UNIVERSITÀ DEGLI STUDI DI MILANO

PHD IN COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

DOTTORATO DI RICERCA IN INFORMATICA
XXIX Ciclo

ADVANCED SLA MANAGEMENT IN CLOUD COMPUTING

INF/01 INFORMATICA

DOCTORAL DISSERTATION OF:
Gerson Antunes Soares

SUPERVISOR:
Prof. Vincenzo Piuri

CO-SUPERVISOR:
Prof. Sara Foresti

DIRECTOR OF PHD PROGRAM:
Prof. Paolo Boldi

Academic Year 2015/16

"There is no such thing as a 'self-made' man.
We are made up of thousands of others.
Everyone who has ever done a kind deed for us, or spoken one word of
encouragement to us, has entered into the make-up of our character
and of our thoughts, as well as our success."

— George Matthew Adams

ACKNOWLEDGMENTS

First of all, I want to thank God that blessed every day of my life, to enlighten my path and gave me the strength to continue on.

I thank Prof. Vincenzo Piuri for the opportunity to have him as my supervisor. I am very proud to quote him as one of the people responsible for my formation. I thank him for his trust, for his friendship, for his advice and for his patience. It is an example of understanding and competence that goes far beyond what is required. Concerned not only with the work, but mainly with the human being. I want to thank also Prof. Sara Foresti for her help and support as co-supervisor during the research work.

My wife, Daiana da Silva Dors Soares, and our children (Maria Clara, Ana Carolina and João Miguel) who often donated and renounced their dreams, so that I could accomplish mine. I want to say that this achievement is not only mine, but ours. All I got was only possible thanks to the love, support and dedication they always had for me.

To my parents, Jurandi and Maria, who have always taught me to act with respect, simplicity, dignity, honesty and love for others. And thanks to the union of all, the obstacles were overcome, victories were won and divided joys. Thanks a lot for their patience and understanding with my absence during this long journey. To my father and mother in-law, Marcial and Clara, and all the family, who were distant in location but always near to me, thanks so much.

I would like to thank my friends, with no particular order: Ravi Jhawar, Aleksandar Rikalovic, Giovanni Livraga, Abhinav Anand, Ala Arman, Ruggero Donida Labati, Angelo Genovese, Gianluca Sforza, Enrique Muñoz Ballester, Ebadollah Kheirati Roonizi, Md. Aktaruzzaman, Massimo Walter Rivolta, Ruby Karmacharya and Tewodros Mulugeta Dagnew, for having made the day-to-day life at the university so enjoyable! It was extremely enriching to meet and to live with each of them. Thanks for the daily company, for the friend's shoulder, for the trust, the trips and the moments that we spent together.

To all the professors who received me so well. Thanks for the support, encouragement and, above all, the opportunity to learn from people I admire so much. To the employees of the Department of Computer Science: Claudia, Daniela, Mirko, Mario, Danio and Davide for constant collaboration and availability. And, in a special way, to Lorena Sala, for the gratifying coexistence throughout this journey.

I would like to thank the referees Vijay Atluri, Sushil Jajodia, and Laurence T. Yang, for their time spent in reading my thesis and for giving me valuable suggestions for improving my work.

Lastly, I want to thank the other people who contributed directly or indirectly in the elaboration of this work or participated in my life, and that, by chance, I have forgotten to thank.

ABSTRACT

The advent of high-performance technologies and the increase in volume of data used by organizations led to the need for migration from an internal structure to Cloud environment. The continuous development of tools, methods and techniques have expanded the understanding of the various functions, structures and processes related to Cloud Computing. However, the increase in computing power led to the development and use of more complex models, including this scope the complexity of Service Level Agreements (SLA). The need for understanding at a high level of SLAs established between customers and service providers in Cloud led to different studies on the definition and standardization of these agreements.

Nowadays, cloud computing technologies are becoming more and more popular, especially with respect to data storage. However, the processes used to determine the Cloud Service Agreements do not consider the final customer's needs, considering only the supply capacity of the service provider. For these reasons, the development of service agreements that meets the needs of customers should be designed in order to increase the usability of Cloud environments, and enabling the discovery of new areas of application in accordance with market demand.

In this context, the use of ontologies that describes the information that composes each type of service, and thus enable an understanding of the agreements reached, is configured as an approach to be considered. Moreover, the generalization and abstraction of information that can be observed in different services allows a broader vision for managing SLAs.

For these reasons, this thesis aims to find innovative methods for the composition of Service Level Agreements in Cloud Computing. In particular, the methods presented allow demonstrate the convergence of several consolidated techniques in research on Cloud SLA using a new approach that considers new demands on Cloud and allows control of the established agreements, in addition to effectively ensure the application of the concept of XaaS (everything as a service). The originality of the approach allows the registration, search, composition and control of services in Cloud using the same structure.

The new approach presented in this thesis allows the understanding of the impact of the new services requested by customers, giving the provider the possibility of simulating the use of the necessary resources to meet the new services' requests. From the presentation of a conceptual framework we can demonstrate the use of our approach through the examples of different situations presented in the real world and considering the new market possibilities.

CONTENTS

LIST OF FIGURES

1

INTRODUCTION

The evolution of communication and computing technologies has been increasing the use of cloud computing in various fields such as industry, health and education. Especially in the industrial area, the provision of information in the cloud, such as management indicators, needs to maintain the privacy of the results, the confidentiality of information and data integrity. In practice, these requirements can be met through the use of vulnerabilities analysis, strong authentication methods, access control restrictions, etc [1].

Taking as example the management of industrial information, which transforms the data available in high-value information, the availability of this information in a cloud environment needs to consider a few key points to keep information security (privacy, confidentiality and integrity). In this sense the Service Level Agreement (SLA) must also reflect the needs of control of customer.

With Cloud Computing, many user applications as well as their files and data no longer need to be installed or stored on the computer, getting available in the "cloud". The application vendor is responsible for all development tasks, storage, maintenance, upgrade, backup, scheduling, etc. Thus, all materials and documents are available in any environment regardless of where the application is running. [2] supplemented with the following advantages: portability of documents, increase the power of applications, platform independence and ease of abstraction. Besides the advantages mentioned, there is the fact of ease of adaptation to different devices that are accessing the Cloud application.

Since failures in data centers usually occur outside the scope of the client organization, the perception of the degree of the risks in customer orders also changed [3]. In addition, the traditional ways to achieve fault tolerance require customers to have a deeper knowledge of the mechanisms used, however the abstraction model and the cloud computing business do not allow the architectural details of the environment in Cloud are widely available to customers [3]. This implies that the traditional ser-

vice agreements often do not report the cloud computing environment and there is a growing need to solve the reliability concerns.

When establishing a SLA between a customer and a provider, the process should culminate in a contract that spells out the obligations and requirements of all players involved. In Cloud Computing normally this agreement only shows the service provider's obligations, while it does not consider all customer requirements. Therefore is necessary to establish an agreement that allows the customer to retain control (at least with contractual guarantees) of data and information available in Cloud.

In [4] the authors describe that one of the principles for the development of the Service Level Agreement in Cloud Computing should be based on neutral business model and should not require a specific approach for each concept. Another important aspect is the continued evolution of Cloud Computing that features a dynamic behavior when we note the possibility of new markets migrate to this environment.

Also according to [4] Standards and guidelines for cloud SLAs should be able to meet the needs of both smaller customers and corporate customers. In addition, each provider uses a different SLA specification language to meet their own Service Level Objective (SLO) and document their own methods to achieve the SLOs although based on standard concepts and vocabulary.

In this thesis, we propose a model of advanced SLA management based on customer requirements, which considers information about their needs and takes a number of situations, events and information necessary for the correct understanding of the service agreement. As a result, we created a management model that allows easy implementation of dynamic control of resources and the determination of new services based on market demand. In addition to supporting an effective way to establish the construction of Service Level Agreements through a simplified generic ontology.

1.1 MOTIVATION

The popularity of cloud computing has increased considerably compared to traditional information processing systems [3]. To meet effectively the demand for cloud-based services, the service providers to increase their capacity building huge data centers that are spread over several geographical regions (e.g., [5], [6], [7]). As a result of this growth and the availability of resources, many customers are migrating to cloud-based services to make their applications and business processes.

In general, the data centers maintained by providers are constructed with hundreds of thousands of commodity servers. Moreover, virtualization technology is used to maintain the provision computing resources over the Internet and, often, the delivery of services follows the pay-per-use business model [5]. According to [3] a single physical host can be used as a set of multiple virtual hosts by the provider, this benefit "masks" the perception of customers with relation to the available resources making it look like an inexhaustible source of computing resources.

Thanks to its elasticity, convenience, and economic advantage, cloud computing is today one of the reference paradigms for data storage and management and for running (heavy computational) applications. To fully benefit from the advantages of cloud

systems, customers need to have sufficient warranties on the overall dependability of these systems, including the reliability and resilience of cloud architectures, the continuous availability of the services they pay for, the security of the operating environment and infrastructure, the protection guarantees on data and applications in the whole operating flow [8].

To ensure full user satisfaction, the customer and the cloud provider need to carefully specify and agree upon a comprehensive Service Level Agreement. Such SLA should define the infrastructure configuration expected by the customer to support its applications. Conventional approaches to SLA specification (e.g., [9]) allow customers to define her requirements by choosing the most appropriate values for the parameters made available by the cloud provider (e.g., the amount of data storage and speed in data transfer) but it does not guarantee that all customer needs are met. On the basis of the SLA agreed with the customer, the cloud provider identifies a mapping of the customers needs (in terms of the minimum resources required by the customers applications) onto the cloud architecture and services but it does not have the ability to identify new demands. During operation, possible changes in the cloud infrastructure (e.g., faults of connections or processing nodes, and security vulnerabilities) may impair the ability of satisfying customers requirements in the due time or even at all. Monitoring and possibly dynamically adjusting the resource mapping is therefore desirable to ensure complete dependability of the cloud architecture.

It is observed the modern information society uses a very broad set of information to allow businesses to remain competitive and improve their production processes. However we see that process automation is reaching its limits. New quality and management practices require a broad view of processes and their interrelationships, demanding of software tools and traditional hardware more than they are able to offer. The effective and efficient use of dependable cloud infrastructures requires a more transparent agreement between customers and cloud providers on resources, services, operating conditions, and features as well as the mapping of customers requirements onto the cloud architecture.

While generally SLA specifications require the definition of values for configuration parameters, customers would appreciate the availability of a more flexible way for expressing their needs, since customers often do not have an exact understanding of the real requirements of their applications. Also, the management of complex and dynamic environments would benefit from flexibility in reasoning on the mappings of customers requirements onto cloud resources. Therefore, the motivation appears to approve and validate an advanced SLA management model in cloud computing enabling a more realistic approach to the characteristics and needs of different customers and situations including the provision of "everything as a service" (XaaS) by providers.

1.2 OBJECTIVES

Basically we consider three key aspects when designing a solution for the Cloud SLA Management, these aspects are described briefly below.

- *Greater flexibility in SLA trading.* The need of creation of a set of features to include flexibility (adaptation and extension) in Service Level Agreements for Cloud Computing. The advent of high-performance technologies in Cloud Computing will result in a growth in the volume of data and information that can migrate to this environment. The continuous development of tools, methods and techniques have expanded the understanding of the various functions, structures and processes related to services in Cloud Computing. This increased computational power leads the development and use of more complex arrangements. The agreements for this new environment should allow various features presented at different scales, can be combined, complementing each other and providing a more accurate view of customer needs. Thus, the use of a conceptual modeling language which can describe declarative and reusable way the application domain, using a shared vocabulary, and development tools so that players can work with different scenarios, are important requirements for the project a framework for Advanced SLA Management. In order to use a generic ontology, two important issues should be addressed. The first one refers to the representation of ontology. Although diagrams, textual description and equations can be used in the publication of the models, they are subject to typographical errors and the lack of definition of the initial conditions or boundary to the ontology. The second issue is related to implementation. The need to apply advanced methods limits the effective use of the ontology.

- *Support for future markets in Cloud Computing.* The design of a conceptual framework based on a generic ontology to support adaptable and extensible components in Advanced SLA Management. Only the use of an ontology, that allows a more flexible SLA negotiation, is not enough for the process of creation of new service agreements because it should provide custom annotation mechanisms to facilitate the reuse and modification of components. The ontology may be described using markup languages and should be validated for syntax errors and for the adherence to specification. However, semantic issues can not be effectively treated directly in a relatively simple schema and should be left to the implementation phase. This implementation phase can be represented through conceptual frameworks arranged in modules that may implement the execution of usage scenarios in Cloud environments allowing the creation of new services and the effective application of the concept of XaaS (everything as a service).

- *Description of a set of adaptive methods for Dynamic SLA Management.* It is an instantiation process and control of requirements independent of the implementation technology for the proposed framework. The use of a standardized and recommended format by W3C enables service agreements to be integrable and domain independent. Any effort to develop advanced management methods to SLAs in Cloud Computing should consider the cloud services industry capabilities. After introducing specific concepts for the definition of an advanced management for cloud SLAs we need to determine proof points to ensure that the concept is feasible for both technical perspectives as well as to business prospects.

This thesis focuses on three high-level objectives mentioned above, with the objective of defining a comprehensive solution for Advanced SLA Management in Cloud Computing. In the rest of the chapter, we discuss in more detail the specific contributions of this work.

1.3 CONTRIBUTIONS OF THE THESIS

The wide market acceptance that cloud computing gained, will make the current data centers work more together, converging into a global architecture of virtual distributed services - hardware, storage, processing, transformation, etc. - Allowing users to access and deploy applications on demand, being anywhere in the world and with a proportional cost to the contracted Quality of Service (QoS) parameters [10]. This integration of services and applications in the cloud, in turn, is also demanding new cloud-based services, for example, to provide interoperability, coordination and load balancing between departments or even the discovery of new services and applications. In addition to requiring more comprehensive service agreements.

This thesis deals with problems related to the availability of data and information in Cloud Computing when the data owner wants to migrate to this environment while maintaining effective control over the information. The specific contributions of the thesis are the aspects of SLA management illustrated above, i.e., a greater flexibility in SLA trading, the support for future markets in Cloud Computing, and the description of a set of adaptive methods for dynamic SLA Management. In the rest of this section, we illustrate these contributions in more detail.

1.3.1 ADVANCED SLA IN CLOUD COMPUTING

The first contribution of this thesis is related to the determination of Service Level Agreements that includes the needs of customers, satisfies the service conditions set by the providers and supports greater flexibility in SLA trading. The contribution of our work can be summarized as follows.

Problem modeling. The determination of a Service Level Agreements should find a good balance between the need for control over the information for the customer and the need to ensure the proper functioning of the cloud environment by providers. The specification of services in a generic way can effectively meet both the market needs as the possible operating restrictions brought in new scenarios. In this thesis, we build an approach for determining SLA that is defined with a generic ontology to describe the characteristics of different services. The peculiarity of our solution consists of a new model of SLA composition problem, which explores the representation of services and restrictions as service conditions and interprets its rules as truth assignments of Boolean variables and XML schema. This model is the basis for defining an effective solution to the SLA composition problem.

Efficient modeling. Thanks to the generic definition of the problem, the SLA composition may consider integrating different services trying to satisfy the market needs. To this end, we create an ontology which includes the restructured and adapted information from other ontologies. This generic ontology is used for interfacing the information between players and supporting needs presented by the customers. We take our modeling based on a generic ontology to formulate efficiently service conditions considered by providers. In addition, to meet the necessary operations for the SLA composition, our modeling also provides support for future markets in Cloud computing.

Freedom of choices. Given a set of needs presented by customers, our goal is to compute a selection of services that can match these needs. The logic is to determine utility functions that are compatible with the services, presenting an integration of the services with same nature and a set of values that can be manipulated by providers. Therefore, the utility functions of the data presented to end users can be changed according to the customer's preference and their practical needs. For this purpose, we define a structure that can receive established values from the providers and, new services and values from customers.

Shared liability. For the use of new services and values required by customers, considering their freedom of choice, we must supplement this data with responsibility information. However, the SLA composition only considers the responsibility of the providers. In this thesis, we present the concept of shared liability which allows the determination of different aspects of the services provided, the relevant dependencies to each one and responsibilities assigned to their values. First, we defined that service conditions are submitted by providers, considering internal and external dependencies between different services. Following, we describe an approach to determine how the responsibility is assigned in a shared way to support the monitoring of SLAs. In doing so, we provide a general solution, applicable to real-world scenarios to better meet customer needs services in cloud and not committing a breach of the agreements.

1.3.2 AUTOMATED FRAMEWORK TO CLOUD SLA MANAGEMENT

The second contribution of this thesis is the definition of a conceptual framework for managing the SLA. Since we use an ontology that integrates different characteristics we define concepts and modules that can handle these information. This framework considers the registration and research of SLA models and perform the SLA composition and its control. The contribution our work can be summarized as follows.

Simplified structure. We have identified and modeled the possible scenarios presented during the process of SLA composition so we could pull a simplified structure that perfectly fits the situations presented. We address a general problem, which is the creation of a SLA, observing the specific characteristics related to each Cloud service provider. With this approach we noted our concern to enable freedom of choice for the cus-

tomers using a simplified structure at the same time respects the existing ontologies and taxonomies in the market. Also, it is flexible enough to support new modules and properties.

Flexibility and adaptability. Based on the metrics identified for each scenario and the settings formally presented in our concepts we describe the modules of control and management for the framework. As the internal modules to the framework are based on a generic concept and a simplified structure is possible to ensure the flexibility to receive new information from both customers and providers. In addition the structure of the framework makes possible to adapt to other monitoring and management services for example, since their outputs represent information that can be manipulated by external modules.

1.3.3 DYNAMIC SLA MANAGEMENT

The third and final contribution of this thesis is a solution for dynamic management of information in a Service Level Agreement. To monitor the non-violation of adjusted values between the parties is necessary to observe the dynamic nature of some properties. Accordingly, our approach is based on the monitoring of different scenarios. The original contribution of our work can be summarized as follows.

Identification of demand for new services. Once we allow freedom of choice for customers, we give the possibility to request non-established services. In this sense we propitiate the identification of new services through the pooling of information and the monitoring of its implementation enabling the application of the concept of XaaS. In our approach, the framework provides a module for inventory of properties that controls the behavior of services. This enables providers identify new demands and eventually deploys the services to meet the market.

Use of natural language. Traditional solutions for the SLA composition assume that the values required for the services to be static and punctual, not considering the needs or the differences in perceptions between the parties, which implies that a simple difference of values makes the customer's request unanswered and that the provider lose market or have no knowledge of it. Such an assumption may result in a market restricted to the providers. We solved this limitation by proposing an approach using values described in fuzzy logic, this allows an approximation between the parties since they can use natural language to describe certain values and allows a greater range of service opportunities for providers.

Control of dynamic information. Analyzing the characteristics of some services in Cloud and the set of relationships between their properties is possible to identify dynamic aspects that influence the composition of services and consequently determine some particular situations for Service Level Agreements management. We present an approach that allows us to consider this behavior and demonstrates how these dynamic

changes can be used to monitor and control cloud services resources and possible violations of SLAs.

1.4 ORGANIZATION OF THE THESIS

In this chapter, we discuss the motivations behind the work proposed in this thesis, and illustrate our high-level objectives and main contributions. The remaining chapters are organized as follows.

Chapter 2 presents the state-of-the-art on Service Level Agreements in Cloud computing environments. It presents the needs of standardization and techniques used for the composition of SLAs, besides describing some frameworks used in the Cloud SLA scope also describes the use and representation of different ontologies that describe the different services in the Cloud.

Chapter 3 illustrates our solution based on the integration and generalization of the information presented by different services in Cloud, ensuring the necessary flexibility to ensure freedom of choice for service Cloud customers. In this chapter, we present the formal definition of our approach and the developed generic ontology that provides the basis for the solutions presented in the another chapters. This formal definition refers to conceptual modeling of the information in SLA composition, and represents values independent of domain.

Chapter 4 provides an overview of the proposed framework in this thesis. The main elements are discussed and the framework architecture is described.

Chapter 5 focuses on the request for issue of new services that have non-established values by providers. The chapter illustrates the possible scenarios for the use of the framework and our model to control and group new information, where sensitive information is characterized by market demand. Then, it describes how service providers can use this approach to develop its portfolio of services.

Chapter 6 addresses the problem of interpretation of values in Service Level Agreements. First, it describes how to identify the possibility of using fuzzy logic at different times in the determination procedure of a SLA. Then, it illustrates our approach to the use of natural language in the services requirements of customers and how this approach can also be used on the provider side.

Chapter 7 shows how the framework presented in this work can be used to control the changes of properties values in the Service Level Agreements. In addition, we describe how the dynamic behavior of certain services is handled by our approach. The parameters that must be monitored to ensure the stability of the agreement between the parties and also how this feature can be used for dynamic prediction and prospecting

for new Cloud services.

Chapter 8 provides an overview of our approach and summarizes the contributions of this thesis provides our concluding remarks and outlines directions for future work.

Appendix A reports a list of publications related to the work shown in this thesis.

2

RELATED WORKS

To define an advanced SLA management strategy in Cloud environment is necessary to understand how is the trading and composition of service agreements in different circumstances and presenting their use and control over various aspects.

This chapter provides considerations about the types of service agreements presented in Cloud Computing and highlights the problem of non-standardization to define SLAs, i.e., the lack of an unique and clear model to determine the relationships between different Cloud Computing areas. In addition, we present some considerations regarding frameworks of Cloud Computing and we describe the use and representation of different ontologies that describe the different services in the Cloud. Throughout this chapter we also present results of some surveys related to SLA management in Cloud Computing.

2.1 THE SLA IN CLOUD COMPUTING

A SLA on a Cloud Computing service must be negotiated and agreed with the customer to provide the features and requirements necessary to meet the SLO. From the customer's point of view an important issue is to have the assurance that all business requirements are met in the choice of new services.

In [11] the authors state that the parameters of services, metrics and functions should be included in a SLA and should be precisely specified in order to define the values of the service properties. In [12] the author points out that, when we have a SLA for cloud environments, certain topics should be clearly identified, discussed and negotiated in order to ensure the protection of information and business functions. Some of these aspects are: support for service interruption, security guarantees of information, services and systems, procedures for incident response, auditability to implement security, payment of compensation for losses and reliable certification (among others).

Although the term "SLA" has become widely known and used, it is common to find documents related to poorly written agreements with missing information and confusing definitions. This is not good for the provider, and not good for the customer as it gives room for discussions that could jeopardize the good relationship between the parties, and in severe cases even cause a contract termination (in case of external service providers) or the decision by outsourcing (in the case of domestic service providers).

