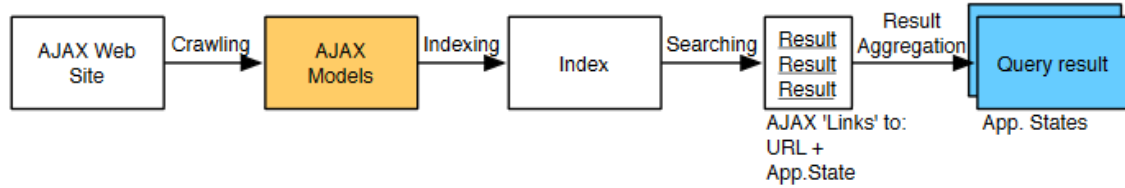
### **2.2.8 Syntax-Based query using XML and Ajax**

XML is a widely used standard for data representation that can be used as a data modeling language or a document markup language. It permits tagged text, element nesting and element references, these features of XML allow us to process queries [38]. With the help of XML, we can create paths and make the query processing easier.

Asynchronous JavaScript and XML (Ajax) is a well-known client-side scripting language that can help us with querying. It allows us to search through the data without having to refresh the web page by running client code in the browser. Since Ajax is using JavaScript, it reacts to user events like click, keywords, mouseover, etc. Another advantage of Ajax is its capability to access XML data which makes a syntax-based query possible and



allows the user to select the appropriate result [39]. The figure below shows how an Ajax search works.



**Figure 14** The architecture of an Ajax search engine [39].

As it is shown in Figure 15, we need the Ajax model to start the indexing. This is possible with the use of Document Object Model (DOM), it helps the query processing by building a tree from the XML which each node is considered an object. The index contains the information about the documents which happens to be the keywords. In this architecture, the user enters a keyword and gets the results which could be either an exact match or something similar according to the keywords entered [39].

In the case of having multiple XML files we need to build multiple DOM trees in order to perform multiple indexing, the rest is same as having only one XML file. This is still possible with one Ajax website in order to search through the data and show the results.

### 2.2.9 Filter by Attribute

This method is syntax-based however; it is not a query engine. Filter by attribute uses XML to eliminate the unrelated tags and select the ones that remain. The XML that is being used in this method needs to have attributes for every tag. Figure 15 shows a sample XML that can be used for this method.

```

1  <pagemap>
2  <DataObject type="document">
3      <Attribute name="keywords">PaaS</Attribute>
4      <Attribute name="keywords">IaaS</Attribute>
5      <Attribute name="keywords">SaaS</Attribute>
6  </DataObject>
7  </pagemap>
8  </page>

```

**Figure 15** Sample XML used for Filter by Attribute

As it is shown in this figure every tag has its own attribute which later on will be used for elimination to find the exact match. Although this method is not as accurate as a syntax-based search, it is widely used as a syntax-based solution for cloud service discovery. Filter by Attribute method is unable to suggest similar results and leaves users with no other option if the exact match does not exist. The figure below is an example of using the Filter by Attribute method.



As it is shown in figure 16 with Filter by Attribute the user cannot enter a query, the only option is to check the attributes that are available and the system will filter the available list based on the selected options and leaves the rest. This method is absolutely inefficient since it forces the user with whatever that is available and has no suggestions system.

## Chapter 3. The Proposed Cloud Service Discovery Framework

---

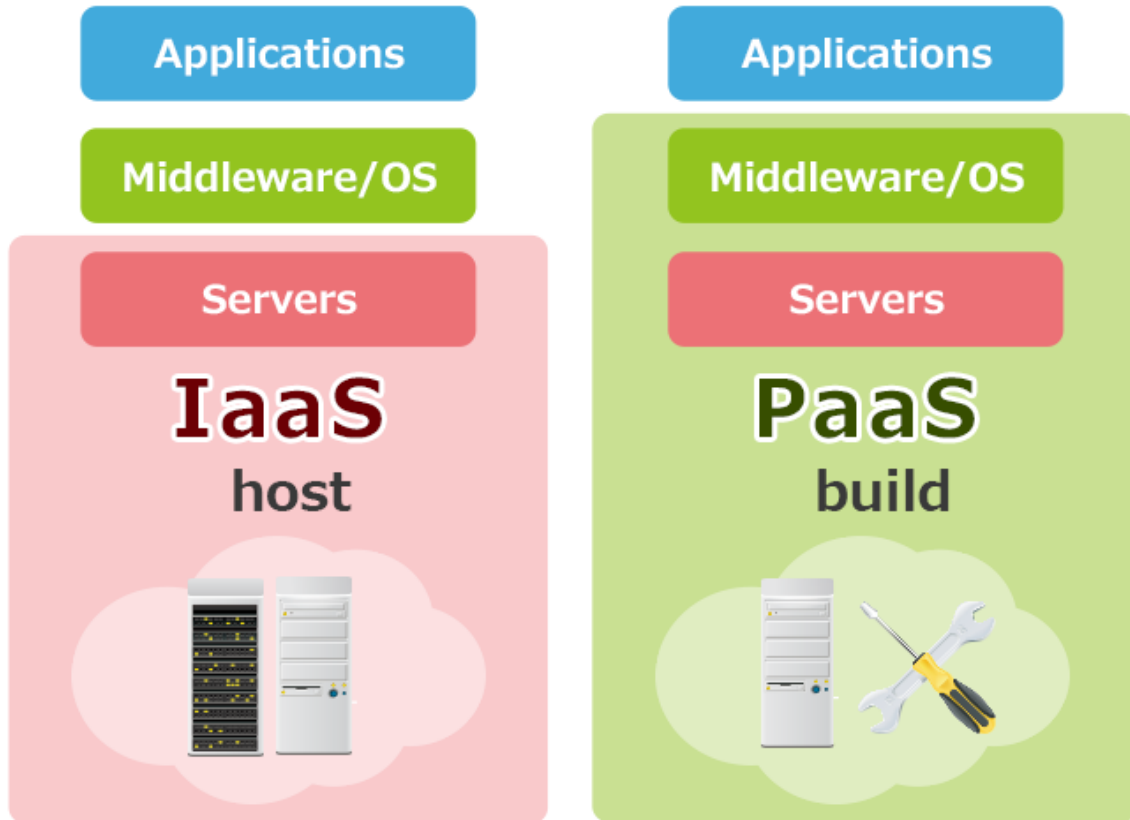
In this chapter, we propose a framework for cloud service discovery with a focus on PaaS and IaaS cloud computing models. First, we begin with introducing the behavioral model of the framework which explains the way our discovery framework functions, then we introduce the proposed framework and explain all of its components and how they interact with each other and in the next chapter, we go through the implementation and testing.

### 3.1. IaaS and PaaS Service Discovery

Before designing any frameworks the first thing we need to focus on is the result or the output of the framework, then we add the components that are necessary to generate the output and avoid overheads for the optimal performance. In the case of IaaS and PaaS cloud computing models, the behaviors of the services provided by these models were studied first to see what are the features that IaaS and PaaS offer and what components are actually needed to classify and discover these services.

In order to develop a framework that works properly with PaaS and IaaS, we have to understand how IaaS and PaaS work and what the features that each computing model provides are. For IaaS the types of services are usually hosting, it is not just limited to storage since the service also hosts user processes on its computing resources. IaaS is fully scalable and it is available to users on demand. PaaS provides tools for software development that are necessary for writing applications, this cloud computing model reduces the complexity of development which can lead to efficiency since it already has the necessary infrastructure built-in. Some services for PaaS model are locked into a certain platform but most of the recent available services are available lock-in free since it gives the developer more control over the development environment for maintenance and enhancement of the application.

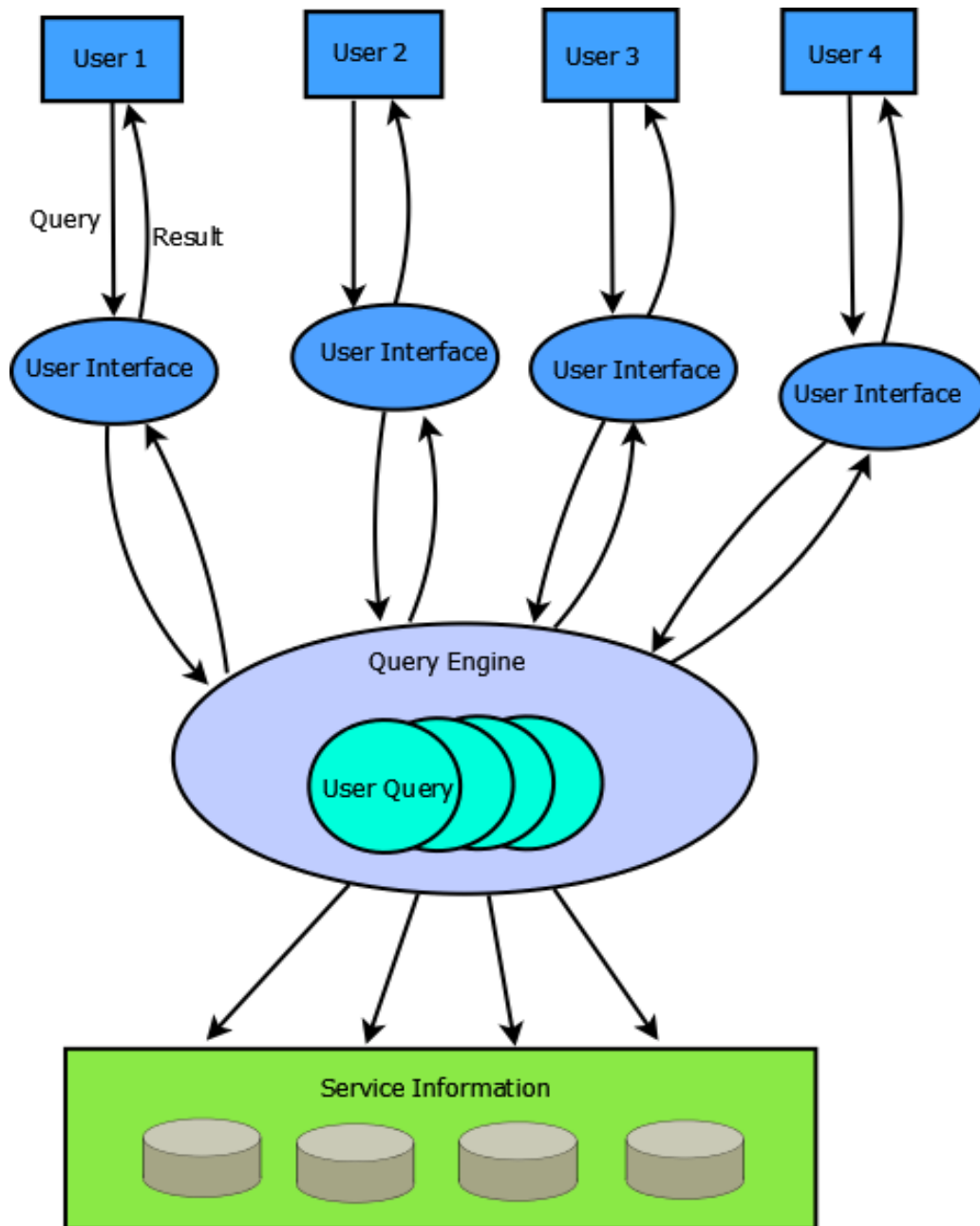
Figure 17 is an abstraction of these core features of IaaS and PaaS cloud computing models.



**Figure 17** IaaS and PaaS core features [41]

### 3.2. Behavioral Design of IaaS and PaaS Cloud Service Discovery Framework

For a discovery framework, the first thing that counts is the user query. This framework is functioning with on a syntax-based query engine, when the query is entered in the textbox the engine immediately starts to process each letter as they are being entered and then gives the user a number of suggestions until it finds the exact match for the user query. The query engine searches from the available service information which is stored in the database. Figure 18 shows the behavioral diagram of our proposed framework for IaaS and PaaS.

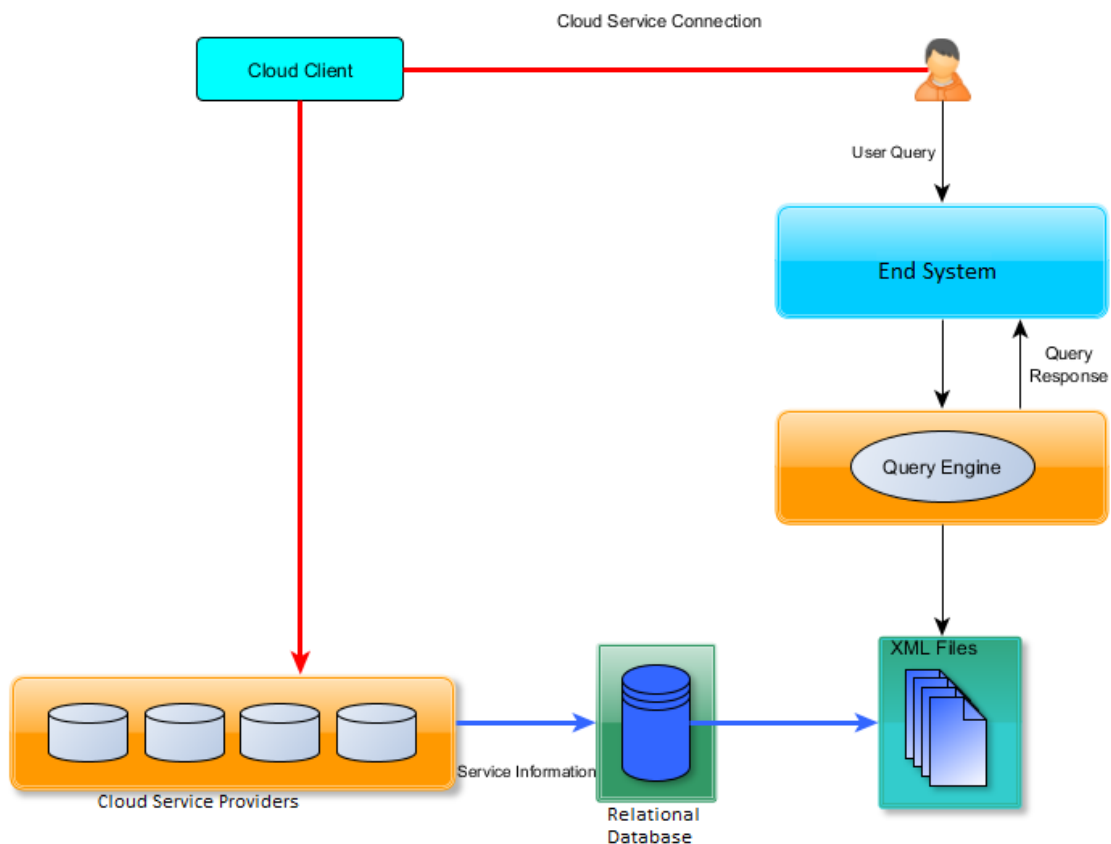


**Figure 18** Behavioral Diagram of IaaS and PaaS Service Discovery Framework

As it is shown in Figure 18 the user enters the query in the interface which is connected to the query engine, then the engine sets the query as the input and searches through the Service Information to find the best match. In our proposed framework the query engine should interact directly with the Service Information in order to give live suggestions as the user enters the query, all the data stored in the Service Information are in XML format for ease of access.