We need to understand that a Service Level Agreement is much more than a document describing service time and problem solving. This is an agreement that should make clear all the guarantees that the service provider in relation to services that were hired, and how these service levels will be measured, reported and improved continuously. This requires the understanding of some concepts related to Cloud Computing.

Usually the Cloud structures are explored through a model "pay-per-use" [13] with guarantees provided by SLAs [14]. According to NIST - National Institute of Standards and Technology [15], the cloud computing concept is "a model for enabling network access, convenient, on-demand to a shared pool of computing resources (such as networks, servers, storage, applications and services) that they can be rapidly provisioned and released with minimal management effort or interaction with the service provider".

Furthermore, the service models in the Cloud Computing may vary according to the nature of the technology offered by providers. According to [16] the architectural models can be commonly referenced by the acronyms "IaaS", "PaaS" and "SaaS". We can describe these architectures with the following settings:

- *IaaS - Infrastructure as a Service* It is the model that is the provision processing, storage, networking and other computer resources that allows the user to execute and implement any software [15]. The user does not manage or control the cloud infrastructure where the service is provided, but have control over the operating system, installed applications and limited control over the network components. The service provider manages all infrastructures, while the customer is only responsible for other aspects of system deployment.

- *PaaS - Platform as a Service* is a service model of Cloud Computing that allows customers to implement and run infrastructure applications in the Cloud. These applications can be created using languages, libraries, services and tools supported by the service provider [15]. The user has no access or control of cloud infrastructure including network, servers, operating system or storage. Compared to traditional environments of application development, the use of PaaS strategy may result in a reduced development time and offers dozens of tools and services that allow rapid scalability application [17].

- *SaaS - Software as a Service* is a model that provides software systems for specific purposes that are available to users from multiple devices through a client interface. In the SaaS architecture the users do not manage or control the infrastructure they directly access the application [18].The only responsibility of the users is send and manage the data that the application will process and the

> stages of interaction with the application. Thus, new features can be automatically incorporated into the software systems without users noticing these actions, making transparent the development and updating of systems. Everything else is the company's responsibility that provides the service [19].

The companies began to adopt Cloud Computing when they realized that this shared infrastructure would reduce costs and make the process more efficient. Since companies do not need a large investment in equipment, manpower or expensive licenses, it is more advantageous to buy a custom package and pay only for what is used. The fact that everything is turning into a "service" is the essence of what is the *XaaS* (everything as a service): this concept encompasses all the other concepts and reflects a trend in IT in recent years that is turning products into services [20]. Although all architectures generally follow the designs described, it can be said that each service provider has a number of characteristics which vary from provide or not provide all models, the supported programming languages and other forms of services.

Analyzing the services offered, we can state that the main providers of commercial cloud services are Amazon (Amazon Web Services - AWS) [5], Google (Google App Engine - GAE) [21] and Microsoft (Microsoft Windows Azure) [22]. This statement is reinforced by name and size of the organizations involved and and by the potential investment to become the leader in this sector [23]. We describe these three service provides as follows:

Amazon Web Services - AWS provides the most complete set of support cloud computing services [24], even though commercial platforms are the oldest ones (launched in 2002). The Amazon Web Services consists of a set of systems, among which we can highlight processing, storage and monitoring. Amazon AWS can be classified as IaaS. The used processing service is the Elastic Compute Cloud (EC2) [25]. It is responsible for the provision and management of virtual server instances on Amazon's infrastructure. EC2 allows full control of instances of systems, being able to access and interact with each one of these, similar to conventional machines [26]. Is possible also choose the characteristics of each instance, such as operating system, software packages and configurations of machines, such as CPU, memory and storage.

Microsoft Windows Azure is Microsoft's proposed computing services in the Cloud. The service consists of a platform (SaaS) for running applications. The platform is composed essentially of three major components that form the core of the service; they are the computing units, storage space and the Fabric. The latter is a software responsible for managing and monitoring all resources of the data center, such as servers, load balancers, switches, routers. This component is also responsible for the management process of the application life cycle and maintaining satisfactory levels of services [27].

Google App Engine is the platform (PaaS) development that enables Web application being available on Google's infrastructure. This platform has many features for developers such as support for Java and Python languages, application models, APIs for integration with Google features and providing an environment with automatic

load balancing and adjustments [28]. Google also offers services ranging from systems for sending emails and editing images to APIs for authentication using an integrated account with Google, scheduling functions, cache and a recovery URL.

Several open source technologies are behind the commercial computing cloud services [29]. Therefore, there are community efforts to create middleware open source to cloud platform. These efforts include the softwares Eucalyptus, OpenNebula and Nimbus.

Eucalyptus: was developed by the University of California. Eucalyptus is an open source infrastructure that recreates Amazon's EC2 from the APIs provided by Amazon service [30]. This allows the customer to have its own structure Cloud (Private Cloud) and can scale to Amazon's service during a "peak demand".

A private cloud with Eucalyptus consists of at least one host and one or more clients or nodes. According to [31] the Eucalyptus architecture consists of four parts. 1) Cloud Controller (CLC): Top level that controls the cloud as a whole. 2) Storage Controller (Walrus): Top level that manages data traffic in and out of the cloud. 3) Cluster Controller (CC): intermediate level which is the communication bridge between CLC and NC. 4) Node Controller (NC): lower level controls instances of virtual machines in the nodes.

OpenNebula: is a set of open source tools to create private and hybrid clouds [30]. Just as Eucalyptus, it supports Amazon EC2 and elasticity. The OpenNebula is also modeled in a classic cluster structure [32]. A master node makes the queuing of tasks, performs scaling and submits jobs to the cluster machines. The work nodes provide the computational power to process jobs sent by the master node.

In OpenNebula it is also possible to add new work nodes and transfer instances of virtual machines between nodes. The architecture can be divided into three lines. 1) Tools: developed management tools using the interfaces provided by the OpenNebula core. 2) Core: contains the main virtual machine, storage, networking for Virtual Machines (VMs) and hosts management components. And 3) Drivers: drivers for different VMs, storage drivers and monitoring drivers and cloud services to the core.

In addition to allowing the creation of private and hybrid clouds, OpenNebula allows creating public clouds, where users can access the infrastructure through public APIs compatible with AWS.

Nimbus: similar to OpenNebula, it implements the AWS APIs. It specifically targets the scientific community [30]. The Nimbus offers features relevant to the community. It can be integrated into the Portable Batch System and Sun Grid Engine. Both tools are widely used by the scientific community to process tasks in large distributed infrastructures. The Nimbus offers features such as dynamic creation of virtual machine clusters [33]. It implements a compliant storage engine with Amazon S3 called Cumulus, designed to be used primarily as a repository of virtual machines; however, it is also possible to use it independently.

Considering the Cloud Computing definition in [34], the authors draw attention to the need to guarantee the supply of services in the cloud using Service Level Agreements (SLA). Cloud Computing gives users less control in the provision of services, so they need to take precautions in order to not suffer poor performance, long periods of inactivity or loss of critical data. The SLA becomes therefore an important part of cloud service delivery model [35].

By definition a SLA is a formal document negotiated between the parties involved in hiring a service [36]. A SLA aims to specify the minimum acceptable for the proposed service and is essential for managing the quality of services provided. A SLA is designed for each individual service and is made prior to purchase, before being invoked and used. According to [37] a SLA should include a description of the services, the guarantees given by providers and the actions and penalties to be observed in case of violation of these guarantees.

Within the SLAs there are SLOs (Service Level Objectives), which describe the actual topics to be observed and measured in a SLA. A SLO is a requirement that the service provider must offer. According to [38] the descriptions of SLOs highly depend on the service architecture to be provided.

Considering the presented concepts and the generalization of service agreements we can then present the life cycle of a SLA as the steps in [39]:

- *Negotiate*: An initial SLA is negotiated to document the desired requirements of the service. The customer and the service provider, which are the parties involved in the process, must accept the terms of the SLA that links them. They must also detail the responsibilities of each party and the resulting consequences of violation of the rules. This negotiation can consume lot of time representing a long process. In an automated SLA, the availability of an interface where the parties can discuss conflicting points of SLA can generate a quick agreement, benefiting more quickly to all;

- *Start*: The service is configured and started to meet the SLA. When a provider accepts an agreement, it must put in a queue and use a scheduling policy to set the order that will meet the services. In addition, the provider must also consider how to optimize the use of resources and how to preserve the QoS parameters that are guaranteed by the SLA. In this scenario, it is very important to consider the possibility of the arrival of new service requests and their priorities, even being already running other tasks, to meet them with the resources that meet the requirements as appropriate;

- *Evaluate*: The executions of services are monitored and evaluated to ensure that the terms of the SLA will be met. Considering the fact that a provider began providing access to resources, it should monitor the operation of these resources. The monitored information can be used to verify that the QoS attributes defined in the SLA are being fulfilled. Those involved that it should not only be interested in knowing only if a task is being performed correctly. Other information such

as breach of contract or usage statistics are also relevant to the verification of the SLA.

Based on this we can say that the identification of relevant information is important to describe faithfully the process. At any time, one of the parties to the contract may want to change the resource usage policies, typically to comply with some external demand arising from changes in context. Considering that changes will always exist while the system is running, it is important to consider this aspect, but despite the environment subject to change the behavior of processes should remain unchanged. That is, we need to ensure that after any migration, addition or removal of the system resources will continue to function properly. The use of resources must obviously generate a list describing which were used, to what extent and for how long, as well as relating the values agreed for the use of each of them in accordance with the definitions set forth in the SLA.

2.2 APPROACHES IN SLA COMPOSITION

From the point of view of customers, the existing SLA in commercial clouds like Amazon and Microsoft are simple because they are static and pre-defined by the providers [40]. The life cycle of the SLA will be described below: The first step performed by the customer is to find service providers according to their needs. The customers find the provider by searching the Internet, and then exploit the suppliers web site to collect more information. Cloud service providers offer static document of SLA. In this case, the following steps are the selection of monitoring and tracking service that normally occur with third-party tools [41].

Considering the SLAs in commercial clouds [42] and [6] describes that Amazon provides its users a SLA related only to availability. Also according to [6] in Windows Azure from Microsoft, the response time is considered, but this only occurs in some of the services. Besides not cover all the metrics needed for computing cloud services, these SLA proposals are just static documents.

In cloud computing middleware Eucalyptus provides a SLA based on regions or availability zones [31]. Similar to availability zones in Amazon AWS, if a SLA is violated is possible to migrate the services to other availability zones. However, there is no integrated monitoring to the SLA and the migration of services is not automatically triggered. Similarly, monitoring of Cloud Computing middleware only provide information on the state of VMs if they are pending, active or off [30]. For more effective monitoring is necessary to retrieve information directly from hypervisors. These deliver information as amount of CPU used, amount of memory available and used, and network traffic.

It can be observed that the existing solutions of cloud computing provide some level of SLA and service quality, but do not present dynamics in the negotiation of the SLA. The solutions of these SLAs are static. Cloud services are subject to load fluctuations and SLA violations are more likely to happen during these fluctuations. The nature of these variations are unpredictable and therefore a static SLA to withstand these conditions will not be efficient.

In the academic community, there are efforts to create protocols to automate the creation and monitoring of SLAs in Web services. The WSLA (Web Service Level Agreement Language) [43] is a protocol to define SLAs based on Web Services and XML, where it creates an XML Schema that includes the definition of the parties involved, the service guarantees and the service description. The WSLA has the following main components: "Parties", "Service Definition" and "Obligations" [38]. Parties describes who is involved in the service (client or provider). Service Definition describes the services connected to the SLA, representing the understanding of both parties on the parameters of the described service. Finally, Obligations defines the level of service that must be guaranteed with respect to the parameters defined in the Service Definition. The WSLA allows management of penalties or compensation when violations occur. However, it assumes that the SLA is already created and not allow the negotiations to create the SLA.

In [44] the authors specify an XML-based protocol, the WS-Agreement (Web Services Agreement Specification), for the establishment of Service Level Agreements and guarantees of offers between a provider and a Web service client. In this specification, an agreement involves multiple services and includes attributes for the parties, to refer previous agreements, service definitions and terms of warranty [45]. The specification is divided into three parts which can be used to specify an arrangement, a structure for specifying a contract template and a set of types of transactions for managing the agreement life cycle, including the creation, validity and monitoring states.

These protocols generally only handle the negotiation of a simple service with the basic message exchange. Auto-negotiation of multiple services and multiple steps still lacks maturity. In addition, both protocols are specific to Web Services and were not applied to Cloud Computing scenarios [46].

According to [47] most commercial providers offers SLAs for cloud computing services only as texts and transfers the activity of monitoring and controlling the SLA for customers. The authors present a new language for the composition of Service Level Agreements in Cloud named SLAC and add a comparison between the most known, which are: 1) SLA* [48] used in general purpose services, 2) SLAng [49], that is a domain specific language for IT services, 3) WSLA [43], 4) WSOL [50], an XML notation compatible with the WSDL (Web Services Description Language), 5) WS-Agreement [44] and 6) SLAC [47]. In the work presented by [46], the authors propose the Cloud Service Level Agreement (CSLA) which is based on WSLA and increased the number of comparison performed by [47]. As a result of this comparisons, the authors present the following information (Table 2.1):

Table 2.1: Comparison of the SLA languages (adapted from [46])

Features	WSOL	WSLA	SLAng	WS-A	SLA*	CSLA	SLAC
Cloud Domain	-	-	-	-	⊖	⊕	⊕
Cloud Service Models	-	-	-	-	⊖	⊕	⊖
Multi-Party	-	-	-	-	-	⊕	⊕
Broker Support	-	-	-	-	-	⊖	⊕
Ease of Use	⊖	-	⊖	⊖	⊖	⊕	⊖
Business Metrics	⊖	⊖	⊖	⊖	⊖	⊕	⊕
Price Schemes	⊖	⊖	-	⊖	⊖	⊕	⊕

According to the results presented in [46] and summarized in Table 2.1 we have: ⊕ represents a feature that is covered in the language, ⊖ is a partially covered feature and - when no support is provided for the feature.

As we can see by the results shown in Table 2.1, none of the SLA composition languages meet all the specifications required by the authors, especially with regards to service models in Cloud, ease of use and support for brokering. This shows that these languages are not enough to meet the new demands of the market, considering the transparency between the parties and the control needs of customers.

In addition to these SLA composition languages it is also important to relate the use of frameworks that seeks to provide SLAs for Cloud Computing services or, failing that, models that provide SLAs for Web Services.

2.3 FRAMEWORKS IN CLOUD COMPUTING

The frameworks discussed in this thesis were analyzed in relation to hiring interface, level of expertise of the metrics or SLOs, if has dynamic aspects, it has integrated monitoring and work with multiple metrics. In this sense, scientific proposals were chosen to have characteristics that somehow meet the requirements, but differ in important ways to the comparative analysis of the models.

Among the considered works, there are the LoM2HiS (Low Level Metrics to High Level SLAs) [51], the DeSVi (Architecture for Detecting SLA Violations in Cloud Computing Infrastructures) [52], the SRV (SLA-based Resource Virtualization) [53], and an approach to a general SaaS architecture for scientific software [54]. Finally, a comparison is made between the studied models and we added the results of different surveys found in the literature.

Some frameworks in the area of Cloud Computing were disregarded as is the case of CASViD [55] which is a framework that considers only the monitoring and detection of violations and the framework presented in [56] that only considers security aspects.

2.3.1 LOM2HIS

The LoM2HiS [51] allows the hiring of SLA from QoS requirements, also called high-level requirements, such as response time rather than low-level requirements or infrastructure such as amount of CPU or memory. Figure 2.1 shows the architecture LoM2HiS.

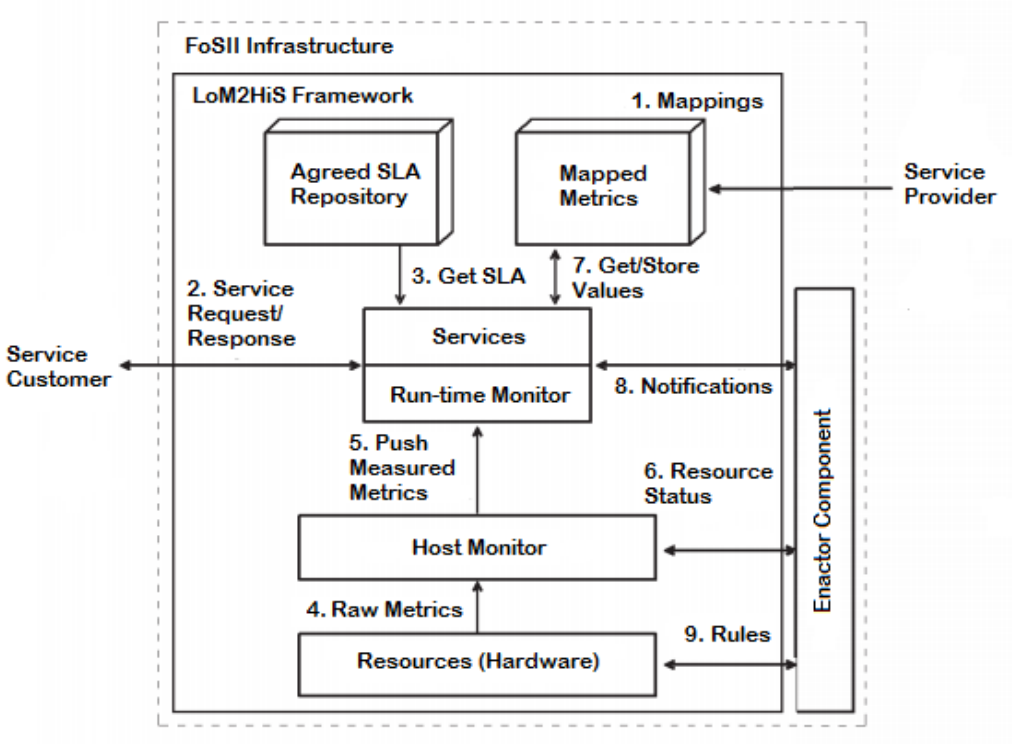


Figure 2.1: LoM2HiS Architecture (adapted from [51])

In Figure 2.1 we have that the service component is the application layer, where services are implemented using a Web container and the run-time monitor is a component designed to monitor the services based on negotiated and agreed SLAs [51]. Following the steps set out in [51] we have:

Step 1: After agreeing to terms of Agreement the service provider creates mapping rules for LoM2HiS using Domain Specific Languages (DSLs) that are simple languages that can be adapted to a particular problem domain;
Step 2: The client requests the delivery of an agreed service;
Step 3: The run-time monitor loads the SLA repository with the SLA agreed;
Step 4: The provision of services is based on the available computing resources. The metrics of these resources are measured by monitoring agents;
Step 5: The host monitor extracts the metrics and submit to the run-time monitor;
Step 6: The host monitor reports the status of the resource to the enactor component;

Step 7: Upon receipt of the metrics, the run-time monitor and the low-level metrics monitor compares based on pre-defined rules to determine the equivalent SLA. The result is stored in a mapping store;

Step 8: The run-time monitor uses the values mapped to monitor the status of execution of the services. If there are threats of future violations of the SLA, the enactor component is notified for preventive action;

Step 9: The decisions of enactor components run directly on the available resources.

The LoM2HiS not have arrangements for the negotiation of SLA since it assumes that the negotiation process has been completed and the SLAs already previously agreed are stored in the service provisioning repository.

2.3.2 DESVI

Regarding the violation of SLA, the DesVi framework [52] performs the detection of SLA violations by monitoring cloud computing infrastructure resources. The DeSVi allocates the resources needed for the service based on the user's request and organizes the implementation thereof within a virtualized environment. Once the level of service is pre-set, it is possible to detect potential SLA violations. According to [52], knowledge bases are used to manage the SLA violations and these knowledge bases are implemented using learning techniques. In [52] the authors also present the execution results in heterogeneous environment showing that DeSVI is able to monitor and prevent SLA violations, considering workloads and different measuring ranges. The service life cycle of DeSVi includes activities such as trading SLA, allocation of resources, resource monitoring and detection of SLA violation. The DeSVI architecture is shown in Figure 2.2.

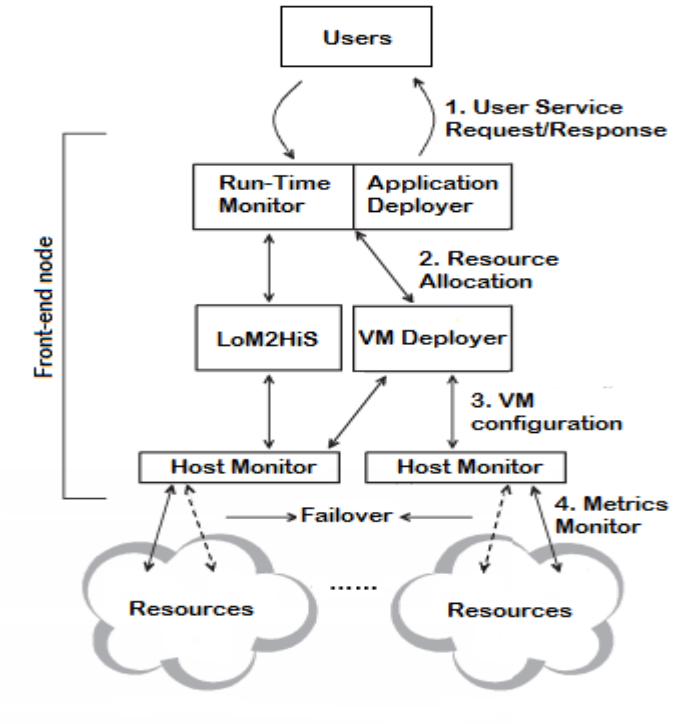


Figure 2.2: DeSVI architecture and interaction between components (adapted from [52])

In the Figure 2.2 the top layer is the users who request services to a service provider and according [52] the interaction can follow the following steps:

Step 1: The user requests service provisioning to the cloud provider;
Step 2: The provider handles user service requests based on the SLA negotiated and agreed previously. The application allocates the necessary resources for the requested service and organizes its deployment on VMs;
Step 3: Then it is performed the deployment and configuration of virtual machines;
Step 4: The host monitor notes the resource pool metrics comprising virtual machines and physical hosts. LoM2HiS manages the relationship between the metrics of resources and SLAs.

In Figure 2.2, the redundancy in the monitoring mechanism is indicated by failover arrow [52]. The host monitor uses monitoring agents that are incorporated into the resources to monitor the metrics. Such agents transmit the monitored values for the other players on the same set of resources, enabling access to the status of resources for any service.

The DeSVI has the capacity to carry out the monitoring and detection of SLA violations with automatic deployer of VMs. However, the DeSVI does not address applications with large variability of resource consumption [52]. In addition, the DeSVI does not address monitoring and SLA violation in the level of applications (only infrastructure).

2.3.3 SLA-BASED RESOURCE VIRTUALIZATION (SRV)

The SRV [53] has three main components: a meta-trading component for the management of a generic SLA, an intermediate component to a diversified management and an automated deployment service that uses virtualization capabilities. Figure 2.3 shows a service architecture that illustrates these components.

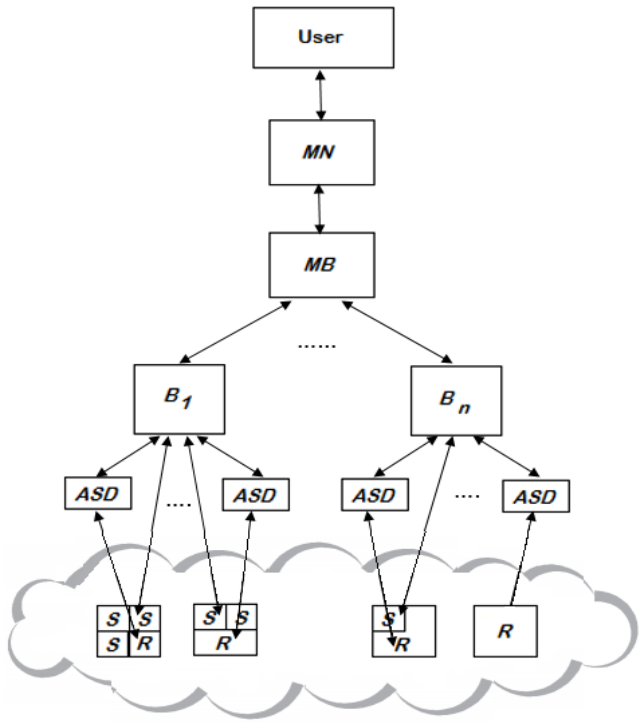


Figure 2.3: SRV Architecture (adapted from [53])

In Figure 2.3 we have "User" that is the person who wants to use a service. "Meta Negotiator" (MN), a component that manages the level of service. This component mediates between the user and the Meta-Broker, selects the appropriate protocols for agreements and negotiating the creation of the SLA and the treatment of violations. "Meta-Broker" (MB) component that has the function of selecting an agent capable of deploying a service with the requirements specified by the user. "Broker" (B) that interacts with the physical and virtual resources and when the service needs to be deployed it interacts directly with ASD. Auto Service Deployment (ASD) installs the required service on the selected feature on demand. "Service" (S) is the service that users want to deploy and/or run. "Resources" (R) are the physical machines into virtual machines can be deployed/installed.

This architecture shows that the negotiation of the agreement and service deployment are closely related. The SRV contains a base model for negotiating the SLA, however, in addition to treating only the negotiation it is generic for virtualized environments and has not been evaluated for Cloud Computing environments.

2.3.4 SLA FOR SCIENTIFIC RESEARCH CLOUDS

In [54] the authors propose a SLA architecture for SaaS scientific research clouds using algorithms estimates that are able to assess the viability of the SLA in advance the use of services. In this work the authors use machine learning principles to create algorithms that control the provisioning of virtual machines to maintain the SLAs. Figure 2.4 shows the SLA SaaS architecture for scientific research clouds.

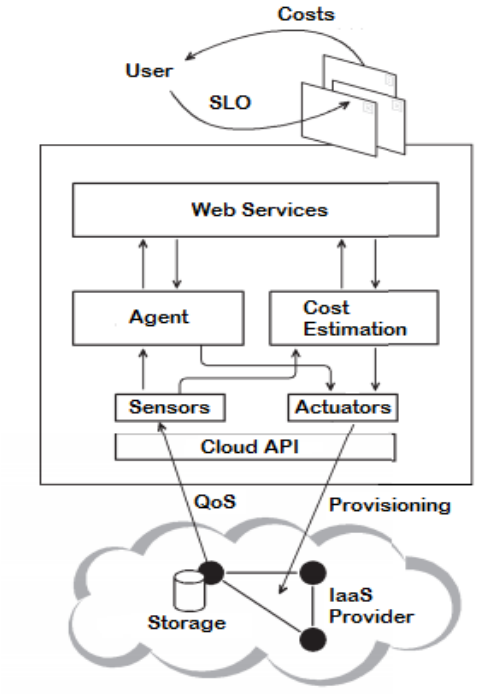


Figure 2.4: Architecture for Research Clouds (adapted from [54])

The cloud infrastructure can be seen at the bottom of the Figure 2.4, where an IaaS provider offering instances of virtual machines within their data center. Just above, a layer called cloud API encapsulates access to infrastructure. The storage containing the data of the applications are located within the cloud. In the next layer, there are sensors and actuators. The components interact with the cloud and running instances. Especially sensors are dependent on the application because they are used to control the state of application. Actuators are used to create, modify, configure and terminate instances of virtual machines. There is also an agent and a cost estimation module. The agent handles the service while running and ensures the service agreement. The cost evaluation module is initially used to check the feasibility of user requests and estimate costs. On top of the architecture provided a Web Service interface, providing access from a user interface and used to communicate with the agent and cost estimation module. The estimated costs are presented to the user and the architecture also provides the allocation of resources in advance. After this, the Agent module invokes the work. Submission implies that the user and the service provider agree with the

SLA. The task of the agents is to enforce compliance with the SLO and minimize the resources used.

This architecture was designed for scientific research SaaS providers and HPC applications (High Performance Computing). In this case, a job is not interrupted until it is completed. Thus, the SLA can only be accepted once and is valid until the end of execution of the work. There is no dynamic renegotiation of the SLA.

2.3.5 COMPARISON OF STUDIED FRAMEWORKS

Considering the frameworks studied in this section, we highlight some aspects: hiring interface, level of expertise of the metrics, it is dynamicity, integrated monitoring, which cloud model is served, strengths and weaknesses.

The SLA hiring interface refers to how the user interacts with the model in hiring the SLA. Requirements or application metrics are usually expressed in low-level or technical language, such as packets, number of bytes received and sent. Furthermore, these metrics can be measured using different units of measurement. Since the goal is to compare how the requirements of customers and providers are negotiated, different units of measurement and expressed in technical language can hinder this comparison. In this sense a requirements of speech interface to QoS parameters, also called the highest level, is expressed in order to be less technical and how it is directed to the evaluation of quality, allowing comparisons with less complexity. The LoM2HiS and the SLA for scientific research clouds have high-level hiring interface.

Considering the hiring metrics used, none of the evaluated models have specific metrics for cloud computing. Furthermore, the models studied using the metrics in isolation from each other. A qualification system could prioritize actions when there are violations of specific metrics by assigning different weights, priorities or degree of importance to a group or type of metrics.

The dynamicity indicates whether the model allows the negotiation and renegotiation of the SLA dynamically, following the elastic behavior of the Cloud. Between the models studied the DeSVI is the only one that is designed to handle the life cycle of the SLA, including trading activities, allocation of resources based on monitoring and forecasting of violations using learning techniques.

Except for the SRV, the studied models perform monitoring of integrated SLA resources. A summary of this evaluation can be seen in Table 2.2, which are also given the strengths, weaknesses and the cloud model where these frameworks are used.

Table 2.2: Comparison of studied frameworks

Framework	Interface	Metrics	Dynamicity	Monitoring	Strengths	Weaknesses	Cloud Model
LoM2HiS	high-level	Only infrastructure, does not work with multiple metrics	Static	yes	Allows the SLA contract using higher level language	Focus on language parser for the requirements for the metrics, does not allow the negotiation of the SLA	IaaS
DeSVI	low-level	Only infrastructure, does not work with multiple metrics	Dynamic but not supports elasticity	yes	Automatic deployment of VMs	Does not address applications with large variability of resource consumption, does not address monitoring and SLA violation in the application level	IaaS
SRV	low-level	Generic for virtualized environments, does not work with multiple metrics	Static	not	Negotiation	It is generic for virtualized environments so it does not address cloud specificities	N/A
SLA for scientific research	high-level	Does not allow the renegotiation of the SLA, does not work with multiple metrics	Static	yes	Cost Estimate	Does not address dynamic negotiation of SLA	SaaS

It is possible to realize that the framework LoM2HiS [51] allows the transformation of application requirements (low-level) in quality requirements (high-level) for hiring the SLA, but does not allow the negotiation of the SLA. The DeSVi [52] allows the detection of SLA violations through monitoring capabilities, but does not address large resource consumption variability (elasticity). The framework SRV [53] is a base model for SLA negotiation, but was not applied to cloud computing. The SLA architecture for search cloud [54] was designed for SaaS providers of scientific research and HPC applications (High Performance Computing). In this case a job usually to be started is

not interrupted until it is completed. Thus, the SLA can only be accepted once and is valid until the end of execution of the work.

2.4 SLA MONITORING

To ensure the life cycle of the SLA, it needs to be checked periodically [57]. For this it must be monitored to ensure it is still valid and feasible. The result of monitoring will inform the SLA may be terminated or renegotiated. This feature should lead to optimization of the execution of an instance of an application, task or service based on SLA parameters in order to maximize the likelihood of SLA satisfaction. The requests submitted to a SLA, are processed to select the best host among all available. The best setting for a host depends on the state of a number of variables in the system, such as available resources, the resources that are needed to meet the SLA requirements and further optimization goals.

Some goals are directly related to the knowledge of the requirements of a SLA, such as reducing the completion time, minimizing costs, maximizing the probability of success [58]. While other objectives are related to the system status, for example the workload balancing. The monitoring process of running an instance of a service, related to the definitions of a SLA, is meant to verify whether the contract is being fulfilled or not. During execution, if any parameter associated with the SLOs, which are effectively the topics to be measured within the SLA, reaches a threshold value, identifies the threat and recovery actions can be activated in order to preserve the SLA or even minimize the consequences of an effective breach of contract.

An interesting option for monitoring is to send threat alerts to the provider enabling him to take steps to try to prevent the violation. Another option for this feature is the ability to provide information to the user who signed the SLA, informing it about the current status of the SLA, making the process more transparent.

To demonstrate these features we present a survey on Cloud monitoring that was carried out in [59] where the authors made comparisons between commercial monitoring platforms and open source monitoring platform. To perform these comparisons the authors found features that must be submitted by monitoring platforms. According to [59] these features are:

- *Scalability*: A monitoring system is scalable if it can cope with a large number of probes (according to [60]);
- *Elasticity*: A monitoring system is elastic if it can cope with dynamic changes of monitored entities [60];
- *Adaptability*: In [60] the authors states that a monitoring system is adaptable if it can adapt to varying computational and network loads;
- *Timeliness*: A monitoring system is timely if detected events are available on time for their intended use [61];
- *Autonomicity*: According to [62] an autonomic monitoring system is able to self-manage its distributed resources by automatically reacting to unpredictable changes;

- *Comprehensiveness*: A monitoring system is comprehensive if it supports different types of resources, several kinds of monitoring data, and multiple tenants [63];
- *Extensibility*: It is extensible if such support can easily be extended [64];
- *Resilience*: According to [65] monitoring system is resilient when the persistence of service delivery can justifiably be trusted when facing changes;
- *Reliability*: A monitoring system is reliable when it can perform a required function under stated conditions for a specified period of time [66];
- *Availability*: According to [66] a monitoring system is available if it provides services according to the system design whenever users request them;
- *Accuracy*: A monitoring system is accurate when the measures it provides are as close as possible to the real value to be measured [59].

The result of the comparison of these features are presented in Table 2.3.

Table 2.3: Comparison features of cloud monitoring platform (adapted from [46])

Platform	**Scalability**	**Elasticity**	**Adaptability**	**Timeliness**	**Autonomicity**	**Comprehensiveness**	**Extensibility**	**Resilience**	**Reliability**	**Availability**	**Accuracy**
CloudWatch [67]		⊕		⊕			⊕				
AzureWatch [68]	⊕		⊕		⊕		⊕				
Monitis [69]						⊕					
LogicMonitor [70]	⊕	⊕				⊕					
Aneka [71]	⊕	⊕									
GroundWork [72]						⊕					
Nagios [73]							⊕				
OpenNebula [74]	⊕		⊕								
Nimbus [75]					⊕						
PCMONS [76]							⊕				
DARGOS [77]			⊕				⊕				
Hyperic-HQ [78]	⊕					⊕					
Sensu [79]		⊕					⊕				

The symbol ⊕ represents a feature covered by the monitoring platforms, as we can see from the results shown in Table 2.3, none of the monitoring platforms meet all

established features. A complementary work to this is presented by [80]. Figure 2.5 shows *ipsis litteris* the results presented by [80] once identified the realization of comprehensive surveys in the area of Cloud Computing (this is not the scope of this thesis). These results provide the basis for the identification of gaps in SLAs management in Cloud.

Cloud based monitoring tool analysis.

Capability/features	Percentage implemented	CloudKick	Nimsoft	Monitis	Amazon Cloud Watch	Azure Watch	PCMONS	Boundary app. monitor	mOSAIC	CASViD
Scalability	78%	yes	yes	yes	yes	yes	no	yes	no	yes
Portability	56%	limited	yes	yes	no	no	no	yes	yes	yes
Non-intrusiveness[a]	94%	yes	yes	yes	yes	yes	limited	yes	yes	yes
Robustness	67%	yes	yes	yes	yes	yes	no	yes	no	no
Multi-tenancy	44%	yes	yes	no	yes	yes	no	no	no	no
Interoperability	33%	no	no	no	no	no	yes	no	yes	yes
Customizability	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Extensibility	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Shared Resource monitoring	78%	yes	yes	yes	yes	yes	no	yes	no	yes
Usability[a]	94%	yes	yes	yes	yes	yes	limited	yes	yes	yes
Affordability	50%	limited	limited	limited	no	no	yes	no	yes	yes
Archivability	78%	yes	yes	yes	yes	yes	yes	yes	no	no
Verifiable measuring	0%	no	no	no	no	no	no	no	no	no
Resource usage metering	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Service usage metering	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Service KPI monitoring	0%	no	no	no	no	no	no	no	no	no
QoS	89%	yes	yes	yes	yes	yes	no	yes	yes	yes
Risk assessment	92%	yes	yes	limited	yes	yes	yes	yes	yes	yes
Component status identification	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Service load monitoring	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Configuration verification	78%	yes	yes	yes	yes	yes	yes	no	no	yes
Configuration drift identification	78%	yes	yes	yes	yes	yes	yes	no	no	yes
Configuration effect monitoring	100%	yes	yes	yes	yes	yes	yes	yes	yes	yes
Security breaches monitoring	56%	yes	yes	no	yes	yes	no	yes	no	no
User access control	67%	yes	yes	yes	yes	yes	no	yes	no	no
User activity monitoring	0%	no	no	no	no	no	no	no	no	no
Secured notification	33%	no	yes	no	yes	yes	no	no	no	no
Secured storage	33%	no	yes	no	yes	yes	no	no	no	no
Service dependency	11%	no	no	no	no	no	no	yes	no	no
Percentage covered by tools		78%	**87%**	70%	85%	85%	52%	73%	54%	69%

[a] We note that some of these capabilities are somewhat subjective. With that in mind the evaluation presented here is based on the weight of opinion as reflected in the reviewed literature.

Figure 2.5: Survey results presented by [80]

As we can see in Figure 2.5 in this work the authors added more characteristics, such as: "Shared Resource Monitoring", "Service KPI Monitoring", "Service Dependency" among others. In addition to performing the comparison, they added other monitoring tools and compared with a cloud resource ontology, i.e., mOSAIC [81] [82].

Despite the research carried out yet they are perceived gaps in relation to the life cycle of a SLA especially when they are considered different metrics. This can be further strengthened by the lack of standardization in the sense that each type of service in Cloud has its peculiarities. So we must understand the market demands and the clever use of any available resource in the cloud environment.

2.5 MANAGEMENT OF CLOUD PROPERTIES

The increasing need for flexibility and scalability in the use of computing resources considering attractive costs and independence of devices results in the search for the adoption of Cloud Computing environments in several areas [3]. Although the benefits envisioned for this migration can be huge, this computing paradigm still presents a large number of uncertainties regarding system faults and their management. As a results of these uncertainties, there is a considerable concern raised by customers [83], especially when it dealing with the reliability and availability of services in Cloud Computing [84].

Some studies reported in the literature, such as [85] and [86] also identifies that the use of commodity components can expose the hardware used in Cloud environments to conditions that were not originally designed. Furthermore, due to the highly complex nature presented in an infrastructure of this type, many data centers, even if carefully manipulated, managed and protected are subject to a large number of failures. [87]. Of course, the occurrence of these failures reduces the overall reliability and availability of the service. As a result, fault tolerance techniques become very important for customers and service providers to ensure proper and continuous operation of the system.

According to [16] Cloud Computing architecture comprises four distinct layers, that are:

1. Physical resources, like most computing architectures are considered the lowest layer of the stack, on it are embedded virtualization tools to form the layer of Infrastructure-as-a-Service (IaaS);

2. The IaaS layer typically uses virtualization technology to maximize the use of physical resources and ensure the quality of services;

3. The layer above the IaaS connecting all the user-level middleware tools is known as Platform-as-a-Service (PaaS);

4. Applications at the user level that are built and hosted on the PaaS layer comprise the Software-as-Service layer (SaaS).

The failure occurrence in a particular layer causes impacts on the services offered by the layers above it [87]. For example, if faults occur in the physical hardware or virtualized structure of the IaaS layer, these failures will affect the majority of services in PaaS and SaaS layers. Similarly, a failure in a middleware in the PaaS layer can produce errors in the instantiated software services in the SaaS layer.

As these layers are interdependent, it is necessary that they are linked between each other as well. To do so, in this section, we present some approaches related to metrics in Cloud SLA and the basic concepts relating to its modeling, are also discussed the levels at which the semantics can be express and made a description of ontologies that describe the different services available in Cloud.

2.5.1 METRICS IN CLOUD SLA

As seen earlier in cloud computing there are three levels of service, IaaS, PaaS and SaaS. In [37] the authors list the specific and most important metrics for a cloud computing scenario. As can be seen in Table 2.4, a model of SLA needs to consider these metrics and the terms of each level of service showing a list of requirements that can be comparable in terms of SLA.

Table 2.4: SLOs or QoS requirements for clouds (adapted from [37])

Cloud Model	Metrics
IaaS	CPU quantity, memory size, bandwidth, boot time, storage capacity, maximum and minimum number of servers per user, time to increase and decrease the number of servers
PaaS	Integration with other platforms services, scalability, cost, versions of servers and browsers, simultaneous number of developers
SaaS	Usability, customization for different types of user, time available services, capacity for a large number of users
General	Availability, performance, reliability, monitoring methods, service cost and how is it calculated, encryption, security, communication (flow, load balancing) support methods to services, privacy, location and legislation

It should be noted that when dealing with XaaS we will have infinite possibilities. In many works, terms typically treat the IaaS model. However, it notes that the terms of the SLA to PaaS and SaaS Clouds are not widely treated. This is because these specific metrics are qualitative, such as the "reliability" and "usability" and therefore difficult to compare.

Another problem encountered in the treatment of multiple metrics concerning how to quantify the relative importance of each of them, plus the fact that they possess different degrees of importance. Although it is not possible to assert that there is a commonly accepted method for setting weights, there are several proposals in the literature for these procedures. Generally the techniques for comparing metrics in Cloud Computing are used for selection of Cloud services based on QoS requirements. Among these techniques we can mention the order of the metrics with the scale and distribution of points [88], the fuzzy logic [89] and the procedures based on pairwise comparison with technique Analytic Hierarchy Process (AHP) [90]. These techniques aim to facilitate the solution of decision-making related to complex problems. Through them, weights and priorities are derived from a set of subjective judgments made by evaluators or participants involved in the selection of services.

In [91] the authors present a survey with the evaluation of different techniques used in the selection of Cloud Services based on metrics of QoS. The results of this work are summarized in Table 2.5.

Table 2.5: Summary of metrics evaluation techniques (adapted from [91])

Evaluated Techniques	**Limitation**
Ranking of Cloud Services using AHP [92]	All QoS requirements are not implemented
An approximate Markov chain model [93]	Not suitable for burst arrivals
Cloud Monitoring System for QoS [94]	Failed to calculate communication cost
Optimal resource allocation replica for maximizing revenue [95]	Not suitable for sensitive QoS applications
Generic QoS framework for Cloud workflow systems [96]	Complex problems such as monitoring and violation handling occurs
Personalized QoS ranking prediction [97]	Accuracy of ranking method has to be considered
AHP based ranking mechanism [98]	Non quantifiable QoS attributes are not used
Cloud Monitoring System for Virtualization [99]	Single QoS Parameter is considered
Software agent based automated service negotiation [100]	Multiple interactions are not possible
A Queueing network model with infinite queue [101]	Only response time is considered as a major factor
Algorithm for resource allocation [102]	Security problems occurs
A scheduling heuristic with multiple parameters [103]	Response time and performance parameters are not used
QoS-aware service selection algorithms [99]	Service Provisioning problems are not overcomed
Profit Balancing and pricing model for QoS [104]	Utilization is not considered for computational cost
Multi-Constraints Path problem [105]	NP-Hard problem occurs
Delay Constrained Least Cost (DCLC) [106]	QoS constraints are not used
AHP hierarchy using SMI architecture [107]	Ranking Algorithms can be deployed to rank infrastructures
A framework for performance monitoring and analysis tools [108]	After analyzing the services can be ranked
A framework to compare the performance of different Cloud services [109]	Authors must deployed more constraints for comparison
AHP hierarchy for web services [110]	VM capacity parameter is not used
Energy efficient resource allocation and scheduling algorithms [111]	QoS parameters are not considered as a major constraints
Business Rules for maximise the revenue of Providers [112]	Customers are not satisfied without QoS Requirements
Admission Control and Scheduling algorithm [113]	Only fewer QoS constraints are considered

Other methods to define weights in comparing metrics are also cited in the literature, such as decision trees [114], artificial neural networks [115] and genetic algorithms [116], but these represent enormous complexity for the treatment of multiple metrics. Decision trees, for example, become more complex as the number of factors increase. As regards the assessment of weights, whenever necessary to express the priorities of a particular group of metrics or criteria, the pairwise comparison method is strongly recommended [117].

Considering the qualitative aspects presented by different metrics and that the terminology of cloud service measurements is not well defined, [118] introduces the concept of abstract metric in combination with the concept of concrete metric. According to definitions presented in [118] an abstract metric is "a collection of elements that defines the expression of a specific metric for a given metric category" and a concrete metric is "a collection of elements that complete an abstract metric definition by linking the metric to its primary abstract metric and assigning specific values to the rule(s) and parameter(s) defined in the abstract metric definition".