### 3.3. Structural Design of IaaS and PaaS Cloud Service Discovery framework

The structural design of the framework represents all the components and how they interact with each other, the framework was designed with the assumption that the Cloud Client is separate from the end user since not all existing end user devices can take advantage of every IaaS and PaaS cloud services, however all existing devices are capable enough for discovering these services. Figure 19 shows the structural design of our Cloud Service Discovery framework.



**Figure 19** Structural Design of IaaS and PaaS Cloud Service Discovery framework

In this framework, the focus is on discovering IaaS and PaaS cloud services using a syntax-based method. Designing components in a general way might work but it will perform poorly because of the overhead in the system and the implementation, for that rea-



son every component in this framework is designed to take advantage of a syntax-based method in order to get the best performance.

### **3.3.1 Data Flow**

As it is shown in Figure 19 every arrow has a different color which represents the data flow in the system, the black arrow represents the query flow which starts with the user entering the query in the system and finishes with the query engine response. Blue arrows represent the flow of service information from the providers to our framework, the service information first needs to be stored in our relational database and then from there it goes to the XML servers so that our query engine can have access to the information. We select XML for data representation because of its tree structure, by incorporating nested nodes we can easily represent metadata. The red arrow represents the connection between the user and the cloud services which has to be done through a cloud client.

### **3.3.2 End System**

The end system in this framework is actually the device which is used to discover the available IaaS and PaaS cloud services. In some cases, this device could also be used as a platform for the actual cloud service which is offered by the cloud service provider however in this framework we make an assumption based on a general approach. Almost every existing device can be used as a platform for cloud service discovery but not all of them are suitable as a platform for using IaaS and PaaS cloud services.

### **3.3.3 Query Engine**

In this framework, we are using a syntax-based query engine. For IaaS and PaaS cloud computing models a syntax-based search will work perfectly since the services in these models are independent of each other, IaaS provides hardware resources which work separate from each other and PaaS provides software development tools which also work independently from each other.

A semantic-based search will also work but it will not change the search result in the cases of IaaS and PaaS cloud computing models, it only adds more complexity and reduces

the performance of the query engine. A syntax-based query engine is implemented to get the best performance for our IaaS and PaaS cloud service discovery framework.

#### **3.3.4 XML Files (Information Storage)**

XML is a markup language that is used for documentation; it is both machine-readable and human-readable, it is also one of the best ways to store information since it provides syntax to declaring the structure of documents and syntax for document markup.

XML provides a friendly environment for programmers because of its computing standards; the syntax contains a set of rules which makes it easy for any future changes in the stored information. The structure of XML is in layers, it is suitable for storing service information since cloud services have different attributes. In the cases of IaaS and PaaS cloud computing models, XML is the best way to store the information and also the cloud service architecture standard is defined in XML. This helps the query engine to have complete access to the available information in order to search them based on the user query.




#### **3.3.5 Relational Database**

Using a relational database in our framework is the best approach. As it is shown in our framework, we are using a relational database to store the cloud service information that we are getting directly from the providers. Using a relational database helps us to distinguish the services properly; this characteristic of the relational database is extremely useful since for IaaS and PaaS models there are usually some cases that a number of providers are offering same infrastructures like the Processor or the RAM. With the use of a relational database we can have classified information about the services and as a result, the XML Data is more precise.

#### **3.3.6 Cloud Client and Cloud Service Providers**

In order to use any software over the internet, a client application is needed. In the case of regular internet services since these applications are not demanding compared to cloud services, a web browser is good enough for interacting with different internet applications. Services offered by IaaS and PaaS cloud service providers usually require virtual-

ization, for IaaS majority of services are hardware resources like storage or Processors and for PaaS, the resources are software development tools which could be in form of an API or a specific development environment. These services require a cloud client since the type of services that are offered are more demanding than regular web services and a web browser is not suitable. Figure 21 shows the classification of cloud services with respect to their access tools.

Service Class	Main Access & Management Tool	Service content
 SaaS	Web Browser	<b>Cloud Applications</b> Social networks, Office suites, CRM, Video processing
 PaaS	Cloud Development Environment	<b>Cloud Platform</b> Programming languages, Frameworks, Mashups editors, Structured data
 IaaS	Virtual Infrastructure Manager	<b>Cloud Infrastructure</b> Compute Servers, Data Storage, Firewall, Load Balancer

**Figure 20** Cloud service models and their main access tools [42]

As it is shown in figure 21 only SaaS cloud computing model is using web browsers as its client application to access the services, for IaaS and PaaS models a web browser is simply not capable of delivering these service to the user and they need their own client applications for service delivery.

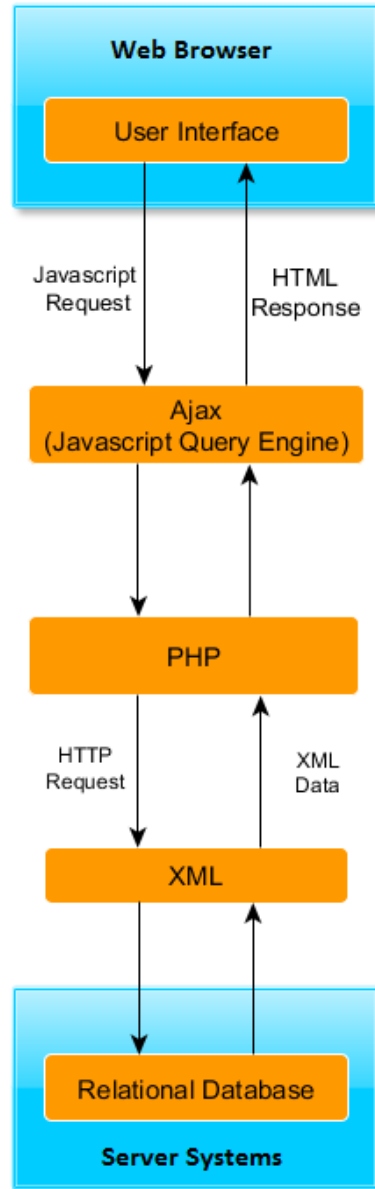
## Chapter 4. Implementation and Experiments

---

In this chapter, we explain how the framework is implemented and what tools were used for the implementation. Our focus here is to go through the method of implementation, the assumptions that were made for this implementation and what scenarios were considered for evaluation. Finally based on our results we point out the advantages of our framework over other methods for service discovery of IaaS and PaaS cloud computing models.