Besides this, [118] also presents the context definition in Cloud Computing as "the circumstances that form the setting for an event, statement, or idea, in which the meaning of a metric can be fully understood and assessed".

2.5.2 ONTOLOGIES FOR CLOUD SERVICES

In order to understand the relationships between the various metrics presented on a Cloud SLA it is necessary to understand how these metrics can be modeled conceptually and how they are described in different ontologies presented in the area. According to [119], conceptual modeling has been characterized in several ways. However, an important definition that we consider in this thesis is offered by [120], which presents the conceptual modeling as a process to formulate and collect conceptual knowledge about an Universe of Discurse and document the results in the form of a conceptual scheme. The Universe of Discourse refers to the set of all entities of interest being modeled and its use is common in logic and mathematics.

This conceptualization process, promoted by modeling, aims to describe a domain (or universe of discourse) through the entities that compose the relationships between these entities. In [121] the process of modeling is presented in detail, including questions relating to the representation of semantic models.

A model is an abstraction of something that omits details that are not essential and able to deal with complex situations and objects [122]. Through abstraction, some real-world aspects are removed or simplified. The emphasis is placed on essential features, creating a vision, that is naturally incomplete or partial, of the environment or modeled context. Normally the modeling is done through an analysis process in which the whole is reduced into component parts that can be addressed separately, in a simpler way.

The conceptual modeling is the process in which knowledge of a domain is organized into levels of abstraction in order to gain a better understanding of the domain, encapsulating details. Thus, the description of an object X in a N_1 abstraction level contains more details than the description of the same object X in a N_2 level of abstraction, if N_2 is at a level higher than N_1.

The models, being an abstract representation, require some representation made by signals - such as icons, images, objects, symbols and tokens. In the context of a particular model, these signals have a form and an associated meaning. The semantics studies the aspects of meaning, while the syntax is related to the form. The representation of a model, therefore, is made using a modeling language, in turn, has an associated syntax

and semantics. A metamodel is a model that establishes the grammar of the language used to build other models.

From the construction of a model, we can get information that is of interest, for example, for the determination of an agreement between the parties that considers informations relevant to the domain. Information is an informal abstraction [123] that represents something significant in a certain context. The information may be represented by data. At a higher level of abstraction there is the knowledge. Knowledge can be characterized as information combined with experience, context, interpretation and reflection [124]. According to [125] there are basically two types of knowledge: tacit and explicit. Tacit knowledge is that available to people and that is not formalized in practical ways. The explicit knowledge is that which can be stored, for example, in documents, manuals, databases or in other media.

With regard to models, formal models are constructed with formal languages, or languages that have strict rules for its construction and interpretation. The rules for the construction define the syntax of the language, while the rules for the interpretation define its semantics.

The syntax works with the formal and structural relations between signs or tokens, as well as the production of new signals or tokens [126]. The syntax of a language involves the definition of the set of reserved words, their parameters and the correct order in which the words are used in an expression. An XML file, for example, used for interoperability and integration between systems, must have a precise syntax. If syntactical rules are not observed, the file can not be processed.

The semantics studies the relationships between the system of signs and its meaning [127]. The goal of semantics is totally different of syntax. This works with the formal structure in which something can be expressed while the semantics is concerned with what something means. One aspect about the semantics, computationally, is to define a formal representation language to capture the semantics in a way processable by machines, achieving a consistent interpretation.

According to [127] and [126] a model that has some form of formal semantics is more expressive than one featuring only implicit semantics. One way to add some semantics to the models is through metadata.

Metadata may exist in different levels. These levels are not mutually exclusive and include information on the content, structure and semantics of the data. Syntactical metadata describe no contextual information about the content, usually providing general information (e.g., document size, creation date, etc.). Structural metadata on the other hand, provide information about the structure of data, independent of the content. They describe how items are arranged in the document and the rules for this organization. An XML schema, for example, shows the structural metadata of an XML document. Metadata add semantic rules, relationships and restrictions on syntactic and structural metadata.

Metadata describe semantic information about the data that are important in a given context or domain, allowing a certain interpretation. Semantic data enable a way to high precision research and allows interoperability between systems or source of heterogeneous data. These data are used to provide the meaning of the elements de-

scribed by syntactic or structural metadata. An important aspect in creating a semantic meta-model for the data, is the possibility of using the inference capability for logical conclusions based on the meta-model, according to the semantic level adopted [126].

[126] points out that: "depending on the approach, models and methods used to add semantic metadata four representations can be used to organize the concepts that semantically describe the terms: controlled vocabularies, taxonomies, thesaurus and ontologies". In this thesis we discuss the characteristics presented by ontologies, since we use this type of representation to describe our approach to the determination of Cloud Service Level Agreements.

In the area of computer science, an ontology defines a formal and explicit specification of the terms of a domain and the relationships between them [128] [129] [130]. An ontology provides a mechanism to capture the common understanding of objects and their relationships in a certain area of interest, and provides a formal and manipulable model of a domain. The formal specification of the meaning of the terms used enables the creation of new terms by combining existing and allows integration with other ontologies.

The study and the use of ontologies in software were popularized with the semantic web idea introduced in [131]. In this article the authors present the semantic web vision "as an extension of the current web in which information is given a well-defined meaning, enabling the cooperative work of computers and people". In this context, the most common type of ontology consists of a taxonomy and a set of inference rules, which allow to capture the knowledge that is not explicit in the taxonomy.

The ontologies can extend the hierarchical relationships of taxonomies, allowing horizontal relationships between terms in a structure type graph. Thus, the ontology facilitates the modeling of typical data requirements of the real world. According [132] different formalisms for knowledge representation exist for implementation of ontologies. Although components of each are different, a minimum set is common to all:

- *Classes*: represent domain concepts; They are defined by terms generally organized in a taxonomy.
- *Relations*: represent a kind of association between classes. Generally associations are binary, with the first argument being called domain and the second argument being the limit (range). The relationships are instantiated according to the available knowledge about the domain. The binary relationships are also used to express attributes or properties of classes. Attributes are different relationships, because they are limited to one type of data (not a class).
- *Instances*: represent elements or individuals in an ontology.

An important property of ontologies is that their representation allows the computer to process similar, being based on logical languages, which allows the formal definition of semantic concepts. In this thesis we present two of the most commonly logics languages to represent ontologies, they are OWL (Web Ontology Language) and SWRL (Semantic Web Rule Language).

OWL

According to [131], a language for representing ontologies must have a well-defined syntax and semantics, have supported the inference, be efficient and expressive. In 2004 the W3C (World Wide Web Consortium) presented the OWL (Web Ontology Language) [133], an ontology language for the Web, based on description logic. OWL is designed to meet the needs of an ontology language for the Web, as the languages that preceded it have some limitations. XML provides a syntax for semi-structured documents [134], but does not associate semantics to the markers. RDF (Resource Description Framework) [135] standardizes the definition and use of metadata, but has a very simple data model, based on triple (subject, predicate, object), to represent the relationship between resources. The RDF Schema provides a type system for RDF, which allows users to define resources with classes, properties and values. However, some features such as cardinality constraint, class disjunction and local scope properties can not be expressed in RDF Schema.

The objective of OWL is to provide an ontology language that can be used to describe, in a natural way, classes and relationships among classes of documents and Web applications. The terms used in an ontology must be written so that they can be used by different software. According to [121] OWL distinguishes between constructors and axioms. OWL constructors are primitive used to specify new classes and axioms are the primitives to make additional statements about classes and properties. The OWL dialects provide builders based classes in descriptive logic. These builders use the data types defined in XML Schema.

As it is based on descriptive logic, OWL enables the use of inference engines allow explicit knowledge that are implicit in a knowledge base. As a result, an OWL document should not be considered only from the point of view of its syntax, but also of their semantics. This means that two superficially different documents in syntactical terms can express the same knowledge if they legitimize the same inferences.

SWRL

Considering the number and variety of existing systems, SWRL (Semantic Web Rule Language) [136] is a proposal for standardization of language rules, aimed at interoperability between these various systems. The SWRL is based on a combination of two sub-languages OWL (OWL Lite and OWL DL) with a RuleML sub-language (Rule Markup Language) [137]. The rules written by users using SWRL can be used to infer new knowledge based on prior knowledge expressed in OWL. Besides this the terms of OWL concepts can be written in HLR (Horn-like rules) [138].

According to [136] SWRL rules are written in antecedent-consequent pair. In this terminology, the antecedent is the *body* of rule, while the consequent is the *head* of the rule. The body and the head consist of a combination of one or more *atoms*. In the proposal submitted to the W3C [136] "SWRL does not support more complex logical combinations of *atoms*".

The rules SWRL perform inference about OWL individuals in terms of classes and properties. OWL can also refer explicitly OWL individuals. SWRL also supports the concepts of *same-as* and *different-from* also supports the use of several prebuilt predi-

cates (*built-in*), which expands its power of expression.

Based on these languages some initiatives are presented in order to create ontologies that can be applied in Cloud Computing. Among them we can mention a search engine called Cloudle [139], the mOSAIC ontology (Open-Source API and Platform for Multiple Clouds) [81] and an ontology-based resource management presented in [140].

In [139] the authors present a services search engine in Cloud Computing based on an Cloud ontology. This ontology contains a set of cloud concepts, the individuals of these concepts, and the relationships between these individuals. In addition to service Cloud models presented previously (IaaS, PaaS and SaaS) in Cloudle the authors add the concepts of CaaS and DaaS where CaaS is an outsourcing model for business communication and DaaS is a data storage service. The applied ontology using the OWL language and is based on the similarity of properties to find compatible service. To determine this similarity the authors present the ontology in the form of a triple containing a subject, a predicate and a datatype value.

Considering this triple an example for an individual "*Provider1*" with a predicate "*hasMemory*" and value "*4000*" would be expressed as follows: (*Provider1*, *hasMemory*, *4000*). Table 2.6 shows some examples of concepts and their individuals presented in [139]:

Table 2.6: Example of concepts and individuals used in Cloudle (adapted from [139])

Concept	**Individual**
PaaS	*Provider1*
IaaS	*Provider2*
IntelCPU	*CPU1*
AMDCPU	*CPU2*
FileSystem	*NTFS*

Based on the concepts presented, the authors create the triples to be used in the calculation of similarity. These triples may be exemplified as shown in Table 2.7:

Table 2.7: Example of triples used in Cloudle (adapted from [139])

Individual	**Property Name (Type)**	**Value**
Provider1	hasCPU (Object)	*CPU1*
Provider1	hasMemory (Datatype)	*4000*
Provider1	hasFileSystem (Object)	*NTFS*
Provider2	hasCPU (Object)	*CPU2*
Provider2	hasMemory (Datatype)	*2000*
CPU2	hasSpeed (Datatype)	*3.4*

Using the triple shown in Table 2.7 the calculation of similarity is performed by joining common objects between the providers, such as: U={(*CPU1*,*CPU2*),(*4000*,*2000*)}.

We do not show how this calculation is performed because this is not the scope of this thesis, but we point out some observations of how the ontology is applied.

For example we can observe the information given in the Tables 2.6 and 2.7 that some individuals can also be expressed as values for other individuals, as is the case of the individual "CPU2" which also serves as a value for the individual "Provider2". Therefore, there exists freedom to express values for Individuals at the same time we observe a strong dependence to express service models, as the ontology only considers the five models presented in this approach (IaaS, PaaS, SaaS, CaaS and DaaS) It is necessary one prior and thorough knowledge about these structures and not allowing the creation of new models, or by the providers or by the customers, who do not have the option to request new services. Another observation is that this ontology treats only aspects related to functional requirements.

In the approach by [82] the authors present the mOSAIC project that also considers non-functional aspects of the services on Cloud. According to the authors, the main non-functional properties used by Cloud components are: "Scalability; Autonomy; Availability, QoS, Performance, Consistency, Security and Reliability" [82]. Although these non-functional characteristics are mentioned, the authors describe some limitations of the approach. According to the authors these limitations are: 1) the intelligent discovery of services, 2) the composition of services and 3) the SLA management [82]. These items are considered limited since the ontology partially solves these challenges.

mOSAIC uses OWL language and often describes his ontology using hierarchical information. The Figures 2.6 and 2.7 shows two fragments of the mOSAIC ontology using an ontology editor [141]. The Figure 2.6 exemplifies items that composes a SLA contract according to the authors.

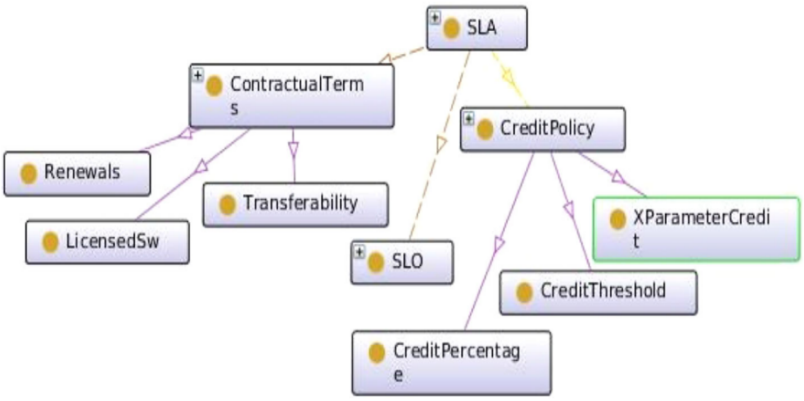


Figure 2.6: mOSAIC SLA Ontology (extracted from [142])

The Figure 2.7 shows the Service Level Objective considering quantitative and qualitative aspects.

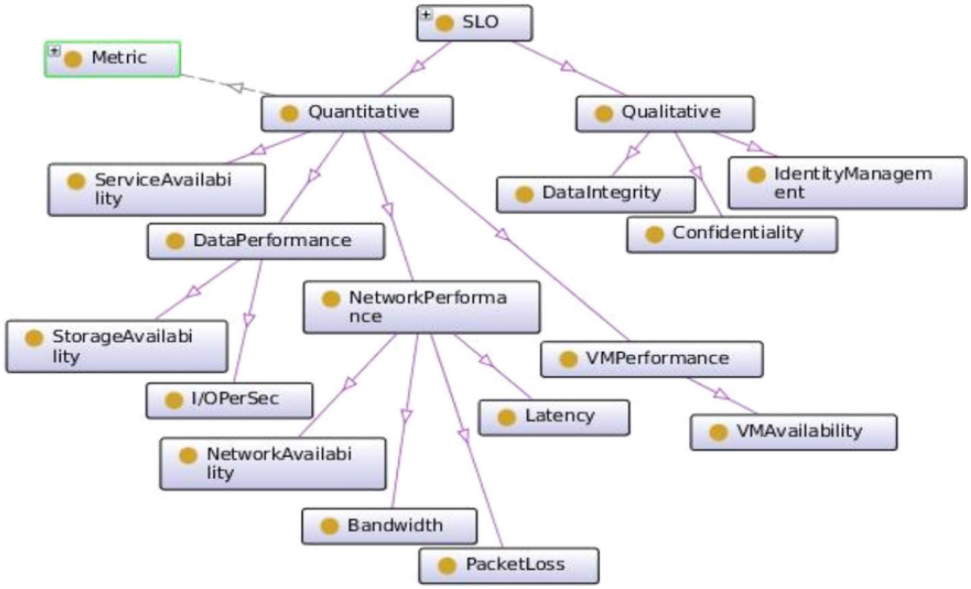


Figure 2.7: mOSAIC SLO branch (extracted from [142])

In the Figure 2.7 we can see the hierarchy of items displayed in each type of service, such as the parameters "Latency" and "Bandwitch" belonging to "NetworkPerformance" which in turn is a "Quantitative" parameter. These figures were extracted from the work presented in [142]. In this work, the authors introduced a new parameter that can quantitatively measure the idea of "Metric". It contains four properties: "*hasUnit*", "*hasValue*", "*specifiesCompareFunction*" and "*isMeasuredByThirdParty*" [142]. Besides that use SWRL language to set rules for the request of individuals. An example of these rules is presented in Figure 2.8.

1. $Request(?request) \wedge Offer(?offer)$
   $\rightarrow matchesWith(?request, ?offer)$
2. $Request(?request)$
   $\wedge specifiesMosaicServiceModel(?request, ?model)$
   $\wedge matchesWith(?request, ?offer)$
   $\rightarrow isOfMosaicServiceModelType(?offer, ?model)$
3. $Request(?request)$
   $\wedge hasAcquisitionStrategyTime(?request, ?time)$
   $\wedge Future(?time) \wedge matchesWith(?request, ?offer)$
   $\wedge isPublishedIn(?offer, ?market)$
   $\wedge hasType(?market, ?type) \rightarrow Forward(?type)$
4. $Request(?request)$
   $\wedge specifiesMosaicServiceModel(?request, ?model)$
   $\wedge IaaS(?model)$
   $\wedge hasAcquisitionStrategyType(?request, ?acquisition)$
   $\wedge Buy(?acquisition)$
   $\wedge hasUsagePattern(?request, ?pattern)$
   $\wedge Continuous(?pattern)$
   $\wedge hasDuration(?request, ?duration)$
   $\wedge OverSixMonths(?duration)$
   $\wedge matchesWith(?request, ?offer)$
   $\wedge hasPriceModel(?offer, ?price)$
   $\rightarrow PrePaid(?price)$

Figure 2.8: Subset of rules in SWRL (extracted from [142])

It is important to note that, the use of these rules in the ontology has a certain level of difficulty, being not trivial its implementation. Besides that this approach applies only to quantitative parameters and beyond to consider service models IaaS, PaaS and SaaS also adds BPaaS model (Business Process as a Service).

According to the authors, the main problem in the definition of ontology is the heterogeneity by different terms used by service providers as well as the lack of standardization identified in standards that have different terminology for services on Cloud [82].

Another approach that shows the utilization of the mOSAIC is demonstrated in [143] in this work the authors present the utilization of WS-Agreement language to represent a service request, as shown in Figure 2.9.

```
[..]
<wsag:VariableSet>
    <wsag:Variable wsag:Name="UserRequests" wsag:Metric=" xs:integer">
        <wsag:Location>
            $this/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm[@wsag:Name = 'Term1']/mosaic:JobSubmission/MaxRequests
        </wsag:Location>
    </wsag:Variable>
        <wsag:Variable wsag:Name="UserCredit" wsag:Metric="xs:integer">
            [..]
<wsag:GuaranteeTerm wsag:Name="MaxCredit">
    <wsag:ServiceLevelObjective>
        <wsag:KPITarget>
            <wsag:KPIName>MaxCreditLevel</wsag:KPIName>
            <wsag:CustomServiceLevel>UserCredit GT 0</wsag:CustomServiceLevel>
        </wsag:KPITarget>
    </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>
<wsag:GuaranteeTerm wsag:Name="MaxRequests">
[..]
```

Figure 2.9: Fragment of User SLA request in WS-Agreement (extracted from [143])

The use of WS-Agreement language proves the flexibility of ontology allowing the creation of specific solutions for each service model. Similarly this approach proves that the SLA management using the mOSAIC ontology does not offer a single general-purpose solution that can ensure the SLA management for any type of Cloud service. Since both SWRL and WS-Agreement needs an exact determination of the parameters to be used, it does not allow the addition of new services in a more simplified form.

In addition to the approaches presented in this chapter we also present the results of a survey conducted by [84]. In this work, the authors showed the results related to Service Level Agreements in Cloud Computing, among the research explored in [84] include: Cloud4SOA [144], OPTIMIS [145], RESERVOIR [146], 4CaaSt [147], CONTRAIL [148], IRMOS [149], SLA@SOI [150], ETICS [151], GEYSERS [152], VISION [153] and CumuloNimbo [154].

As a synthesis of the results shown by [84] we can extract some interesting aspects: "Service Level Agreements are increasingly becoming the key criterion for service selection. Users are now demanding agreements with clear attainable terms, services with guaranteed quality levels, offerings that meet specific legal and protection terms, accurate reporting on the service usage and runtime adaptation for evolving requirements". In fact, considering on the approaches presented, we can claim that the main problem in cloud computing is the lack of unified standards.

2.6 CHAPTER SUMMARY

The need for a management model that showcases the flexibility and control of the information contained in Service Level Agreements in Cloud Computing has been rec-

ognized by the research community. Many studies try to present different models and techniques to ensure consistent management with market needs. In this chapter, we illustrate some of the known approaches, as well as the results of surveys carried out in the scope of Cloud SLA. We focus on approaches to SLA composition focusing on languages of agreements specification and we describe some frameworks used to support its management, we also describe some metrics used in Cloud SLA and ontologies that use these metrics, highlighting the gaps shown in the different approaches. In the rest of this thesis, we propose a new ontology considering the lack of standardization in the terminology used in Cloud Computing and also to support new services, thus illustrating a possible way to improve the usefulness of the agreements in an advanced management process. We will also present a conceptual framework to support this new ontology, considering aspects of composition, monitoring and resource allocation. Finally, we will present the overview of our approach.

3

ADVANCED SLA IN CLOUD COMPUTING

Efforts to standardize the representation of Service Levels Agreements generated different languages, as seen in the previous chapter, while identifying the need for greater clarity to agreements between customers and service providers. Both technical approaches and the search for standardization advocated by international entities such as ISO/IEC for example, provide a formal representation of the main components of Cloud Computing for each type of Cloud service: However, on the other hand, these approaches do not specifically aim the generic modeling task of a SLA, limited to the specificities of each business model.

This chapter proposes the use of an ontology applied to SLA representation creating a set of resources to include adaptation and extension of business models allowing greater flexibility in SLA negotiation. The use of an expressive logical language to represent different services in Cloud allows enrich them through the application of inference mechanisms and the definition of semantic rules.

3.1 INTRODUCTION

Along with the processing capacity, generation capacity and data storage are also growing in similar proportions. This evolution can be seen by the way the man stored and passed through ideas of the times, from the paintings on the walls made by our ancestors, through the invention of writing, the role and the Gutenberg printing machine, until we reach the current medias of storage. Currently, we are used to deal with huge amounts of data and information in our lives. Government and commercial agencies devoted enormous resources to collect and store information. Systems of automation and information are becoming common in most companies. The science also has become another major producer of data.

Although methodological advances in data analysis are necessary to transform the experimental techniques in information and knowledge, the problems in the era of

Cloud Computing are not just experimental or technical but also conceptual. According to [155], conceptual modeling has been characterized in several ways. However, an important setting for this thesis is offered by [156], which presents the conceptual modeling as a process to formulate and collect conceptual knowledge about a universe of discourse and documents the results in the form of a conceptual scheme. This conceptualization process, promoted by modeling, aims to describe a domain through the entities that compose it and the relationships between those entities.

Clearly, a model that has some formal semantics is more expressive than one featuring only implicit semantics. One way to add some semantics to the models is through metadata. Metadata may be defined as "data about data" [157] and the goal of adding metadata models and data sources is to allow the user to find relevant items according to the context. The use of metadata is influenced by the structure of the data. The data can be unstructured, structured or semi-structured. Unstructured data can be of any kind and do not necessarily follow any format, rule or order. Semi-structured data have some structure, but this is not rigid. Structured data have a rigid structure, describing the objects through strongly typed attributes.

3.1.1 CHAPTER OUTLINE

Throughout this chapter we highlight the way in which resources are available in the advanced SLA management in Cloud Computing and a generic ontology is defined to support different services, and clarifies its relations, that is, regardless of the application domain. Objectives, characteristics and an ontology overview with a detailed description of its elements are presented. Initially we present the terminology and explain the concepts and then present the formal specification of the terms that will be used in the proposed generic ontology. It is noted that while many of the examples used in this chapter are related to existing areas in Cloud Computing, the proposed approach can also be used for new services, such as Internet of Things (IoT) environments. The main contribution of the Chapter is the presentation of a generic ontology that allows freedom of choice of services to customers and introduces the concept of shared responsibility.

3.2 GENERALIZED SERVICE LEVEL AGREEMENT

The interaction between a Cloud Provider (CP), offering a service, and the customers to which the service is delivered is usually based on SLAs. A SLA represents a contract between the Cloud Provider and the customers on different functional/nonfunctional properties of the provided service. However, relying on predefined SLAs might represent a limitation in the context of the customers needs, due to the richness and diversity of collected data, and the heterogeneity of the applicative scenarios. For instance, differently from traditional outsourcing scenarios in which oftentimes a bulk data collection is entirely transmitted to the provider at outsourcing time, an IoT application for pollutant monitoring, for example, might require timely transmission to the Cloud Provider of each measurement as soon as it is captured by a sensor.

The SLA between the customer and the Cloud Provider, rather than being based on a pre-defined model produced by the CP, can therefore be established by taking into consideration all specific requirements characterizing the application. This problem can however be complicated by the fact that the satisfaction of some requirements might depend on the satisfaction of other requirements, of which the subject might be unaware. For instance, to ensure a response time less than a given threshold, due to its hardware and software configurations, a CP might not be able to provide other features (e.g., the encryption of the communication with the data sources). Our solution overcomes this problem by taking into consideration dependencies among requirements in the establishment of the SLA.

Building on these observations, in this section we aim at bridging the gap between customers and Cloud Providers by proposing an approach for supporting specific service requirements in the definition of a generalized SLA.

3.2.1 GENERIC DESCRIPTION

To create a consistent SLA between the customer and the Cloud Provider we must present it into three complementary scenarios:

1. The enforceability of services by the provider: To define the services that are made available by the provider we introduce the following description: $S = \{s_1,..., s_n\}$ where S is the set of the end services provided by the provider and s_1 to s_n represents each service separately. For example: S = {Storage, Security, Availability, etc.} as in Figure 3.1.

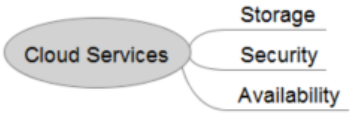


Figure 3.1: Cloud Services representation

2. The necessary attributes for each service or resource: Each service or resource may depend on other attributes that are not classified as service or even as a resource. These attributes are described as follows: $A = \{a_1,..., a_n\}$ where A is the set of attributes and a_1 to a_n represent each attribute. For example: A = {Location, Key Length, Time of Service, etc.} as in Figure 3.2.

Figure 3.2: Cloud Services and attributes representation

3. The resources used by the services: The third part relates to the resources properly used for the performance of services and assumes the following description: $R = \{r_1,..., r_n\}$ where R is the set of required features and r_1 to r_n represent each resource. For example: R = {Number of Replicas, Number of Virtual Machines, CPU usage, etc.} as in Figure 3.3.

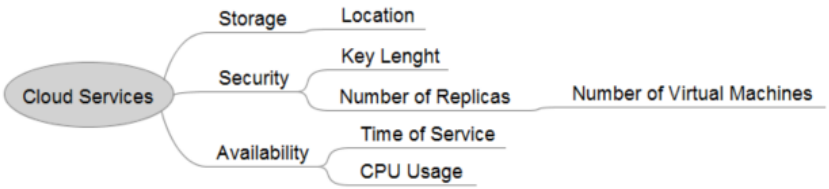


Figure 3.3: Cloud Services, attributes and resources representation

As these three scenarios may be represented in the same way and are complementary to them, we can use a single representation to incorporate the three characteristics. We called this representation "property". A property is a single description that represents each service, resource or attribute available. To define how the provider offers its properties and how the customer holds the request we use this description: $P = \{p_1,..., p_n\}$ where P represents the set of all possible properties of being set in a SLA and p_1 to p_n represents each property separately. For example: P = {Storage, Location, Security, Key Length, Number of Replicas,..., CPU usage, etc.} as in Figure 3.4.

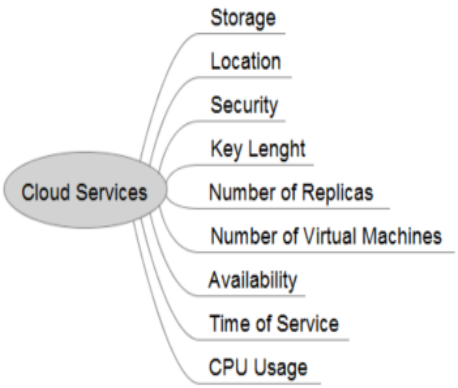


Figure 3.4: Properties representation

Each property is then meant as a generalization of different scenarios in Cloud environments, where we describe the integration of different services highlighting its special features and a simpler way for the customer.

With the use of properties (P) the customer can order end services (S) with attributes (A) of some services or specific resources (R) according to the user's needs. Each property has its value assigned by the provider according to the conditions in which they are available. On the other hand each property has the value requested by the customer according to their needs. This approach allows the establishment of a model that considers aspects recognized by both parties through the service conditions and the customer requests.

Through a request, a customer can order any property it deems relevant to the service agreement, keeping track of what it is negotiated. And the provider can provide all the properties that it considers to be necessary to settle the agreement. Then the service requirements are formulated as Boolean formulas over conditions defined on properties that represent (functional or non-functional) properties characterizing the cloud services and are taken from a common/shared ontology.

3.2.2 FORMAL DEFINITION

The following properties: *srv_loc*, *resp_time*, *encr*, *sec_audit*, and *access_log* are examples of properties (modeling the physical location of a server, the response time of the service, the encryption algorithm adopted by the provider, the auditing frequency, and whether accesses are logged respectively) that can be used to define conditions on the required service. Let P be the set of properties. Each property $p \in P$ takes values from its domain *dom(p)*. For instance, *dom(srv_loc)*={USA, France}, *dom(resp_time)*={10, 20, 50}, *dom(encr)*={AES, DES, no}, *dom(sec_audit)*={weekly, monthly, no}, *dom(access_log)*={yes, no}, etc. A condition defined over a property restricts the values that the property can assume in the provision of the service. A condition is formally defined as follows.

Definition 3.1 *(Condition). Given a set P of properties, a property $p \in P$ with domain dom(p), and a value $val \in dom(p)$, a condition c over p is a term of the form c : (p op val), with $op \in \{=, \neq, <, \leqslant, >, \geqslant\}$ a comparison operator.*

The Figure 3.5 illustrates a set of conditions for an example in Cloud environment. For instance, c_5: (*resp_time* < 10) and c_{10}: (*srv_loc* = USA) model two conditions demanding that the service exhibits a response time less than 10 milliseconds (c_5) and uses a storage server being located in USA (c_{10}). Service requirements can be composed of different conditions over different properties. More precisely, by interpreting each condition c as a Boolean variable, a service requirement can be naturally expressed as a Boolean formula over such variables.

c_1: (*proc_num* $\geqslant$ 2)	c_5: (*resp_time* < 10)	c_9: (*backup* = daily)
c_2: (*encr* = AES)	c_6: (*sec_audit* = weekly)	c_{10}: (*srv_loc* = USA)
c_3: (*encr* = no)	c_7: (*access_log* = yes)	c_{11}: (*storage* < 100TB)
c_4: (*req_rate* < 1/min)	c_8: (*backup* = no)	c_{12}: (*storage* $\geqslant$ 100TB)

Figure 3.5: Example of a set of conditions

For simplicity but without loss of generality, we assume requirements to be in disjunctive normal form (DNF). A service requirement is defined as follows.

Definition 3.2 *(Service requirement). Given a set C = {c_1,..., c_n} of conditions over a set P of properties, a service requirement R over C is a formula of the form* $\vee_{i=1}^{m}(\wedge_{j=1}^{k(i)} c_{i_j})$*, with k(i) the number of conditions of the* i^{th} *clause, and* $c_{i_j} \in C$.

For instance, considering the conditions in Figure 3.5, the service requirement can be formulated as $R : (c_5 \wedge c_{10}) \vee (c_5 \wedge c_2) \vee (c_5 \wedge c_6)$. Intuitively, R states that to satisfy the requirements, the cloud service should exhibit a response time less than 10 milliseconds (c_5) and use a storage server located in USA (c_{10}), or exhibit a response time less than 10 milliseconds (c_5) and use AES for encryption (c_2), or exhibit a response time less than 10 milliseconds (c_5) and weekly execute security auditing (c_6).

A SLA should also consider possible dependencies related to the conditions included in R. Dependencies capture generic relationships among properties, implying that the enforcement of a condition over a property *depends on* the enforcement of another condition over another property. For instance, with reference to our running example, properties *resp_time* and *req_rate* are linked by a dependency. If R includes a condition over *resp_time*, then a valid SLA should also include a condition over *req_rate*.

While property dependencies can be considered to always hold (e.g., the responsiveness of a service is always impacted by rate of requests it receives), the specific conditions holding for the properties involved in a dependency can vary depending on the CP (e.g., a CP with a set of servers running in parallel might accept more requests per time unit than another CP with a single server). Upon receiving a service requirement R from the customer, the CP must verify whether the conditions in R imply other conditions due to the presence of dependencies. Note that dependencies can model both incompatibilities among conditions (i.e., enforcing a condition over a_i does not allow to enforce another condition over a_j) and implications among them (i.e., enforcing a condition over a_i requires the enforcement of another condition over a_j). Building on our interpretation of conditions as Boolean variables, a condition dependency, meaning a property dependency instantiated with conditions over its attributes, is defined as follows.

Definition 3.3 *(Condition dependency). Given a set C = {c_1,..., c_n} of conditions over a set P, a condition dependency d over C is defined as* $d : c_h \rightsquigarrow (\vee_{i=1}^{m}(\wedge_{j=1}^{k(i)} c_{i_j}))$*, with k(i) the number of conditions of the* i^{th} *clause, and* c_h,$c_{i_j} \in C$.

A dependency $d : c_h \rightsquigarrow (\vee_{i=1}^{m}(\wedge_{j=1}^{k(i)} c_{i_j}))$ can be interpreted as a material implication: if condition c_h is satisfied, then also $\vee_{i=1}^{m}(\wedge_{j=1}^{k(i)} c_{i_j})$ must be satisfied.

The Figure 3.6 illustrates five condition dependencies defined over the set of conditions shown in Figure 3.5.

$$d_1\text{: } (serv\_loc = \text{USA}) \rightsquigarrow (storage < 100\text{TB})$$
$$d_2\text{: } (resp\_time < 10) \rightsquigarrow (backup = \text{no}) \wedge (req\_rate < 1/\text{min}) \wedge (encr = \text{no})$$
$$d_3\text{: } (encr = \text{AES}) \rightsquigarrow (proc\_num \geqslant 2)$$
$$d_4\text{: } (backup = \text{daily}) \rightsquigarrow (storage \geqslant 100\text{TB})$$
$$d_5\text{: } (sec\_audit = \text{weekly}) \rightsquigarrow (access\_log = \text{yes})$$

Figure 3.6: Example of conditions dependencies

The dependency d_1 states that providing a server located in USA implies a maximum storage capacity of 100TB. Dependency d_2 states that a response time less than 10ms is incompatible with the execution of backups and of encryption operations (incompatibilities), and imposes a maximum rate of requests of 1 per minute. Dependency d_3 states that to provide AES encryption, the server must have at least two processors. Dependency d_4 states that a daily backup requires a storage capacity greater or equal to 100TB. Dependency d_5 states that to ensure a weekly auditing process, accesses should be logged. Conditions in a dependency can involve properties under the control of either the CP (e.g., *storage* < 100TB in d_1) or the customers (e.g., *req_rate* < 1/min in d_2, being the request rate dependent on the operations of the customers).

In our scenario, given a set $C = \{c_1,..., c_n\}$ of conditions, a SLA is naturally represented as a set $\{c_1,..., c_k\} \subseteq C$ of conditions, whose enforcement is guaranteed by the CP in the service provision. Note that a SLA should include at most one condition over each property $p \in P$ (as otherwise they would be in conflict).

We refer to a set of conditions satisfying this property as well-formed, as follows.

Definition 3.4 *(Well-formed set of conditions). Given a set C of conditions over a set P of properties, C is said to be well-formed iff $\forall c \in C, \forall p \in P, |C_p| \leqslant 1$, with $C_p \subseteq C$ the conditions over property p.*

For instance, with reference to the conditions in Figure 3.5, the set $\{c_2, c_{11}, c_{12}\}$ is not well-formed as c_{11} and c_{12} are defined over the same property *storage*.

Given a set $C = \{c_1,..., c_n\}$ of conditions, a service requirement R over C, and a set $D = \{d_1,..., d_l\}$ of dependencies over C, our goal is to find a subset of conditions in C that forms a valid SLA, meaning that the SLA is well-formed and satisfies both R and D. Following our logical modeling where conditions are interpreted as Boolean variables, service requirements as Boolean formulas, and dependencies as material implications, we introduce an assignment function $f : C \to \{0,1\}$ assigning to each condition in C a value from the set $\{0,1\}$. With a slight abuse of notation, we use f to denote also the list of values assigned by f to the conditions in C. Therefore, given a requirement R over C, *f(R)* will denote the result of the evaluation of R with respect to the values in f.

Since R must be satisfied when assigning values to C to compute a SLA, f is a correct assignment w.r.t. a requirement R iff $f(R) = 1$. Similarly, a dependency $c_h \rightsquigarrow P(c_i,..., c_j)$ is satisfied provided that, if $f(c_h) = 1$, then $f(P(c_i,..., c_j)) = 1$. Therefore, f is a correct assignment w.r.t. a set D of dependencies iff $f(d) = 1, \forall d \in D$.

A SLA is then interpreted as a complete value assignment over the conditions in C, where the conditions included in the SLA are those assigned value 1 by f. Based on the presented formal concepts began the description of the generic proposal ontology.

3.3 OBJECTIVE OF ONTOLOGY

According to [158], one of the main objectives of development ontology is to provide interfaces between the human, that understands the concepts, and the machine, which provides accurate, consistent and unambiguous representations of the models. The specific objectives of the construction and application of generic ontology proposal are:

1. To represent a model for the description of Service Levels Agreements with a logical language, with inference capability and usage rules.
2. Represent the adaptation of the existing models, described in Chapter 2, with annotations semantics in order to overcome the limitations described. This implies explicitly represent knowledge that is implicit in the various services by associating semantics to the components and variables.
3. Allow the creation of agreements to which the variables are defined from its meaning and not just syntactically.
4. Allow the creation of a repository of templates that can be searched semantically, either directly through the web, or through a database capable of inference.
5. To promote the reuse of existing agreements, by the possibility of automatic or semi-automatic composition of complex models from simpler existing models.

These objectives determine the fundamental characteristics to our ontology, other relevant aspects considered in this work are: the representation of classes as property values, the modeling of the whole-part type relationships, the definition of n-ary relationships and representation of values through set of values.

3.4 CHARACTERISTICS OF THE ONTOLOGY

The language chosen for the representation of service agreements was XML, this language is based on Description Logics, being expressive enough to represent the components and services properties, while allowing the use of inference mechanisms and the definition of semantic rules. A description of XML can be found in [134]. Considering the applicability of ontology for the description and simulation of service agreements in Cloud Computing, the existence and availability of various services and various

tools that work with the language, the project of ontology sought to reflect a structure of a general service agreement and dynamic to facilitate the use of existing designs.

The simplification and integration of the proposal ontology, with ontology defined for various services, allows the create of tools to integrate existing templates, so that the user work has a high level of abstraction.

Another important issue is associated with the service request validations. Since it is based on XML, the validation is merely syntactic. Through the use of Schemes, the request can be validated for syntax errors and for the adherence to service conditions determined by the CP. Semantic questions can be effectively treated (e.g. prevent a property "response time" contains a value like "USA") or are left to the implementation phase (e.g. compliance in the use of different measurement units for the variables, as "second" and "milliseconds").

Since it is based on XML, the ontology can specify elements that can be used to represent service agreements formally, unambiguous, human readable and processable by machines. Metadata of each agreement can be stored together with the own agreement. The controls of the properties may be represented independently of a specific implementation. The ontology can be used to represent, store and share agreements, increasing their availability and facilitating the use and validation by customers.

Basically the elements in our approach are used to describe properties, values and units. These elements are grouped to represent service conditions and service requests.

3.4.1 SERVICE CONDITIONS IN XML SCHEMA

As seen in subsection 3.2.2 a condition c over p is a term of the form $c : (p\ op\ val)$, with $op \in \{=, \neq, <, \leqslant, >, \geqslant\}$ a comparison operator. This definition should be represented in our XML Schema. To enable the desired generalization we must also introduce the definition of dependency in the same schema.

The Figure 3.7 shows a diagram of the elements that make up a property to a service condition. The following will be described the main elements that composes this Schema.

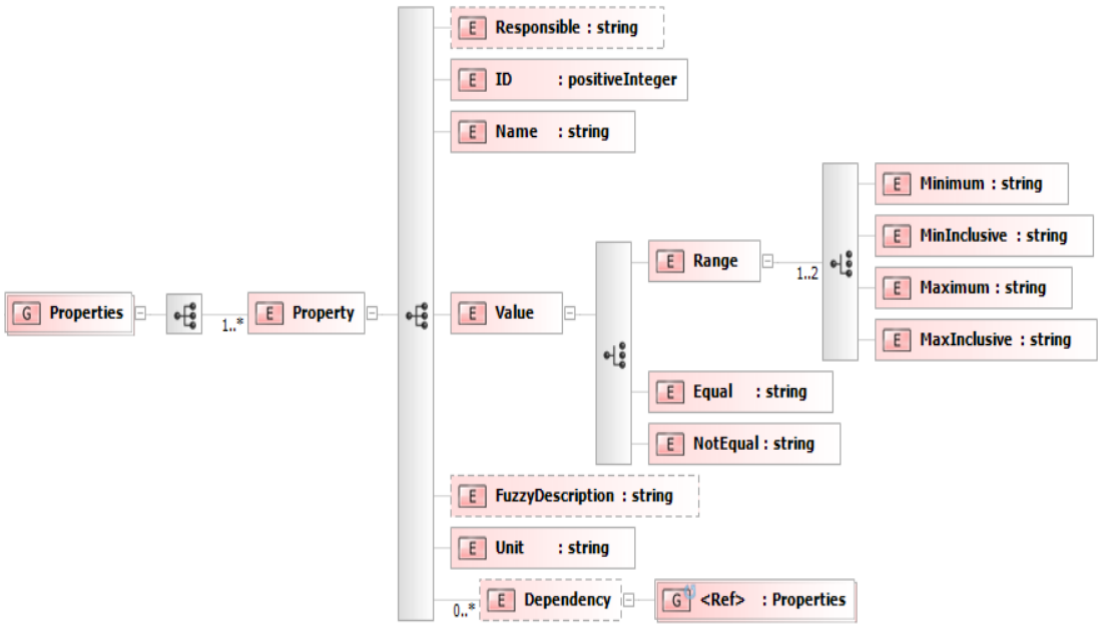


Figure 3.7: Properties representation in service conditions XML Schema

Since a property set is represented as $P = \{p_1,..., p_n\}$ then we define a Group of properties containing one or many properties Elements. Each element property contains his "name", "value" and "unit." We added the concept of "dependency" that allows a property to be dependent on other properties. In addition to a "identifier" element and a "responsible" element for the property, which will be detailed later. We also add the element "FuzzyDescription" which will be explained in the Chapter 6.

The element "name" is actually the name of the property and is used to identify the different services, attributes and resources sold by CP.

According to the Definition 3.1 the element "value" takes on the different possibilities according to the property in question, i.e., *Range* $(<, \leqslant, >, \geqslant)$ to properties that have values expressed in minimum, minimum inclusive, maximum or maximum inclusive (or two of these situations), *Equal* $(=)$ to properties that express exact values and *NotEqual* $(\neq)$ to determine values not required or not supported by the provider.

The element "unit" to determine the unit of measurement or identification of a property as second, millisecond, country, etc.

The element "dependency" that enables mapping a property that depends on the value of another property or set of properties.

The Figure 3.8 shows the XML schema representation of a service condition, and as a set of properties is arranged, besides the elements already described, we add the "provider" attribute to identify the service provider and a "ServiceID" attribute as identifier.

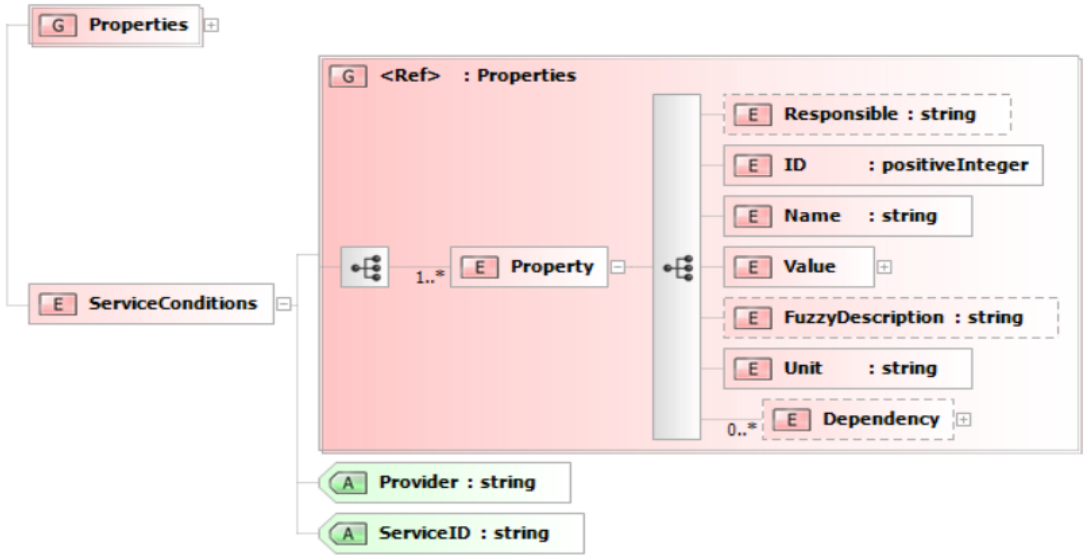


Figure 3.8: Service Conditions representation in XML Schema

The Figure 3.9 shows the code to the XML Schema used in the creation of a service condition, note that the "responsible" element have three possible values: Provider, Customer and Both Parts. Since the ontology presented is generic, it can be used to describe any type of service agreement, including Master Service Agreements and Contracts. Then we can show who is responsible for certain property.