### 4.1. Work Flow Diagram of IaaS and PaaS Cloud Service Discovery Framework

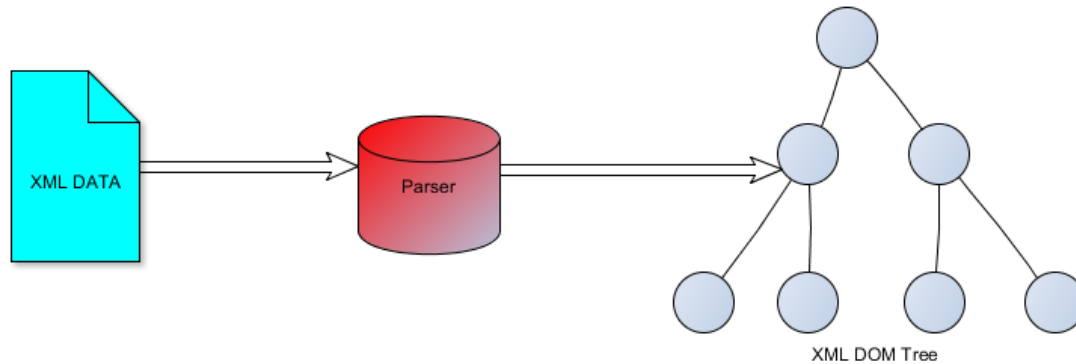
In this section we present the implementation method of the framework, it is important to use every component of the framework properly and avoid any overheads in order to get optimal performance. Figure 22 shows the Workflow of our cloud service discovery framework.



**Figure 21** IaaS and PaaS Cloud Service Discovery Workflow Diagram

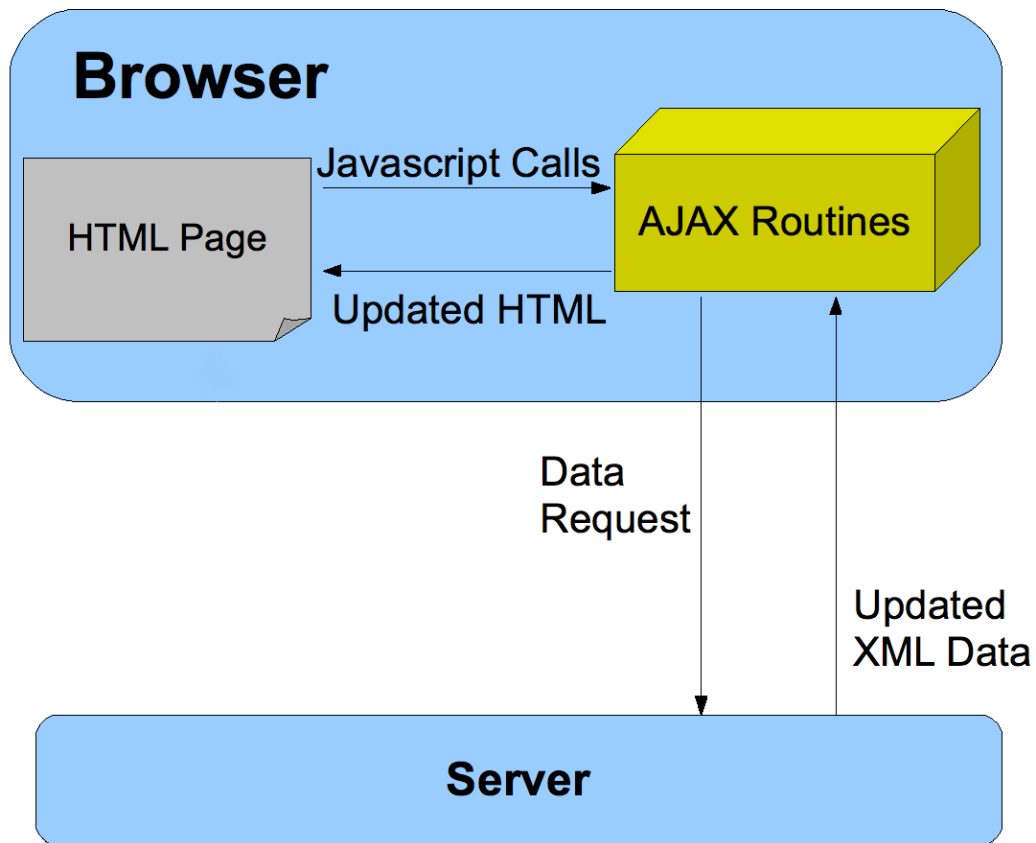
As it is shown in figure 22 since our concern is just with the discovery of the cloud services, the web browser is good enough for any platform. We use AJAX to implement the syntax-based query engine. The advantage of using AJAX is its ability to communicate with a remote server such as sending a request and getting the response without any interference with the current state of the page and using DOM API for dynamic display and interaction with data. DOM is an API which helps to parse an XML document in order to

be treated as a tree structure with multiple nodes where each node is an object representing a data in our XML file. Figure 23 shows the relation between XML data and DOM tree.



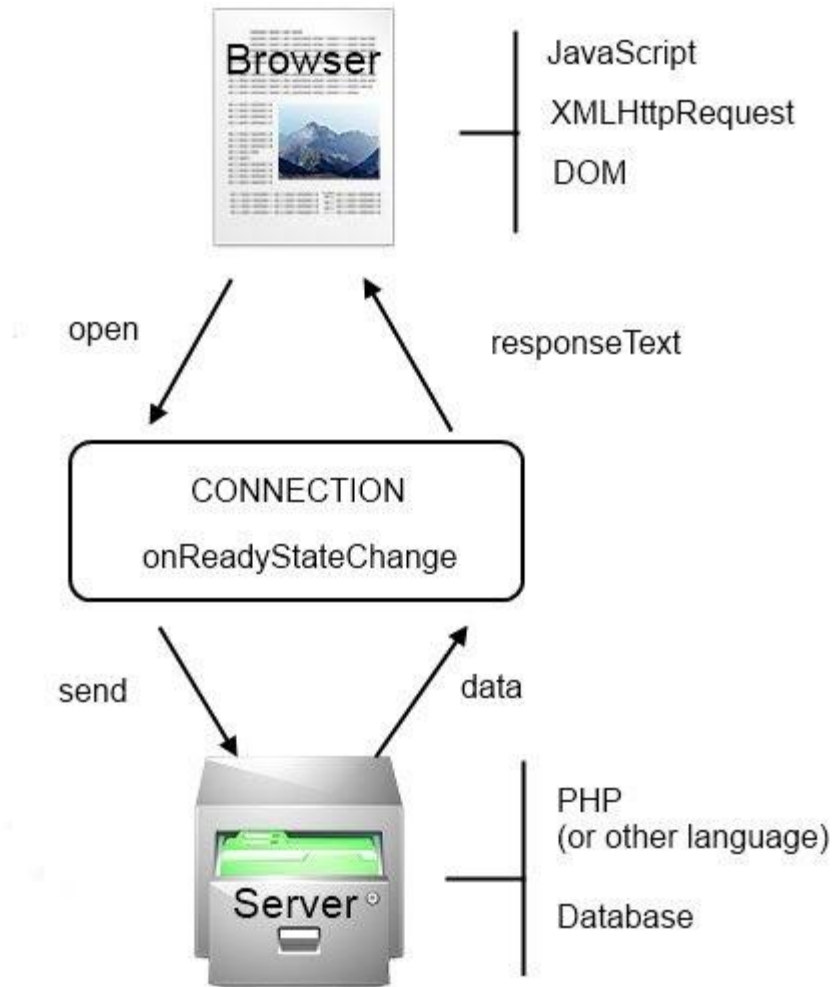
**Figure 22** Transition from XML Data to DOM Tree

AJAX can work with and without incorporating DOM. If we do not incorporate DOM, the user query is treated as a JavaScript call and goes to the server for requesting data. The server searches through the XML document and tries to find the match or something similar to the user query and then after applying AJAX routines it updates the HTML page on the user side. Figure 24 shows how the user request goes from the HTML page on the user side, through AJAX functions and reaches the server without incorporating DOM API.



**Figure 23** Using AJAX without DOM API [43]

In the case of applying DOM API, the type of the request changes and the XML data is being treated as a DOM tree with objects as our XML data. In this scenario, the user query is treated as *XMLHttpRequest* which helps to establish a connection between the server and the user before sending the request. In order to communicate properly with the server, we need to use a server-side scripting language such as PHP. Since all the information is on the server side and we want to access the XML data from the XML server, DOM API needs to be included in the PHP code. Figure 25 shows how AJAX works if we use DOM API and PHP to have access to XML data.



**Figure 24** Using AJAX with DOM API [44]

As it is shown in figure 25 the form of request and response is completely different from the first scenario where DOM API is not applied. For our IaaS and PaaS cloud service discovery framework we are applying DOM API and PHP scripting language which is shown in figure 19.

In our implementation DOM API has to be included since the nature of any discovery framework on the web is to get the user query through the user interface as the input in the framework, sending that request through appropriate routines in order to give it access to the data on the server side and after searching for a match or similarity in the stored data, returning the response from the server back to the user.



The implementation that we are showing in our workflow diagram is suitable for our cloud service discovery framework since it is taking advantage of all the components in our framework. DOM and AJAX in this framework both need HTML 5 capable browsers to function properly, however, this is not a concern since every browser since 2015 runs on HTML 5 and we implemented the framework with this assumption that all users are at least using a one-year-old web browser.

## 4.2. Implementation Method

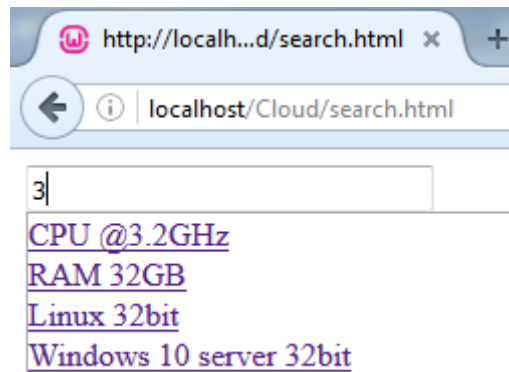
For our implementation, we used WAMP server, a software stack consisting of Apache web server, OpenSSL for SSL support, MySQL database and PHP programming language. As it is shown in figure 19 the relational database is getting the relevant data from cloud service providers and our XML files are generated from the data in our relational database. There are several ways to generate the XML file from the data in the database, however; we are going to explain the process that is the most common. Data is stored in tables in the database; we have to first establish criteria for our table, then database relationships and constraints are converted into W3C XML schema constraints. XML schema uses XML Schema Definition (XSD) to describe the structure of an XML document; since XSD uses XML syntax any XML editor can be used to parse schema files, this helps us to easily generate the XML document from our schema.

After we set up the WAMP server and store our XML files we begin implementing the query engine. The engine uses Ajax functions, we want the script to work with all major browsers so we have to use *if (window.XMLHttpRequest)* statement, this helps us to run the code on Chrome, IE7, IE6, Firefox, Opera and Safari web browsers. We also have to include our PHPDOM code in order to have access to XML files. In the PHPDOM code, we use the DOM function and load the available XML files to scan the XML content and give suggestions to users based on the query or generate a message in case that a match is not found in the XML files.

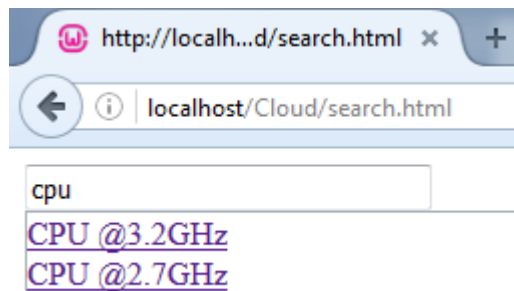
## 4.3. Generating Output

### 4.3.1 Query Engine Testing

For this demonstration we used (<http://azure.microsoft.com>) in our XML file as the IaaS and PaaS cloud service provider which means if users select any of the services, they will be directed to Microsoft Azure webpage. We generated four different outputs that resemble real scenarios, figures 26 and 27 show how the query engine responds when there are similarities between the query and available data in our database.



**Figure 25** Similarity far from an exact match

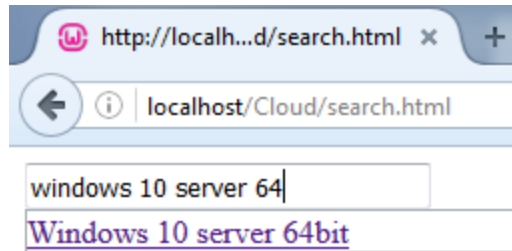


**Figure 26** Similarity close to match the query from the user

In both figures 26 and 27 our syntax-based query engine is giving the user similar results based on the input, the search is syntax-based which means every time the user enters a letter the engine searches through the entire XML data based on the input string and tries to find the closest match based on that string. As it is shown in figure 23 our suggestions are very general because we have only entered one character, in these scenarios the sug-

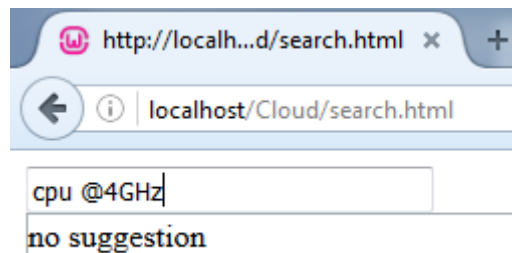
gestions are almost any available string with that character in it which is not accurate. The scenario in figure 26 is just to test if the engine works in the worst case, a real user never enters a query with just one character.

Figure 27 uses the same concept as figure 26 however since the input is more accurate, the suggestion that we are getting is also more relative and shows the user more accurate results than figure 26. As our input string gets more accurate, the suggestions get closer to the input string until they get to the exact match otherwise suggestions stay with the closest result. Figure 28 shows the scenario where the input has an exact match in the query results.



**Figure 27** Input getting an exact match from the query engine

As it is shown in figure 28 there is only one suggestion, this output is generated because the input string is precise enough for the query engine to find the exact match in our XML files. The figure below shows another scenario where the query engine cannot give any suggestions to the user.



**Figure 28** Input with no suggestions from the query engine

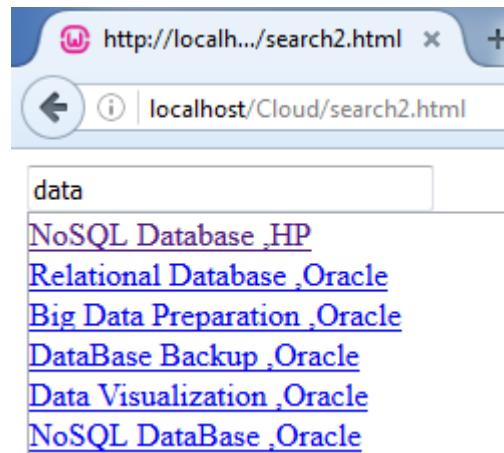
As it is shown in figure 29 there are no suggestions for our input, this is because the query engine could not find any similarity between the input string and the available data in our XML files.

### 4.3.2 Experimenting from the user point of view

For this experimentation, we used a different XML file which has IaaS and PaaS data from real providers. Figures below represent how the query engine responds if we use real cloud service information in our XML file.