This approach facilitates the registration and storage of templates for services of any kind, both Cloud and for other environments.

The dependencies are used in the ontology with the basic purpose of inferring knowledge that is implicit in existing models, converted to the ontology; This is done by associating the properties defined in the ontology, from the details obtained from existing template (e.g., from the name of the property), also considering the already established knowledge (e.g., resource values that can be associated with end services).

The dependencies can also be used to validate semantically the agreement regarding their consistency and completeness; a validation example is to check whether the composition of the properties is in line with the service hierarchy.

```
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:group name="Properties">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="Property">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="Responsible">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="Provider" />
                  <xs:enumeration value="Customer" />
                  <xs:enumeration value="Both Parts" />
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element minOccurs="1" maxOccurs="1" name="ID" type="xs:positiveInteger" />
            <xs:element minOccurs="1" maxOccurs="1" name="Name" type="xs:string" />
            <xs:element minOccurs="1" maxOccurs="1" name="Value">
              <xs:complexType>
                <xs:choice minOccurs="1" maxOccurs="1">
                  <xs:element name="Range">
                    <xs:complexType>
                      <xs:choice minOccurs="1" maxOccurs="2">
                        <xs:element minOccurs="1" maxOccurs="1" name="Minimum" type="xs:string" />
                        <xs:element minOccurs="1" maxOccurs="1" name="MinInclusive" type="xs:string" />
                        <xs:element minOccurs="1" maxOccurs="1" name="Maximum" type="xs:string" />
                        <xs:element minOccurs="1" maxOccurs="1" name="MaxInclusive" type="xs:string" />
                      </xs:choice>
                    </xs:complexType>
                  </xs:element>
                  <xs:element minOccurs="1" maxOccurs="1" name="Equal" type="xs:string" />
                  <xs:element minOccurs="1" maxOccurs="1" name="NotEqual" type="xs:string" />
                </xs:choice>
              </xs:complexType>
            </xs:element>
            <xs:element minOccurs="0" maxOccurs="1" name="FuzzyDescription" type="xs:string" />
            <xs:element minOccurs="1" maxOccurs="1" name="Unit" type="xs:string" />
            <xs:element minOccurs="0" maxOccurs="unbounded" name="Dependency">
              <xs:complexType>
                <xs:group ref="Properties" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:group>
  <xs:element name="ServiceConditions">
    <xs:complexType>
      <xs:group minOccurs="1" maxOccurs="1" ref="Properties" />
      <xs:attribute name="Provider" type="xs:string" use="required" />
      <xs:attribute name="ServiceID" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 3.9: Service Conditions XML Schema

Since XML allows the sharing of information independent of the type of application used to view, it is necessary to identify the structure of the XML document. That is, an XML Schema allows the definition of the structure, content and semantics of a document. In the XML schema shown in figure 3.9 we define a group of elements

called Properties (<xs:group name="Properties">) this allows elements to be specified within other elements and obeys an order or specific choice through connectors.

Then we use the connector "sequence" that is specified for all elements of each property appear in the correct order. The first statement in the sequence creates an element called "Property", which may appear at least one (1) time and no maximum limit. Within each "Property" element we have a "complexType" with a new connector "sequence". The first element of this sequence is the "responsible" which is optional and when used presents one of three restricted values (Provider, Customer or Both Parts). Following the sequence we have the elements "ID" as a type positive integer, element "Name" as a type string and the element "Value".

The element "Value" has another complex type with a connector "choice" that is mandatory. If the choice is the element "Range" we have another complex type with another connector "choice". In this case the choice can be for only one alternative and at most two alternatives. All elements in this code fragment are of type string and the alternatives are "Minimum", "minInclusive", "Maximum" and "maxInclusive". If the choice is not the element "Range" then the choice should be between the other two elements of type string: "Equal" and "NotEqual". Following the sequence we have the element "FuzzyDescription" of type string and is optional, the element "Unit" of type string and mandatory and the element "Dependency" which is optional and makes reference to the group of elements "Properties".

The element "ServiceConditions" has the group of elements "Properties" as described, plus two attributes of string type and binding. The first is the attribute "Provider" and second attribute is the "ServiceID".

This XML Schema can be used as follows: for example if the provider Amazon want to register an audit service and, for this, access should be logged, as shown in dependency: d_5: (*sec_audit* = weekly) $\rightsquigarrow$ (*access_log* = yes) an XML file in accordance with the proposed Schema would be presented as in Figure 3.10.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceConditions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Users\Gerson\Desktop\Works\ArquivosXML\ServiceConditions.xsd"
Provider="Amazon" ServiceID="15223">
  <Property>
    <Responsible>Provider</Responsible>
    <ID>2995563595</ID>
    <Name>Auditing Frequency</Name>
    <Value>
      <Equal>weekly</Equal>
    </Value>
    <Unit>period</Unit>
    <Dependency>
      <Property>
        <Responsible>Customer</Responsible>
        <ID>3635883089</ID>
        <Name>Access Log</Name>
        <Value>
          <Equal>yes</Equal>
        </Value>
        <Unit>response</Unit>
      </Property>
    </Dependency>
  </Property>
</ServiceConditions>
```

Figure 3.10: XML file to a Service Condition

In this way, a service provider can register and post their service plans more simply adding all the properties related to each service/plan, including contractual properties such as price, time of contract, among others.

Complementing the example of service condition in Figure 3.10 we can add more information to the service with the following properties and dependencies: C: $\{d_1$: $(serv\_loc = \text{USA}) \rightsquigarrow (storage < 100\text{TB})$, d_2: $(resp\_time < 10) \rightsquigarrow (backup = \text{no}) \wedge (req\_rate < 1/\text{min}) \wedge (encr = \text{no})$, d_3: $(sec\_audit = \text{weekly}) \rightsquigarrow (access\_log = \text{yes})$, c_1: $(price = 100)\}$ as shown in Figure 3.11.

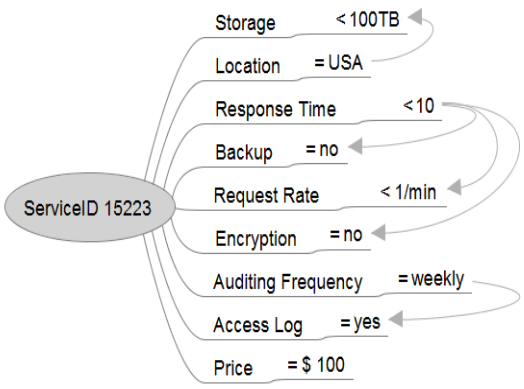


Figure 3.11: Example of a service condition in the ontology

The Figure 3.11 illustrates the dependencies between the properties as: Service Location in USA depends on a smaller storage capacity than 100 TB; A response time smaller than 10 ms depends on a request rate lower than 1 per minute and that does not have backup or encryption and a weekly audit frequency since access logging is performed.

These dependencies are optionally indicated by the provider to make strong and transparent the agreement between the parties, in addition to determining the responsible for each property at the time of service registration. Dependencies are important for the CP to determine how the services can be composed. At the same time they are not used by customers in their requests as they are not necessary at this time.

3.4.2 SERVICE REQUEST IN XML SCHEMA

As shown in subsection 3.2.2 a service requirement is also based on properties and their values, thus proving that the proposal ontology is common for all parties involved in the elaboration of service agreements.

As one of the main objectives pursued in this thesis is to give greater freedom to service customers in the preparation of agreements we should describe a specific XML Schema for service requests, which contains elements that allow this freedom.

The Figure 3.12 shows how XML elements are arranged in a service request.

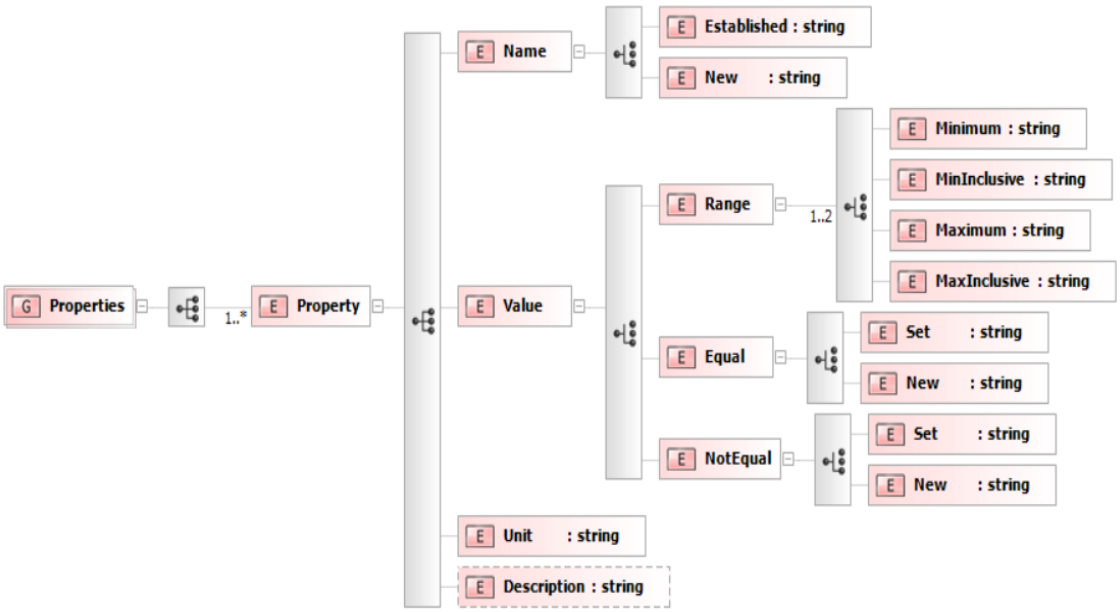


Figure 3.12: Properties representation in a service request XML Schema

According to the Definition 3.2 a Service Request can be represented as: $R : (p_1 \wedge p_n)$. To simplify the procedure for service request by the customer we can infer that a request can also be represented as follows: R: (p_1,..., p_n).

In this way, we can use the same property record format used in the service conditions, adding the peculiarities inherent to the desired freedom.

In the Service Request of the XML Schema, the elements "Name", "Value" and "Unit" has the same function presented in XML Schema Service Conditions.

The element "Name", that in the service condition was simply a type string, now shows two other elements: "Established" and "New". The element "Established" has the properties already registered and recognized by service providers. This approach should be used by a Broker service this element takes the list of all properties available for all providers. If a service desired by the customer is not yet established, it can request through the element "New".

The same procedure occurs with the elements "Equal" and "NotEqual". If the value is already evaluated and recorded by the provider this amount is listed in the element "set" if the value still does not exist may be asked by the element "New" of the absolute values.

Once a property is new, the customer has the option to describe the desired characteristics in the element "Description" that will be later used by the properties repository.

The Figure 3.13 shows the XML schema representation of a Service Request, besides the elements already described, we add the attribute "Customer" to identify the client requesting the service. This then enables the elaboration of the agreement between the parties.

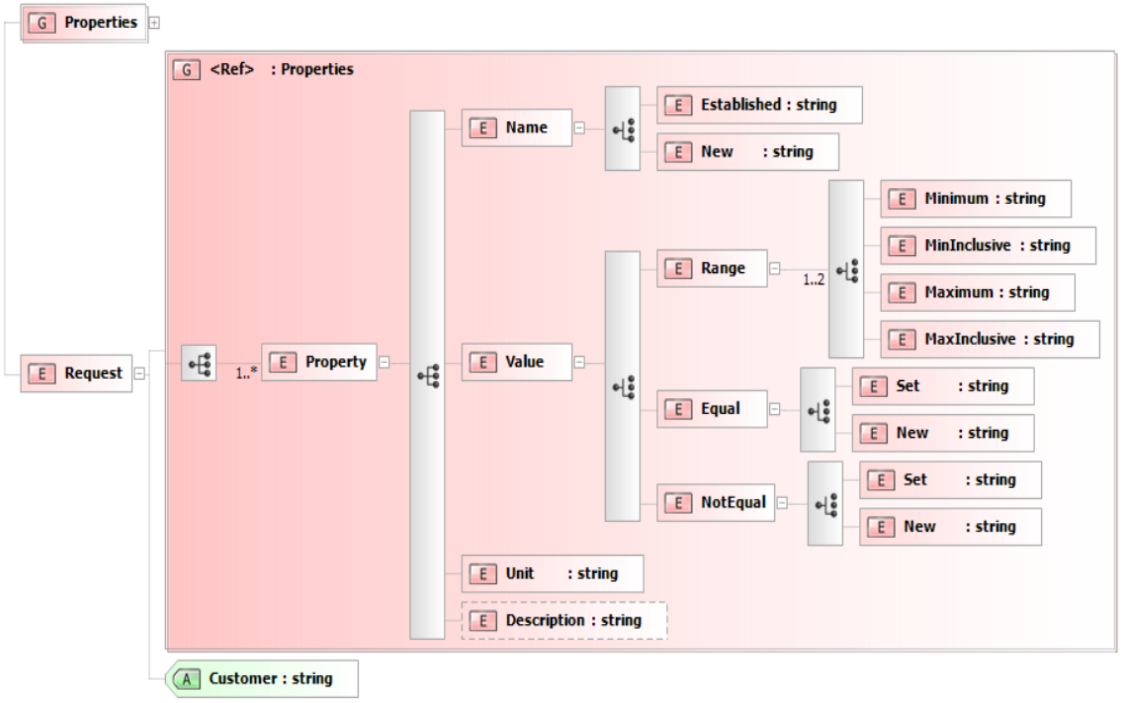


Figure 3.13: Properties representation in a service request XML Schema

The use of the element "Established" allows the customer to search for services already registered by the providers according to the ontology proposed as well as allowing the insertion of new properties according to the real needs of the customer. The same also happens to property values already listed by the provider. If the customer does not locate the desired value it has the possibility to ask the provider.

The Figures 3.14 and 3.15 shows two XML schema fragments used for the composition of a service request. The Figure 3.14 exemplifies a list of established properties by providers.

```
<xs:element name="Established">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Location" />
      <xs:enumeration value="Response Time" />
      <xs:enumeration value="Access Log" />
      <xs:enumeration value="Backup" />
      <xs:enumeration value="Storage" />
      <xs:enumeration value="Time of Contract" />
      <xs:enumeration value="Bandwidth" />
      <xs:enumeration value="Latency" />
      <xs:enumeration value="Request rate" />
      <xs:enumeration value="Number of Servers" />
      <xs:enumeration value="Throughput" />
      <xs:enumeration value="Processing Time" />
      <xs:enumeration value="Encryption" />
      <xs:enumeration value="Price" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 3.14: Fragment of XML Schema for a list of established properties

The Figure 3.15 shows an example of values assumed in the properties, here we can also take the choice between some previously setted value or the insertion of a new value.

```
<xs:element name="Equal">
  <xs:complexType>
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="Set">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="USA" />
            <xs:enumeration value="UK" />
            <xs:enumeration value="Italy" />
            <xs:enumeration value="yes" />
            <xs:enumeration value="no" />
            <xs:enumeration value="weekly" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element minOccurs="1" maxOccurs="1" name="New" type="xs:string" />
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Figure 3.15: Fragment of XML Schema for a set of values for the element "Equal"

In the Figure 3.15 the element "Equal" has a complex type with a connector "choice", this connector allows the choice between an element "Set" and an element "New". If the

choice is the element "Set" the customer can choose between the values presented, if the choice is the element "New" the customers can customize the solicitation according to their needs.

To illustrate the use of this XML Schema we demonstrate how a service request can be submitted by a customer (Figure 3.16).

```
<?xml version="1.0" encoding="utf-8"?>
<Request xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\Gerson\Documents\Works\ArquivosXML\Request.xsd"
Customer="ABC">
  <Property>
    <Name>
      <Established>Location</Established>
    </Name>
    <Value>
      <Equal>
        <New>France</New>
      </Equal>
    </Value>
    <Unit>country</Unit>
  </Property>
  <Property>
    <Name>
      <New>Video Streaming</New>
    </Name>
    <Value>
      <Equal>
        <New>720p</New>
      </Equal>
    </Value>
    <Unit>resolution</Unit>
  <Description>Video streaming with 720p resolution (also known as HD) is a
progressive HDTV signal format with 720 horizontal lines and an aspect ratio (AR)
of 16:9 (1.78:1)</Description>
  </Property>
</Request>
```

Figure 3.16: Properties representation in a XML file of service request

In this example, the customer ABC needs a video streaming service with 720p of resolution on servers located in France, so he order a service with two properties. The property "Location" (p_1) that is established by providers and a new property called "Video Streaming" (p_2). The property "Location" do not have the country France on her list of values, then the customer can request a new value to this information. For the new property "Video Streaming" in addition to the procedure to request the new property and its value the customer can also give a generic description for the service thus aiding the understanding of its needs by the provider.

Based on pre-defined conditions of service and the customer requirement we can then define all service concepts at a higher level needed for use in a real agreement. Thus determining the creation of a SLA that can be generalized to suit any customization required by the customer and ensuring the effective implementation of the concept of XaaS (everything as a service).

3.5 ADVANCED ISSUES IN A GENERIC SLA

One of the initial tasks carried out in the research for this study was the identification of requirements that must be met to establish a Service Level Agreement. As seen until the moment these requirements were divided into three groups: service conditions (given by the CP), service request (given by customer) and valid SLA (goal of our approach).

The ontology used for a valid SLA (vSLA) is defined through the integration of the two ontologies presented previously (ServiceConditions and Request) as shown in Figure 3.17.

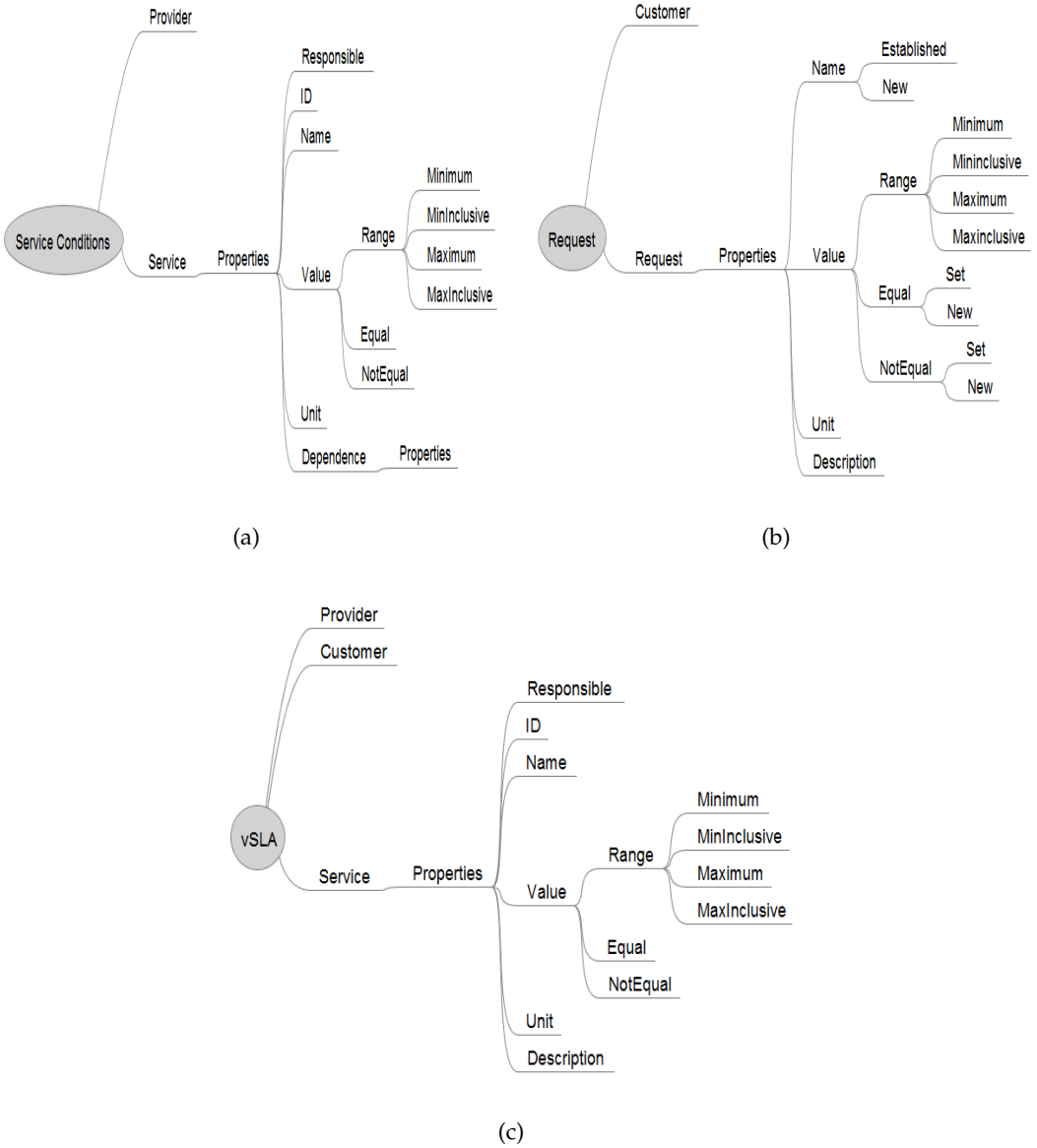


Figure 3.17: Ontology proposed in three complementary groups: Service Conditions (a), Request (b) and vSLA (c)

This approach meets one of the ontology proposed objectives which is to allow the creation of agreements to which the variables are defined from its meaning and not just syntactically.

Furthermore we can use patterns already established in the market, as in [4] to establish a repository of description of properties to be used by providers in the enforcement of their services and to facilitate the search for services by customers.

From the customer's point of view a service request R to determine a vSLA should consider two important aspects:

1) Knowledge and understanding of the required properties: Once the goal is to establish a high level of freedom for the customer we must use the definition of property in a way that both parties can understand their meaning. For this we use the property descriptions to determine a common sense for each property. As in Table 3.1.