#### Experiment-1: Generic Query

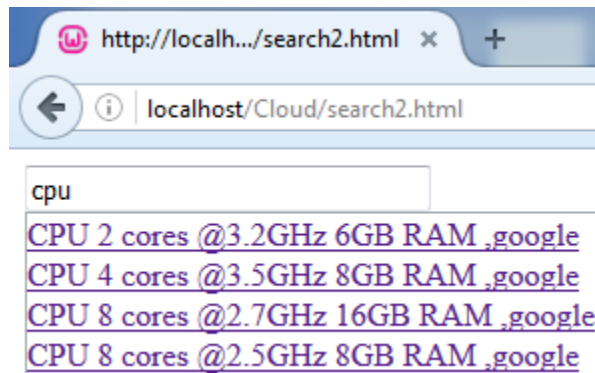
When the user enters a specific keyword the query engine searches through the XML file. The DOM API assists the search to perform better by transforming the XML into a tree with objects as nodes. Since the matching function searches through the files letter by letter, it is not important if the keyword is in the middle or the beginning of the phrase. As it is shown in figure 30 the keyword “Data” is in the middle and we can get all the services with that keyword regardless of its position. Figure 31 shows the same example with a different keyword just to point out the accuracy of the matching function.



**Figure 29** Using keyword “Data” as our generic query

#### Experiment-2: Searching for Available CPU’s From All Providers

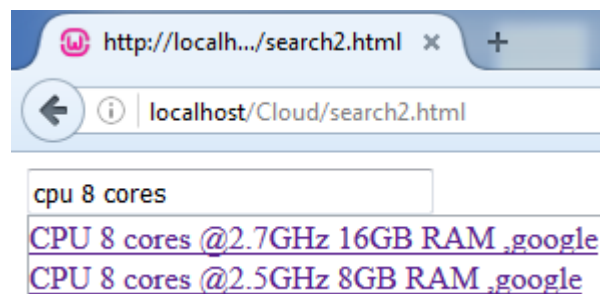
If the user is looking for all CPU’s offered by all service providers available in the database, they can submit a query as shown in Figure-31.



**Figure 30** Using keyword “CPU” as our generic query

### Experiment-3: Searching for a Specific Resource

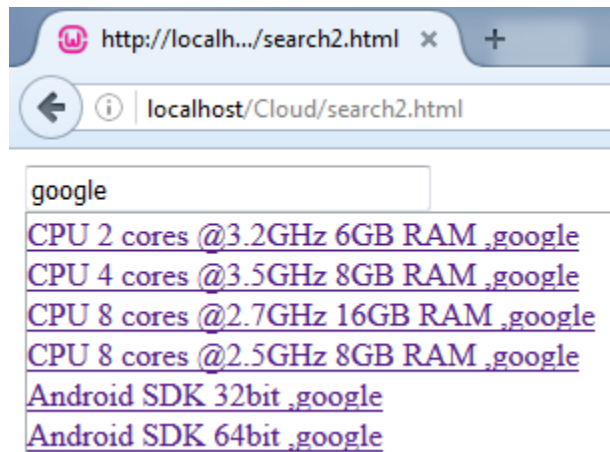
Figure 32 is an example of searching for a specific service, accurate query results for accurate suggestions.



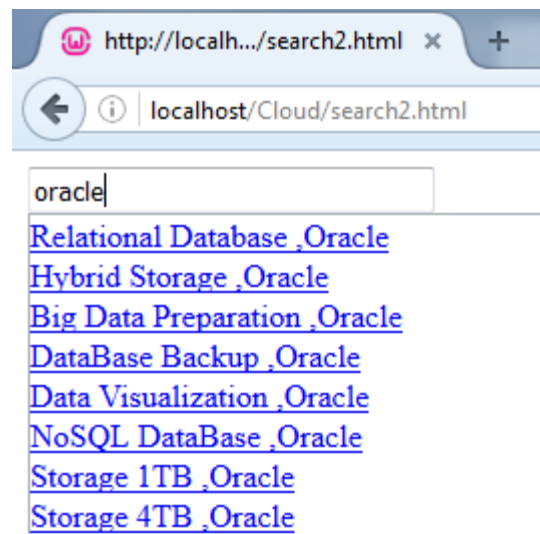
**Figure 31** Searching for an actual IaaS cloud resource

### Experiment-4: Searching for a Specific Service Provider

Figures 33 and 34 are examples of searching for a specific cloud service provider. As it is shown when the user enters the name of the provider, every available cloud service from that provider shows up in the suggestions.



**Figure 32** Searching for a specific service Provider, Example 1



**Figure 33** Searching for a specific provider, Example 2

## 4.4. Comparison

We comparing our framework with Semantic-based and Filter by Attribute method since these two methods are currently being used as cloud service discovery solutions by several companies.

### 4.4.1 Semantic-based method

In this section we compare our framework based on the implementation of a semantic based discovery framework in general, we are not going to select a specific semantic-based framework since all semantic based discovery frameworks have a number of core characteristics that does not change with different implementations. Both a syntax-based framework and a semantic-based framework have advantages and disadvantages in different situations, the comparison between these two approaches towards cloud service discovery are as follows:

- Syntax-based frameworks have better runtime than semantic-based frameworks, however; semantic based frameworks are more precise in the case of resource-intensive services.
- In the case of IaaS and PaaS cloud computing models regardless of a number of resources, a syntax-based framework is better since there are no relations between the services and a semantic-based framework does not help much other than making the search more complex.
- Syntax-based frameworks are not very efficient when it comes to SaaS cloud computing model, software services have a lot of dependencies and the way to model them is to have ontology tree. Semantic-based frameworks work best with SaaS models and are much more accurate than syntax-based frameworks in this category.
- Both discovery methods have their own benefit but for our proposed framework which focuses on IaaS and PaaS cloud computing models, a semantic-based method would just add overhead and makes the search slower due to its complex algorithms, syntax-based method is the optimal solution for IaaS and

PaaS models based on the current technology and available services in the industry.

- There are semantic-based implementations in references [1] and [45] that are applied for IaaS and PaaS. These implementations use an ontology to model the available services and they clearly point out the complexity since there are a limited number of IaaS and PaaS resources. Using ontology for a limited number of resources does not enhance the accuracy of the search.

#### **4.4.2 Filter by Attribute method**

This method is syntax-based however it does not take user query; it only allows users to eliminate unrelated services by selecting the attributes that are available from the system. The specifications of this method are as follows:

- It is the simplest and the most inaccurate discovery method available.
- This method is a widely used syntax-based service discovery solution, however; it is just a filter, not a query engine.
- Compared to a syntax-based query engine this method is not effective for any cloud computing model.



## 4.5. Summary of the Comparison

This section summarizes the comparison in section 4.3. The table below shows this comparison.

Discovery Method	Accuracy (IaaS, PaaS)	Complexity	Discovery Method
Filter by Attribute	There is no query processing in this method, the user is provided with a number of options to choose from based on the filters that are available	This method has no complexity since there is no processing in the background.	Attribute-based XML and JavaScript
Semantic-based	It has high accuracy but requires lots of processing and it is best suited for resource intensive models.	This method uses reasoning algorithms and ontology, as a result it has high complexity	Semantic Algorithm and Ontology[26, 1]
Syntax-based (Proposed)	The accuracy is good enough for IaaS and PaaS compared to other methods [1].	It is using a syntax search with the help of DOM API. The complexity of the search depends on the structure of the DOM tree.	XML and using Ajax with PHP

Table 1 Comparison Summary

## Chapter 5. Conclusions

---

### 5.1. A Summary of the Research

Cloud computing is one of the fastest growing computing platforms available recently, these platforms are available as services and they use the internet as a medium to provide these services. Currently, there are no major frameworks that can be used for cloud IaaS and PaaS service discovery, the ones that are available only show the services for a specific provider and those frameworks do not operate across the internet identifying other available IaaS and PaaS cloud services.

The current literature on cloud service discovery frameworks only focuses on cloud services in general, not a particular computing model. As a result, the majority of the solutions were semantic-based and did not care about the performance of the discovery framework. In this research, we develop a cloud service discovery framework that focuses on IaaS and PaaS cloud computing models which can perform as good as the previously introduced cloud service discovery frameworks with lower overhead. We used a syntax-based query engine for query processing and XML to store the data.

After the implementation, we tested the query engine to make sure it functions properly with the user input and then we tested the framework as a user, for the second test we collected real cloud service information from real cloud service providers to mimic a real world scenario. At the end, we compared our framework with the existing ones and pointed out the advantages and disadvantages in different scenarios.

### 5.2. Contributions of the thesis

In this thesis, we propose a cloud service discovery framework for IaaS and PaaS cloud computing models. Using this syntax-based cloud service discovery framework, the user enters the query, the query engine takes that as an input, goes through the XML files that

contain the cloud service information, performs a syntax-based search and shows either the exact match or the available services that are close to what the user requested.

It is much more efficient to use this solution over a semantic-based discovery framework for IaaS and PaaS cloud computing models. Services that are available for IaaS and PaaS do not require ontology and for that reason using a semantic-based discovery framework adds complexity to the framework and reduces system performance. Our framework also is much more efficient over the Filter by Attribute method since it is able to show similarities in case if an exact match could not be found.

### **5.3. Publication**

F. Firozbakht, W. Obidallah, B. Raahemi, “Cloud Computing Service Discovery Framework for IaaS and PaaS Models “, accepted at The Second International Conference on Internet of Things, Data and Cloud Computing, University of Cambridge, United Kingdom, August 2016, will be published in ACM library.

### **5.4. Limitations and Future Works**

One of the limitations of this research was during the implementation phase; it was simply not possible to test the framework with the cloud services that are being provided by real providers, we could not configure our own database to have access to what cloud service providers are offering since we did not have their permission to do such a thing and instead we came up with our own assumptions which mimic a real-world scenario and proves that the framework works properly. We also could not calculate computational time since we did not have access to other framework in order to compare and determine which one is faster.

As future work some new components could be used for the payment system, they need their own dedicated server and security protocols. Another upgrade to the framework could be adding a web crawler that can automatically get the cloud service information from the providers and store them in the database, however, for this upgrade permission from the cloud service providers that are giving us their service information is necessary.

## Appendix A: Source Code

---

### JavaScript Code

```
<html>
<head>
<script>
function showResult(str) {
    if (str.length==0) {
        document.getElementById("PHPDOM").innerHTML="";
        document.getElementById("PHPDOM").style.border="0px";
        return;
    }
    if (window.XMLHttpRequest) {

        xmlhttp=new XMLHttpRequest();
    } else {
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
            document.getElementById("PHPDOM").innerHTML=xmlhttp.responseText;
            document.getElementById("PHPDOM").style.border="1px solid #A5ACB2";
        }
    }
    xmlhttp.open("GET","PHPDOM.php?q="+str,true);
    xmlhttp.send();
}
</script>
</head>
<body>

<form>
<input type="text" size="30" onkeyup="showResult(this.value)">
<div id="PHPDOM"></div>
</form>

</body>
</html>
```

## PHP Code

```
<?php
$xmlDoc=new DOMDocument();
$xmlDoc->load("services.xml");

$x=$xmlDoc->getElementsByTagName('service');

$q=$_GET["q"];

if (strlen($q)>0) {
    $hint="";
    for($i=0; $i<($x->length); $i++) {
        $y=$x->item($i)->getElementsByTagName('title');
        $z=$x->item($i)->getElementsByTagName('url');
        if ($y->item(0)->nodeType==1) {

            if (strstr($y->item(0)->childNodes->item(0)-
>nodeValue,$q)) {
                if ($hint=="") {
                    $hint="<a href='" .
                    $z->item(0)->childNodes->item(0)->nodeValue .
                    "' target='_blank'>" .
                    $y->item(0)->childNodes->item(0)->nodeValue .
"</a>";
                } else {
                    $hint=$hint . "<br /><a href='" .
                    $z->item(0)->childNodes->item(0)->nodeValue .
                    "' target='_blank'>" .
                    $y->item(0)->childNodes->item(0)->nodeValue .
"</a>";
                }
            }
        }
    }
}

if ($hint=="") {
    $response="no suggestion";
} else {
    $response=$hint;
}

echo $response;
?>
```

## XML Document for query engine demonstration

```
<cloudservices>

<service>
<title>CPU @3.2GHz </title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>CPU @2.7GHz </title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Storage 500GB</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Storage 1TB</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>RAM 16GB</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>RAM 32GB</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Linux 64bit</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Linux 32bit</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Windows 10 server 64bit</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Windows 10 server 32bit</title>
```

```

<url>http://azure.microsoft.com/en-us/</url>
</service>

</cloudservices>

```

## XML for user testing

```

<cloudservice>

  <service>
    <title>CPU 2 cores @3.2GHz 6GB RAM ,google</title>
    <url>https://cloud.google.com/products/</url>
  </service>

  <service>
    <title>CPU 4 cores @3.5GHz 8GB RAM ,google</title>
    <url>https://cloud.google.com/products/</url>
  </service>

  <service>
    <title>CPU 8 cores @2.7GHz 16GB RAM ,google</title>
    <url>https://cloud.google.com/products/</url>
  </service>

  <service>
    <title>CPU 8 cores @2.5GHz 8GB RAM ,google</title>
    <url>https://cloud.google.com/products/</url>
  </service>

  <service>
    <title>Storage 500GB ,Rackspace</title>
    <url>https://www.rackspace.com/</url>
  </service>

  <service>
    <title>Storage 1TB ,Rackspace</title>
    <url>https://www.rackspace.com/</url>
  </service>

  <service>
    <title>Linux 64bit VM ,Amazon</title>
    <url>https://aws.amazon.com/ec2/</url>
  </service>

  <service>
    <title>Linux 32bit VM ,Amazon</title>
    <url>https://aws.amazon.com/ec2/</url>
  </service>

```

```

<service>
<title>Windows 10 server 64bit VM ,Microsoft</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Windows 10 server 32bit VM ,Microsoft</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Windows Azure SDK ,Microsoft</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>NoSQL Database ,HP</title>
<url>http://www8.hp.com/us/en/cloud/helion-overview.html</url>
</service>

<service>
<title>Windows 10 server 64bit VM ,Microsoft</title>
<url>http://azure.microsoft.com/en-us/</url>
</service>

<service>
<title>Amazon Lumberyard Engine ,Amazon</title>
<url>https://aws.amazon.com/lumberyard/</url>
</service>

<service>
<title>virtual machine Import/Export ,Amazon</title>
<url>https://aws.amazon.com/lumberyard/</url>
</service>

<service>
<title>Relational Database ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>Hybrid Storage ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>Big Data Preparation ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>DataBase Backup ,Oracle</title>