Table 3.1: An example of property descriptions

Property Name	**Description**
Authentication	Specifies the available authentication mechanisms supported by the CP on its offered cloud services.
Availability	The property of being accessible and usable upon demand by an authorized entity.
Response Time	Time interval between a cloud service customer initiated event (stimulus) and a cloud service provider initiated event in response to that stimulus.
Service Reliability	Describes the ability of the cloud service to perform its function correctly and without failure over some specified period.

These descriptions were taken from [4]. However, our approach allows some generic description, such as response time to be edited and present a more specialized description, e.g., determining which stimulus starts the event.

The response time is a measure related to the overall performance of the system and not of either component. The response time is defined as the difference between the time that the customer has initiated a request or question and the time that the system presented to the customer her answer. For example, the time interval between the request for an account balance in a bank terminal and the presentation in the video response (the account balance). Or we can determine that the response time is the time the CP has met the request to perform a particular task. Such as performing mathematical calculations and presentation of results.

So we may have different properties "Response Time" characterized by their description and with different names such as: "Response Time for bank account balance" and "Response Time for mathematical calculations" each one with its respective description.

2) Determining the values of the required properties: Freedom in the composition of a SLA also must allow the customer to request new values for the required properties. For this, the values of each property are shown as in Table 3.2.

Table 3.2: An example of property values

Established Property	**Set of Absolute Values**
Location	=USA =Italy =Brazil ≠North Korea
Encryption	=no =Triple DES =RSA =Blowfish =Twofish =AES
Auditing Frequency	=weekly =monthly

This common nomenclature allows the creation of a repository of templates that can be searched semantically, either directly through the web, either through a database capable of inference. Besides to represent a model for the description of Service Levels Agreements with a logical language, with inference capability and usage rules. The vSLA is then interpreted as an assignment of values on properties P, where the conditions included in the SLA are those value 1 assigned by the service conditions C.

3.5.1 DETERMINING A VALID SLA

Our problem of determining vSLA given an service request R and a set D of dependencies over a set C of conditions can therefore be interpreted as finding a value assignment f being correct w.r.t. R and D, and such that the set of conditions assigned value 1 by f be well-formed, as formally defined as follows.

Problem 3.1 *(vSLA). Given a set $C = \{c_1,..., c_n\}$ a service request R over C, and a set $D = \{d_1,..., d_l\}$ of dependencies over C, determine (if it exists) a value assignment f to the conditions in C s.t.:*

1. *$f(R) = 1$ (requirement satisfaction);*
2. *$f(d) = 1, \forall d \in D$ (dependence satisfaction);*
3. *$\{c_i \in C : f(c_i) = 1\}$ is well-formed according to Definition 3.4 (conflict satisfaction).*

To demonstrate the generality of our approach, we refer our examples to a municipality owning a sensor network to measure air pollutants in its area. Each sensor measures specific pollutants at regular time intervals, and the recorded measurements

need to be collected and analyzed to set appropriate countermeasures (e.g., restricting vehicles circulation) when needed. Since sensors have limited storage capacity, the municipality aims at relying on an external CP to store and manage the collected data.

Outsourced measurements need to be retrieved by the municipality health office whenever needed and, since timely retrieval is a critical factor for fast air quality analysis, the municipality wishes the CP to ensure a maximum response time to requests. Since the outsourced measurements are considered sensitive information (the existence of correlations between high levels of air pollutants in a certain area and respiratory diseases of citizens living nearby is well known), the municipality wishes also that data be either: *i*) physically stored in a chosen trusted country, or *ii*) physically stored at a CP audited for security every week; or *iii*) encrypted by the CP (since measurements cannot be encrypted before storage, we assume for the sake of the example the municipality to choose a CP among those considered trusted for accessing plain text data, hence confidentiality is required against intruders/unauthorized third parties). These requirements set the parameters of the service that the CP provides to the municipality and are part of the SLA between the CP and the municipality (hereinafter, the *customer* of the service).

The SLA establishment starts with the communication to the CP of the requirements imposing arbitrary conditions on functional/nonfunctional properties to be satisfied in the service provision. For instance, in our running example the municipality application requirements comprise a condition restricting the *response time* to a maximum value. Upon receiving the request, the CP can check whether it can satisfy them and, if this is the case, the conditions in the request are inserted into a SLA on which both the CP and the requesting party can agree. If the CP cannot satisfy the given conditions, a SLA cannot be established.

The process of checking whether the request can be fulfilled can be complicated by the possibility that the enforcement of a condition might be possible only provided that other conditions be also enforced. For instance, to ensure a response time less than a given threshold, a CP might be able to accept only a limited *number of requests* per time unit. This is due to the fact that the response time of a system is not an isolated property: on the contrary, it is linked to other properties by a *dependence* (such as the rate of requests, which have a clear impact on the responsiveness of a system).

We note that dependencies cannot be assumed to be known by IoT infrastructure authorities, and taken into account before formulating their application requirements. In fact, they can be provider-dependent, meaning that some dependencies might hold for a given CP while not holding for other ones. While dependencies must therefore be transparent for the customers, each CP knows the specific dependencies that hold for its services. To build a vSLA starting from application requirements, the CP must then check such requirements against possible dependencies.

It is easy to see that the consideration of both requests and dependencies in the establishment of a SLA can result in different outcomes: *i*) the service conditions can be satisfied as they are (i.e., no dependency is involved) and can be put into a vSLA; *ii*) the conditions cannot be satisfied (e.g., the CP does not have resources to fulfill them), and a vSLA cannot be created; and *iii*) some conditions involve dependencies

that require the enforcement of further conditions, which then also need to be inserted into a vSLA.

Our problem can be naturally represented through a mixed and colored hypergraph representing the input of Problem 3.1. And then we can present a solution based on a translation of the problem as a Constraint Satisfaction Problem (CSP) [159]. Such mixed hypergraph $G\ (V, E, E^u)$, with V the set of vertices and E (E^u, resp.) the set of directed (undirected, resp.) hyperedges, is defined as follows.

- Each condition appearing in R and in the set $D = \{d_1,..., d_l\}$ of dependencies holding for the CP, as well as the request R, correspond to a vertex $v \in V$;
- Each dependency $d : c_h \rightsquigarrow P(c_i,..., c_j)$ where $P(c_i,..., c_j)$ is composed of m OR-ed terms is translated into m directed hyperedges in E where the i^{th} hyperedge connects c_h to all conditions of the i^{th} term, $i = \{1,..., m\}$;
- The request R, composed of m OR-ed terms, is translated into m directed hyperedges in E where the i^{th} hyperedge connects vertex R to all conditions of the i^{th} term, $i = \{1,..., m\}$;
- For each property p appearing in the conditions in the graph, the set C_p of conditions defined over the same property p is translated in an undirected hyperedge in E^u connecting all conditions in C_p.

The Figure 3.18 illustrates the hypergraph modeling our running example, where hyperedges in E (E^u, resp.) are represented as arrows (dotted boxes, resp.) linking (surrounding, resp.) the involved conditions. The computation of a solution to Problem 3.1 can be interpreted as a coloring of the vertices of the hypergraph, starting from the vertex representing R and recursively propagating the color through the directed hyperedges in E.

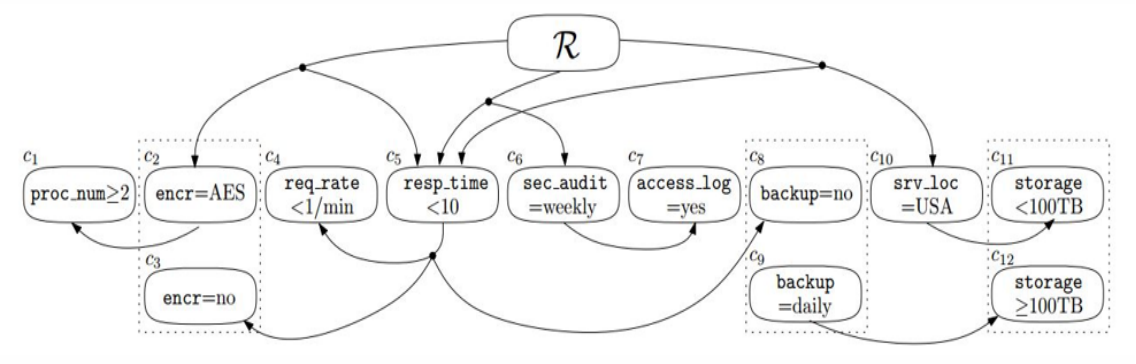


Figure 3.18: Graphical representation of Problem 3.1 for our running example

Note that, when more than one hyperedge originates from the same vertex $v \in V$, it is sufficient to propagate the color through one hyperedge (recall that m hyperedges correspond to m OR-ed terms). Such color propagation through directed hyperedges guarantees that the colored vertices represent a set of conditions satisfying the first two

conditions in Problem 3.1. In fact, directed hyperedges link all conditions included in the OR-ed terms in R, and also conditions in the OR-ed terms in the dependencies enabled by the coloring of a term in R.

The Figure 3.19 illustrates the hypergraph of Figure 3.18 after the color propagation from R through the hyperedge representing $R : (c_5 \wedge c_{10})$.

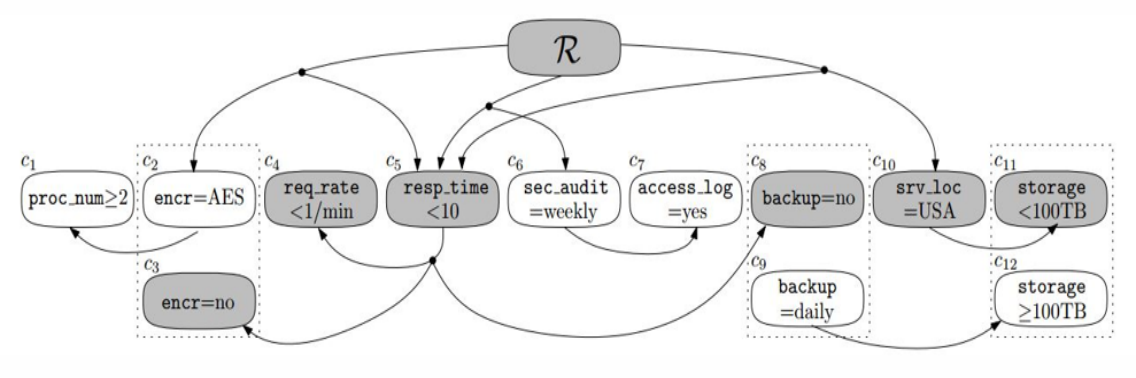


Figure 3.19: Color propagation in the hypergraph of Problem 3.1 for our running example

Once color has propagated through directed hyperedges in E, undirected hyperedges in E^u can be exploited to check the satisfaction of the conflict condition (Condition 3) in Problem 3.1. By restricting $G\ (V, E, E^u)$ to $G'(V, E^u)$, such condition is satisfied iff the colored vertices represent an independent set for G'. In fact, since all conditions defined over the same property are connected through an undirected hyperedge, if for every hyperedge $[v_i,..., v_j]$ in E^u, at most one vertex $v_x \in \{v_i, ..., v_j\}$ is colored, then the set of colored vertices in G' includes at most one condition for every property. Figure 3.19 shows that the set of colored vertices form an independent set for $G'(V, E^u)$. To find a solution to our problem, we represent it as a CSP [159], which can then be solved with CSP solvers [160].

3.5.2 CSP FORMULATION

According to [160] a Constraint Satisfaction Problem can be formulated as follows: given a triple (X, D, K), with X a set of variables, D the domain of variables in X, and K a set of constraints over X, find an assignment $w : X \rightarrow D$ that satisfies all the constraints in K. Our translation interprets:

- All conditions appearing in R and in the set D of dependencies as the set X of variables;
- The set of integers $\{1, 0\}$ as the domain D of the variables in X;
- The requirement R, the set D of dependencies, and the conflicts among conditions as the set K of constraints.

A solution to the problem so defined corresponds to a value assignment $w(c)$ to all conditions in C such that w satisfies all the constraints in K. With reference to our

hypergraph, X corresponds to the set V of vertices excluding R, D corresponds to the domain of colors (1 translates to gray), K corresponds to R and the dependencies and conflicts modeled through hyperedges, and w corresponds to the coloring function.

We now illustrate how the requests, dependencies, and conflicts can be translated into equivalent CSP constraints.

1. Requests: Given a service request

$$R : \vee_{i=1}^{m} (\wedge_{j=1}^{k(i)} c_{i_j})$$

Composed of m OR-ed terms, then all conditions in at least one of these terms must be included in a vSLA. In terms of the CSP assignment function w, at least one of the m terms must be assigned value 1. Formally, a requirement R is interpreted as

$$\bigvee_{i=1}^{m} (c_{i_1} = ... = c_{i_{k(i)}} = 1)$$

2. Dependencies: Given a dependence

$$d : c_h \rightsquigarrow P(c_i, ..., c_j)$$

With $P(c_i, ..., c_j) = \vee_{i=1}^{m} (\wedge_{j=1}^{k(i)} c_{i_j})$, then *all* conditions in at least one of the m OR-ed terms must be included in a vSLA if c_h is also included. In terms of the CSP assignment function w, at least one of the m terms must be assigned value 1, if also c_h is assigned value 1. Formally, a dependency $c_h \rightsquigarrow P(c_i,..., c_j)$ is interpreted as

$$(c_h = 0) \vee \left(\bigvee_{i=1}^{m} (c_{i_j} = ... = c_{i_{k(i)}} = 1) \right)$$

3. Conflicts: Given the set C_p of conditions over property p, then at most one condition $c \in C_p$ can be included in a vSLA. In terms of the CSP assignment function w, at most one condition $c \in C_p$ must be assigned value 1. Formally, a set $C_p = \{c_1, ..., c_k\}$ is interpreted as

$$(c_1 = ... = c_k = 0) \vee \left(\bigvee_{i=1}^{k} \left(c_i = 1 \wedge (c_{j_l} = ... = c_{j_{k=1}} = 0) \right) \right), c_{j_l} \in \{c_1, ..., c_k\} \setminus \{c_i\}$$

Note that CSP constraints correspond to the conditions of Problem 3.1, and a function w satisfying them corresponds to a correct value assignment f of Problem 3.1.

The Table 3.3 illustrates the CSP constraints for our running example. A vSLA will include all conditions assigned value 1 by the assignment function w.

Table 3.3: Requirement, Dependencies, and Conflicts with their CSP formulation

	Input	**CSP formulation**
Requirements	$(c_5 \wedge c_{10}) \vee (c_5 \wedge c_2) \vee (c_5 \wedge c_6)$	$(c_5 = c_{10} = 1) \vee (c_5 = c_2 = 1) \vee (c_5 = c_6 = 1)$
Dependencies	$c_5 \rightsquigarrow c_8 \wedge c_4 \wedge c_3$	$(c_5 = 0) \vee (c_8 = c_4 = c_3 = 1)$
	$c_{10} \rightsquigarrow c_{11}$	$(c_{10} = 0) \vee (c_{11} = 1)$
	$c_2 \rightsquigarrow c_1$	$(c_2 = 0) \vee (c_1 = 1)$
	$c_9 \rightsquigarrow c_{12}$	$(c_9 = 0) \vee (c_{12} = 1)$
	$c_6 \rightsquigarrow c_7$	$(c_6 = 0) \vee (c_7 = 1)$
Conflicts	$C_{storage} = \{c_{11}, c_{12}\}$	$(c_{11} = 0 \wedge c_{12} = 1) \vee (c_{12} = 0)$
	$C_{backup} = \{c_9, c_8\}$	$(c_8 = 0 \wedge c_9 = 1) \vee (c_9 = 0)$
	$C_{encr} = \{c_2, c_3\}$	$(c_3 = 0 \wedge c_2 = 1) \vee (c_2 = 0)$

The CSP translation illustrated in Table 3.3 can be solved by adopting any CSP solver and obtains a result assigning value 1 to the colored vertices in Figure 3.19, corresponding to a vSLA (i.e., a solution to Problem 3.1). With the determination of a vSLA we can then consider the responsibilities assigned to each property.

3.6 SHARED LIABILITY IN CLOUD SLA

In order to create a vSLA where the customer has the control over some aspects of the service, also the customer must give mechanisms that support this feature. In this sense our approach applies the use of shared liability and policies to establish a certain level of freedom for the negotiation.

Usually the QoS parameters are used by companies to check their technical performance and to check customer satisfaction with its services. Quality monitoring of services provided to companies in SLAs gives to provider the opportunity to support new services and applications, building a strong reputation based on stable relationships and brand image, allowing maintenance and increasing its market share. Based on this premise we can infer that the same monitoring of internal QoS can be extended to the monitoring of external metrics to CP.

So that the provider can support new services it is necessary to establish a context for the use of agreed resources. The challenge, then, is based on creating an enabling environment to meet different customer needs, balancing issues of control and transparency, and at the same time can be monitored through the contextual information provided by the shared liability relationship.

In our approach we assume that any property required by customer can have many assumptions for its smooth functioning. And these dependencies must be mapped and used in accordance with shared information, both internal and external to the provider. In this case we use the NIST definitions given in [118] which defines abstract and concrete metrics.

Thus we adopt the definition of abstract metrics to provide the customer a choice of parameters it needs, regardless of the resources provided by the provider, i.e., independent of the list of services provided by each provider, the customer can request

a particular property based on their needs and control of services thus enabling the usage control over their information. In this way we ensure that any property can be requested by the customer.

But for the management of these abstract metrics (and consequently, the control of concrete metrics) is important define the behavior of the same. Moreover, using this approach, we can define the use of dynamic and static properties, which could be considered both to ensure a certain level of service by the supplier as to require a certain level of use by the customer.

If a metric is static the property takes a value that is unchanging in time. But if the metric is dynamic, i.e., the property value has changes over time, then it is necessary to establish the relationship of responsibility between the entity that controls and controlled property.

The determination of responsibility for each property is defined by the cloud provider in the service conditions. By default each property is the provider's responsibility but if we have a dependence on external properties such as inherent to the customer properties, this relationship can be considered as a dependence to the shared responsibility. By definition our approach on shared liability presents the following situations:

- No Liability Statement: A service condition may have a property that does not have a direct responsible, for example, a condition of incompatibility where a property can not assume certain value such as "Location $\neq$ North Korea", in this case the provider can inform that do not performs services in a particular location or the customer requests a service indicating an undesired location. Therefore we have no direct responsible for this condition;

- Provider's Liability: When a property has values or conditions that are uniquely provider responsibility, such as "Backup Frequency = weekly", where the provider assumes that backs up the information according to the established period or "Location = USA", when the provider ensures that where the information is stored will be the same as described in the agreement;

- Customer's Liability: Once we present properties that depend on other properties some of them may be different responsibility of the service provider, for example "Request Rate $<$ 1/min" that determines the rate of requests made by the customer;

- Liability of Both Parties: Some properties may have shared liability itself, i.e., the two entities present in the agreement (provider and customer) are responsible for maintaining the property value, such as: "Contract Time = 1 year".

We then briefly describe these situations in the Table 3.4:

Table 3.4: Properties on Shared Liability Situations

Property \ Liability	No	Provider	Customer	Both Parties
Location	$\neq$ North Korea	= USA		
Backup Frequency		= weekly		
Request Rate			< 1/min	
Contract Time				= 1 year

Note that with this approach, we can clearly identify the entities responsible for each property thus facilitating the monitoring of dependencies and we can say that any change in the dynamic value of a property may indicate violations in the SLA between the parties.

3.7 CHAPTER SUMMARY

In this chapter, we address the problem of creating a Service Level Agreement that meets the customization needs of customers ensuring control over the process of migrating to a Cloud environment. The proposed solution is based on a modeling based on a generic ontology that takes advantage of an approach based on XML Schema representing the relationships between different services in Cloud Computing. The problem constraints observed in some types of service is then reformulated in terms of dependencies modeling the fragments which satisfy the constraints and incompatibilities. The set of services and attributes is treated generically as properties and each property is represented in tuples of service conditions which carry the necessary information for the correct formulation of a valid Service Level Agreement. We present the characteristics of ontology, describing the XML Schemes for both the definition of service conditions and the definition of requests and an approach to the treatment of new properties and values to solve the problem of customization and generalization. We also present an approach for managing and monitoring external properties to the Cloud Provider, based on shared liability. First, we describe how the properties should be treated and defined by providers and then present an approach to support the setting using colored hypergraph to represent the problem of definition of a valid SLA beyond to demonstrate how we can solve the problem through Constraint Satisfaction Problem formulation.

4

AUTOMATED FRAMEWORK TO CLOUD SLA MANAGEMENT: CONCEPTUAL MODEL

In the previous chapter we have detailed important aspects of the creation of a generic ontology, fundamental to the development of the framework proposed in this thesis, which are: the different definitions of services and the consequent variety of ontologies and present a generic definition to support the dynamic nature of cloud environments.

The approach proposed in this work, in addition to use of techniques of Multi-Criteria Decision Making (MCDM), is the creation of a conceptual framework for analysis of models of Service Level Agreements. The most general objective of the proposal is to develop a conceptual tool that facilitates the work of the cloud-services providers, enabling efficient analysis on market needs and to allow the negotiation of service agreements that keep track of the needs of users as well as keeping transparency. This tool is based on a generic ontology to support adaptable and extensible components in Advanced SLA Management and allows support for future markets in Cloud Computing.

Since the creation of a generic ontology is not enough to achieve our goal of SLAs customization it is important to create mechanisms that can support this approach, in this sense we present a conceptual framework designed specifically to meet the particularities of our ontology.

4.1 INTRODUCTION

The generic ontology presented is part of a broader context that is the conceptual specification of an infrastructure for utilization in Service Level Agreements, whose objective is to support the potential services in Cloud Computing and provide greater control and freedom to customers to determine their needs. This infrastructure aims to support the search and discovery of cloud services that are independent by the domain.

The ontology is then used as the basis for research and composition of Service Level Agreements in the framework.

The conceptual framework proposed in this thesis is based on the rules and formalities of object orientation and represented in the notation of the UML class diagram methodology. The purpose of a conceptual framework is to provide a class diagram that can be used as a basis for modeling of the application domain classes. A conceptual framework does not necessarily imply a finished and executable product, but in a conceptual data schema that subsequently must be translated into a specific data schema.

The framework for advanced management of SLA has been proposed with the aim of easing the process of creation and use of services based on Cloud. The main concern of the research reported in this thesis was, from its inception, the search for an approach to determining SLA for Cloud-based services, and provide freedom to customers to order services based on their needs, could facilitate the task of adaptation and extension of new properties in an organized manner for the services providers.