```



```
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>Data Visualization ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>NoSQL DataBase ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>Android SDK 32bit ,google</title>
<url>https://cloud.google.com/products/</url>
</service>

<service>
<title>Android SDK 64bit ,google</title>
<url>https://cloud.google.com/products/</url>
</service>

<service>
<title>Storage 1TB ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

<service>
<title>Storage 4TB ,Oracle</title>
<url>https://cloud.oracle.com/en_US</url>
</service>

</cloudservice>
```

## References

---

- [1] Kang, J., & Sim, K. M. (2011, October). Towards agents and ontology for cloud service discovery. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) IEEE, 2011 International Conference on* (pp. 483-490).
- [2] Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3), 45-77.
- [3] Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *MIS quarterly*, 37(2), 337-355.
- [4] Vaishnavi, V., & Kuechler, W. (2004). Design research in information systems.
- [5] <http://www.anlaufmanagement.rwth-aachen.de>, July 2016
- [6] <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211>, July 2016
- [7] <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211>, July 2016
- [8] Paliwal, A. V., Adam, N. R., Xiong, H., & Bornhovd, C. (2006, December). Web service discovery via semantic association ranking and hyperclique pattern discovery. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, IEEE Computer Society, (pp. 649-652).
- [9] <http://www.eio.upc.edu/lceio/manuals/cplex-11/html/globalfiles>, July 2016
- [10] Bhagdev, R., Chapman, S., Ciravegna, F., Lanfranchi, V., & Petrelli, D. (2008, June). Hybrid search: Effectively combining keywords and semantic searches. In *European Semantic Web Conference*, Springer Berlin Heidelberg, (pp. 554-568).
- [11] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M., 2010. A view of cloud computing. *Communications of the ACM*, 53(4), pp.50-58.
- [12] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I., 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6), pp.599-616.
- [13] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing—The business perspective. *Decision support systems*, 51(1), 176-189.
- [14] Yu, R., Yang, X., Huang, J., Duan, Q., Ma, Y., & Tanaka, Y. (2012, September). QoS-aware service selection in virtualization-based Cloud computing. In *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific IEEE*, (pp. 1-8).

- [15] Gorelik, E. (2013). Cloud computing models (Doctoral dissertation, Massachusetts Institute of Technology).
- [16] [http://en.wikipedia.org/wiki/cloud\\_computing](http://en.wikipedia.org/wiki/cloud_computing), July 2016
- [17] Jinesh Varia, Architecting for the Cloud: Best Practices, Amazon Web Services, January 2010
- [18] Tsai, W. T., Sun, X., & Balasooriya, J. (2010, April). Service-oriented cloud computing architecture. In Information Technology: New Generations (ITNG), 2010 Seventh International Conference on (pp. 684-689). IEEE.
- [19] Hoefer, C. N., & Karagiannis, G. (2010, December). Taxonomy of cloud computing services. In 2010 IEEE Globecom Workshops (pp. 1345-1350). IEEE.
- [20] Sheu, P. C. Y., Wang, S., Wang, Q., Hao, K., & Paul, R. (2009, September). Semantic computing, cloud computing, and semantic search engine. In Semantic Computing, 2009. ICSC'09. IEEE International Conference on (pp. 654-657).
- [21] Elgazzar, K., Hassanein, H. S., & Martin, P. (2014). Daas: Cloud-based mobile web service discovery. *Pervasive and Mobile Computing*, (Vol. 13, pp. 67-84)
- [22] Sheu, Yu, Ramamoorthy, Joshi, and Zadeh, (2007), Editorial Preface, *International Journal of Semantic Computing*, (Vol. 1.1, pp. 1-9)
- [23] Ross, J. W., & Westerman, G. (2004). Preparing for utility computing: The role of IT architecture and relationship management. *IBM systems journal*, (Vol. 43.1, pp. 5-19)
- [24] Kang, J., & Sim, K. M. (2011, October). Towards agents and ontology for cloud service discovery. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)* IEEE, 2011 International Conference on (pp. 483-490).
- [25] Han, T., & Sim, K. M. (2010, March). An ontology-enhanced cloud service discovery system. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (Vol. 1, pp. 17-19)
- [26] Andreasen, T., Bulskov, H., & Knappe, R. (2003, November). From ontology over similarity to query evaluation. In *2nd International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems (ODBASE)*, Catania, Sicily, Italy
- [27] Aleman-Meza, B., Halaschek-Weiner, C., Arpinar, I. B., Ramakrishnan, C., & Sheth, A. P. (2005). Ranking complex relationships on the semantic web. *IEEE Internet computing*, (Vol. 9.3, pp. 37-44)
- [28] Noor, T. H., Sheng, Q. Z., Alfazi, A., Ngu, A. H., & Law, J. (2013, June). CSCE: a crawler engine for cloud services discovery on the world wide web. In *Web Services (ICWS)*, 2013 IEEE 20th International Conference on (pp. 443-450).
- [29] Joshi, K. P., Yesha, Y., Ozok, A. A., Yesha, Y., Lahane, A., Kalva, H., ... & Furht, B. (2010). User-centric smart services in the cloud. In *The smart internet*, Springer Berlin Heidelberg, (pp. 234-249).

- [30] Han, L., & Berry, D. (2008). Semantic-supported and agent-based decentralized grid resource discovery. *Future Generation Computer Systems*, (Vol. 24.8, pp. 806-812)
- [31] Rasmus Knappe and Henrik Bulskov and Troels Andreasen,(2008), Department of Computer Science, Roskilde University, “ On Similarity Measures for Concept-based Querying ”, Roskilde, Denmark
- [32] Giunchiglia, F., Gomez-Perez, A., Pease, A., Stuckenschmidt, H., Sure, Y., & Willmott, S. (2003, May). Ontologies and Distributed Systems. In *Proceedings of the IJCAI-03 Workshop, CEUR Workshop Proceedings* (Vol. 71)
- [33] Troels Andreasen and Henrik Bulskov and Rasmus Knappe,(2007), Department of Computer Science, Roskilde University, Roskilde, Denmark
- [34] Sim, K. M., & Wong, P. T. (2004). Toward agency and ontology for web-based information retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, (Vol. 34.3, pp. 257-269)
- [35] Li, S., & Chen, H. P. (2014, August). A Context-Aware Framework for SaaS Service Dynamic Discovery in Clouds. In *International Conference on Algorithms and Architectures for Parallel Processing*, Springer International Publishing, (pp. 671-684).
- [36] Makhzan, M. A., & Lin, K. J. (2006, June). Solutions to a complete web service discovery and composition. In *The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06)* (pp. 73-73).
- [37] Oh, S. C., Kil, H., Lee, D., & Kumara, S. R. (2006, June). Algorithms for Web Services Discovery and Composition Based on Syntactic and Semantic Service Descriptions. In *CEC/EEE* (p. 66)
- [38] McHugh, J., & Widom, J. (1999). Query optimization for XML.
- [39] Duda, C., Frey, G., Kossmann, D., Matter, R., & Zhou, C. (2009, March). Ajax crawl: Making ajax applications searchable. In *2009 IEEE 25th International Conference on Data Engineering, IEEE*, (pp. 78-89).
- [40] <http://www.intelcloudfinder.com/cloud-provider-search>, July 2016
- [41] <http://www.pbxl.co.jp/en/saas-paas-iaas/>, July 2016
- [42] <http://1.bp.blogspot.com>, July 2016
- [43] <http://loadstorm.com>, July 2016
- [44] <http://www.xul.fr>, July 2016
- [45] Han, T., & Sim, K. M. (2010, March). An ontology-enhanced cloud service discovery system. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (Vol. 1, pp. 17-19)