The challenge to be overcome is the existence of a lot of services in Cloud, which creates a difficulty to be exposed to their target audience and applied more consistently. Aiming to overcome this challenge, this thesis proposes the use of an infrastructure approach that enables the documentation, storage and dissemination of the different Cloud services through a repository of properties, documented from a metadata profile, manageable and accessible via Web services.

4.1.1 CHAPTER OUTLINE

This chapter describes the objectives of the framework and the proposed architecture for its effective implementation. It shows the scenario of using the framework based on established properties and values determined by providers beyond the conceptual description of the framework operation. The main contribution of the Chapter is the presentation of a conceptual framework that enables the use of our generic ontology through a simplified structure allowing flexibility and adaptability in the composition of Service Level Agreements.

4.2 OBJECTIVES

The framework proposed in this thesis uses the concepts of template repository and semantic web services. The objective of this framework is to provide an infrastructure for registration, search, composition and simulation of components used in SLAs in Cloud Services. Specifically, the framework should allow:

1. Registration and storage of semantic models for representing templates for Service Levels Agreements and service composition in cloud computing.
2. Searches based on the semantics expressed in models with recovery templates as a generic ontology.

3. Composition of SLA models to generate more complex templates that can be stored and used as MSA.

4. Control and monitoring of valid SLAs, i.e. the submission of models for tools that allow the simulation model and the return of the results generated.

The semantic description of properties allows the composition with other properties, while referring to models that can be simulated in established tools (or building such models, if necessary), it provides great flexibility to the process of creation of SLAs, common in the Cloud Computing environment.

This framework can serve as a basis for the development of new components and to explore other areas of research in cloud computing, hypercloud or also in Internet of Things, providing a modular architecture for its own expansion. Each component is also encapsulated in web services that may be run remotely (either independently or with other compound components).

4.3 FRAMEWORK STRUCTURE

In essence, a data repository is a computing environment, usually a Web site capable of storing data and metadata , i.e., resource, about a particular subject. The interested community can access to the resource at any time, being available for indefinite period of time by the repository. An overview of the framework architecture for this repository is shown in Figure 4.1. The framework implement concepts of SOA and considers two distinct layers, as shown:

- *Advanced SLA Framework*: is the framework itself. It is implemented as a web service, and is independent of the interface that accesses their services, facilitating integration with existing tools.

- *Customer/Provider*: It is the interface for interacting with users of the framework and can be developed in any language with access to web services. This layer considers two types of users for the framework: customers and providers.

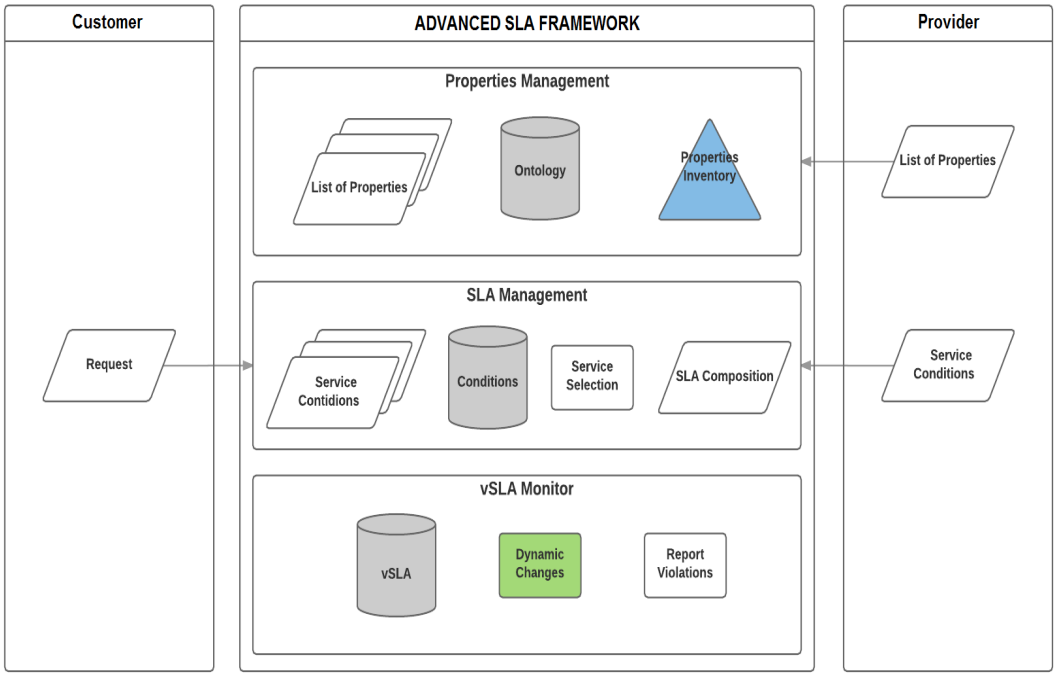


Figure 4.1: Overview of the Framework Architecture

The interface layer for customers users is responsible for receiving service requests from clients, while the interface for providers users is responsible for receiving the list of established properties and the conditions of these services.

4.3.1 THE USERS LAYERS

Usually when working with frameworks, it is necessary to develop class libraries and their subroutines. This way of setting up and running components shows quite cumbersome and unintuitive process that implies the use of documentation understand the interfaces and write the code to be able to use the libraries.

The idea of creating a framework based on XML Services allows for easy composition of the components by utilizing any graphical interface. This allows the framework a dual role serving for two types of users:

Providers can use the framework to register their services and check for new market demands in addition to being able to use the framework as an aid mechanism for monitoring their own agreements.

Customers who use the framework to search for available service environments in the Cloud and can make the composition of services based on their needs, besides the possibility of adding new services by solicitation.

To build a framework without worrying too much about the interaction with the user, it is necessary to define interfaces independent from implementation. With respect to this premise, efficient access to already implemented object classes is facilitated with

the use of generic ontology, allowing its easy extension since new components also follow the same format.

The usual form of object orientation is often based on domain analysis, which leads to the development of specialized and specific interfaces for each domain. Usually most of the software interfaces are not adequate to fulfill the needs of customization and interoperability.

When these subsystems are considered as part of a large class of subsystems, it is possible to define new interfaces that are applied to the entire system, and not only to a single instance. If there are multiple subsystems on the same system, it can be categorized in the same way then there are immediate benefits for interoperation.

As we have different services that can be made available in Cloud Computing and since each user interface can be developed independently of the internal implementations of the framework, it is possible to carry out changes in the independent interfaces without an impact on the system as a whole.

The internal and external interfaces of the framework define the static structures of the proposed approach. This approach allows customers to be protected from framework components, thereby reducing the number of objects to be used by the user, making it easier to use the framework. For this to be possible we need an efficient mechanism for storing and retrieving the properties and templates.

In this sense the properties can be grouped into service conditions according to each description and the short hierarchical structure, the modularity and the need for extension of service agreements can be described using the XML representations presented in the previous chapter.

The Figure 4.2 presents a diagram of Use Cases that demonstrate the interaction between the users of the framework.

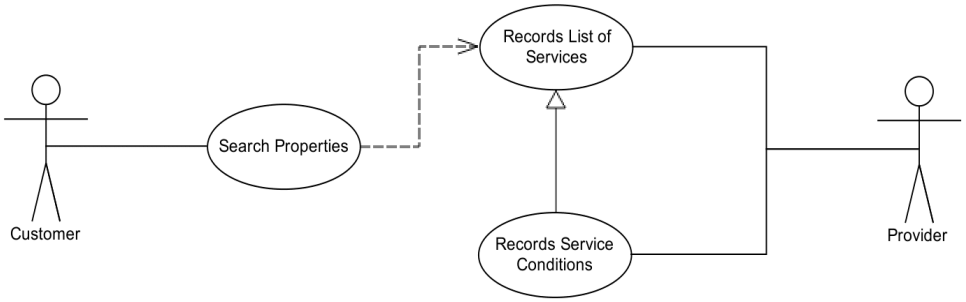


Figure 4.2: Scenario of interaction between the users of framework, represented in Use Cases.

Focused on customer problem, which is the search for cloud services that meet their needs and maintain their control documented in the contract, and provided with a list of services and conditions offered by the provider; it is the framework that reconciles the information to perform analysis considering the scenarios that might solve the problem.

As exemplified by the Use Case in Figure 4.2 and using the generic representation services through the properties a customer can search services using only properties and values as the example shown in the Table 4.1.

Table 4.1: Example of a service request used in the search for properties

Customer	Response Time	Service Location	Storage Capacity	Encryption Algorithm	Price
ABC	< 30 (ms)	= Uruguay (country)	= 100 (TB)	= AES (name)	< 350 (US$/year)

Translating the information presented in Table 4.1 to our definition of property, we have the following request service: R = {*resp_time* <30, *location* = Uruguay, *storage* = 100, *encryption* = AES, *price* < 350}. Based on the definitions provided in chapter 3 we can represent the example of Table 4.1 as follows in Figure 4.3:

$$p_1\text{: } resp\_time < 30 \text{ (ms)}$$
$$p_2\text{: } location = \text{Uruguay (country)}$$
$$p_3\text{: } storage = 100 \text{ (TB)}$$
$$p_4\text{: } encryption = \text{AES (name)}$$
$$p_5\text{: } price < 350 \text{ (US\$/year)}$$

Figure 4.3: Formal representation of a service request

In this chapter we treat this search based only on values and properties previously established by providers, so the property names and values must be already present in the list of available services.

The list of services maintained by the framework is a simple list of all services and possible values submitted by all providers, such as: *resp_time*={10, 20, 30}, *location*={USA, Uruguay, France}, *storage*={100, 500}, *encryption*={AES, DES, no}, etc. In this way our approach is shown to be compatible with existing market approaches facilitating its extension and utilization.

Therefore any provider can register their services. The services registration, besides being based on the lists of properties and values, it is also based on service conditions and can occur in two ways:

1. *Registration of service plans*: Where the properties are grouped into distinct services plans.
2. *Registration of single properties*: Each property and its dependencies are recorded separately.

In the first case the properties are registered in service plans that represent the provider implementation possibilities. Where the provider can register each set of properties and dependencies combinations as a single block of service conditions. As in the example shown in Table 4.2.

Table 4.2: Example of a service plan

Service ID: 15225							
Provider	Properties						
Amazon	Resp.	ID	Name	Value		Unit	Dep.
	Provider	01	Storage	Range		TB	02
				Min.	Max.		
					100		
	Resp.	ID	Name	Value		Unit	Dep.
	Provider	02	Location	Equal		country	
				USA			
	Resp.	ID	Name	Value		Unit	Dep.
	Provider	03	Response Time	Range		ms	04
				Min.	Max.		05
					30		06
	Resp.	ID	Name	Value		Unit	Dep.
		04	Backup	Equal		response	
				no			
	Resp.	ID	Name	Value		Unit	Dep.
	Customer	05	Request Rate	Range		Req./min.	
				Min.	Max.		
					1		
	Resp.	ID	Name	Value		Unit	Dep.
		06	Encryption	Equal		response	
				no			
	Resp.	ID	Name	Value		Unit	Dep.
		07	Price	Equal		US$/m.	
				100			

In this example the provider Amazon offers a service plan with a stated maximum data storage capacity of 100 TB which is located in the USA, adding a service of response time less than 30 ms, backup is not allowed, none use of encryption and the request rate should be less than 1 per minute, the provider also determines that the price for this plan is US $ 100 per month.

Using our definitions we can represent the example of Table 4.2 as follows in Figure 4.4:

c_1: *storage* < 100 (TB)
c_2: *location* = USA (country)
c_3: *resp_time* < 30 (ms)
c_4: *backup* = no (response)
c_5: *req_rate* < 1 (req./min.)
c_6: *encryption* = no (response)
c_7: *price* = 100 (US$/m.)

Figure 4.4: Formal representation of a service plan

Considering the service conditions and the dependencies presented for the properties c_1 and c_3 we can summarize the service plan as shown in Figure 4.5.

15225: $c_1 \rightsquigarrow c_2; c_3 \rightsquigarrow c_4 \wedge c_5 \wedge c_6; c_7$

Figure 4.5: Example of a service plan summary

This allows the provider to present their service plans in accordance with the combinations it considers appropriate. Thus each provider can present its list of service plans (as shown in Table 4.3) which will be used later as input to MCDM techniques to determine the best choice for the customer.

Table 4.3: Example of a list of service plans

Service ID	Service Conditions
15222	$c_2; c_5; c_6; c_7$
15225	$c_1 \rightsquigarrow c_2; c_3 \rightsquigarrow c_4 \wedge c_5 \wedge c_6; c_7$
15226	$c_8 \rightsquigarrow c_9 \wedge c_{13}; c_{10} \rightsquigarrow c_6; c_{11}$
15227	$c_9; c_{10} \rightsquigarrow c_6; c_{13}$
15230	$c_8 \rightsquigarrow c_9 \wedge c_{13}; c_{15}$
15232	$c_4; c_{12}; c_{16}$
15233	$c_3 \rightsquigarrow c_4 \wedge c_5 \wedge c_6$

As can be seen each service plan allows the sale of a set of properties. Considering, for example, the service plan 15222, we can see four properties (c_2, c_5, c_6 and c_7), whereas the price of the service plan is also a property, our approach allows us to state a services plan available in the framework is self sufficient and can be used to form the SLA.

But let us say that a customer only needs the property c_2, that obliges the customer to purchase a plan with more properties. So we must ensure that the framework can support any form of service delivery. This is possible through the registration of single property.

As an example of this situation we present the Table 4.4.

Table 4.4: Registration of a single property

Service ID: 15233							
Provider	Properties						
Amazon	Resp.	ID	Name	Value		Unit	Dep.
	Provider	03	Response Time	Range		ms	04 05 06
				Min.	Max.		
				30			
	Resp.	ID	Name	Value		Unit	Dep.
		04	Backup	Equal		response	
				no			
	Resp.	ID	Name	Value		Unit	Dep.
	Customer	05	Request Rate	Range		Req./min.	
				Min.	Max.		
					1		
	Resp.	ID	Name	Value		Unit	Dep.
		06	Encryption	Equal		response	
				no			

In this example the provider Amazon is only providing the *response time* service (c_3), the other properties (*backup*-c_4, *request rate*-c_5 and *encryption*-c_6) are the dependencies to ensure the service.

Considering the property c_3 we have then two occurrences, as shown in Table 4.5.

Table 4.5: Records to property c_3

Service ID	**Service Conditions**
15225	$c_1 \rightsquigarrow c_2; c_3 \rightsquigarrow c_4 \wedge c_5 \wedge c_6; c_7$
15233	$c_3 \rightsquigarrow c_4 \wedge c_5 \wedge c_6$

The first occurrence (the Service ID 15225) refers to a services plan that provides, in addition to the property c_3, also the property c_1 with its dependencies and the price for the service plan (property c_7). The second occurrence (Service ID 15233) refers to record only the property c_3 (and its dependencies). This gives the freedom to the provider that can create SLA templates based on their ability.

Given a list of services and values determined by the provider we can take an example as demonstrated in Table 4.6.

Table 4.6: Example of a list of services

Services	**Values**
Number of Processors	=1 =2 =3
Encryption	=AES =no
Response Time	<10 ms
Auditing Frequency	=weekly
Backup	=no =daily
Location	=USA =Italy =France
Storage	<100 TB ⩾100 TB to ⩽200 TB
Price	=50 US$/m. =100 US$/m. =150 US$/m.

In the example shown in Table 4.6 is presented a list of different services converted into properties that can be grouped to SLA composition. Once the user interfaces of the framework are based on XML files, both service requests, as the service conditions, can be presented in a format that can be easily handled by the framework. In the case of services, a provider can provide both service plans as single properties using XML files that meet the given XML schema for the service conditions, as shown in Figure 4.6.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceConditions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Users\Gerson\Desktop\Works\ArquivosXML\ServiceConditions.xsd"
Provider="Amazon" ServiceID="15233">
  <Property>
    <Responsible>Provider</Responsible>
    <ID>03</ID>
    <Name>Response Time</Name>
    <Value>
      <Range>
        <Minimum>30</Minimum>
      </Range>
    </Value>
    <Unit>ms</Unit>
    <Dependency>
      <Property>
        <ID>04</ID>
        <Name>Backup</Name>
        <Value>
          <Equal>no</Equal>
        </Value>
        <Unit>response</Unit>
      </Property>
    </Dependency>
    <Dependency>
      <Property>
        <Responsible>Customer</Responsible>
        <ID>05</ID>
        <Name>Request Rate</Name>
        <Value>
          <Range>
            <Maximum>1</Maximum>
          </Range>
        </Value>
        <Unit>req./min.</Unit>
      </Property>
    </Dependency>
    <Dependency>
      <Property>
        <ID>06</ID>
        <Name>Encryption</Name>
        <Value>
          <Equal>no</Equal>
        </Value>
        <Unit>response</Unit>
      </Property>
    </Dependency>
  </Property>
  <Property>
</ServiceConditions>
```

Figure 4.6: XML file to service conditions

As we can see by the example, the XML file shown in Figure 4.6 is exactly the information in Table 4.4. This proves that the framework is sufficiently flexible enough to support any composition determined by the provider and any request from the customer.

4.3.2 THE FRAMEWORK LAYER

Focusing on information available and after receiving the customer's request, the framework should look for alternative solutions to meet the required needs. For this the framework services are divided into three management modules:

1. *Properties Management*: Responsible for properties storage and integration of ontologies in the database, as well as to the inventory of properties for the service providers.
2. *SLA Management*: Responsible for interaction with users of the framework. Its purpose is to suggest the templates list to the service selection and later the SLA composition.
3. *vSLA Monitor*: Responsible for controlling the SLAs and monitoring the change of scenarios and possible violations.

In this chapter we present some aspects related to modules 1 (Properties Management) and 2 (SLA Management). Module 3 (vSLA Monitor) will be described in Chapter 7.

The **Properties Management** module must consider the structure shown in Figure 4.7.

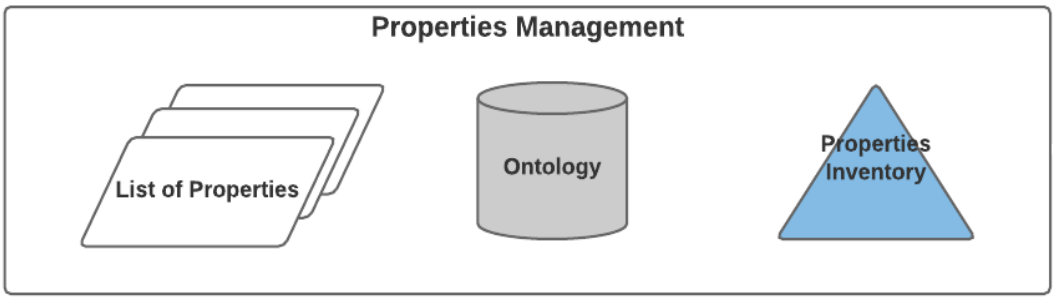


Figure 4.7: Properties Management module

This module receives the different lists of established properties from providers and integrates the generic ontology in addition to performing the inventory of properties (that will be explained in the next chapter of this thesis). The example in Table 4.7 allows us to understand a practical real case application, involving services provided by three providers. In this example, we have considered three lists of different providers, each one containing the relation of available services and values that each property can take according to the provider.

Table 4.7: List of services to the Properties Management Module

Services	Values of Provider A	Values of Provider B	Values of Provider C
Encryption	=AES =no	=RSA =AES =no	
Location	=USA =Italy =France	=Uruguay =Brazil =Argentina	=USA =Italy =Brazil ≠North Korea
Storage	<100 TB ⩾100 TB to ⩽200 TB	<100000 GB	<100 TB ⩾100 TB to ⩽200 TB >200 TB

In the example presented in Table 4.7 the provider A provides a list of services containing *encryption*, *location* and *storage* with the respective values for the properties; the provider B offers the same services with different values for the properties and the provider C offers a list containing only 2 services (*location* and *storage*) and their values.

These services are considered established properties by the framework and a fragment of XML schema that represents this is shown in Figure 5.3.

```
<xs:element name="Established">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Encryption" />
      <xs:enumeration value="Location" />
      <xs:enumeration value="Storage" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 4.8: Fragment of XML schema with the established properties shown in the example

The list of absolute values present in the list of services is used as a set of values that can be assumed by the properties in the XML schema provided by the framework. As these lists are independent, the framework needs groups them in a integrated ontology that can be presented as in Figure 4.9.

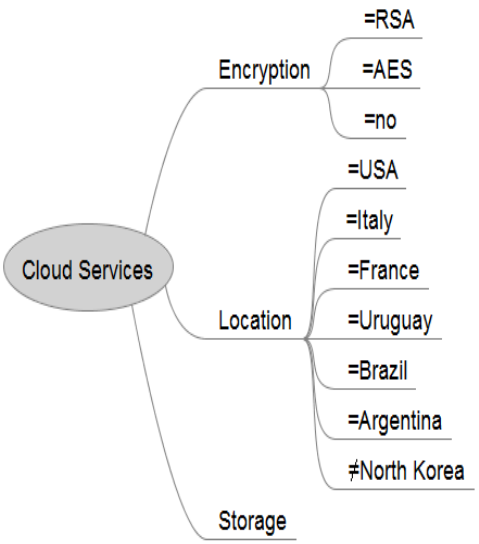


Figure 4.9: Integrated Ontology

Note that the integrated ontology does not present the values for the property *storage*, this is because these values are not absolute but ranges of values. This ontology is then published and serves as the basis for the absolute values used in the XML schema. An example of how these absolute values are used by XML Schema is shown in Figure 4.10.

```
<xs:element name="Equal">
  ...
      <xs:element name="Set">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="RSA" />
            <xs:enumeration value="AES" />
            <xs:enumeration value="no" />
            <xs:enumeration value="USA" />
            <xs:enumeration value="Italy" />
            <xs:enumeration value="France" />
            <xs:enumeration value="Uruguay" />
            <xs:enumeration value="Brazil" />
            <xs:enumeration value="Argentina" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
  ...
</xs:element>
<xs:element name="NotEqual">
  ...
      <xs:element name="Set">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="North Korea" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
  ...
</xs:element>
```

Figure 4.10: Fragment of XML schema with absolute values for established properties

These absolute values represent the alternative choice that customers have to request in a Service Level Agreement. According to our approach, the XML schema used does not use namespaces just to provide the necessary freedom for the customer to select their properties according to their needs. For this reason it is important that the ontology is published for those developers who need to use the XML schema in their interfaces not commit syntax errors.

Briefly this approach integrates the lists of services from different providers and presents a public ontology that determines the XML schema, this ontology allows customers to use the listed absolute values and the established properties as demonstrated in Figure 4.11.

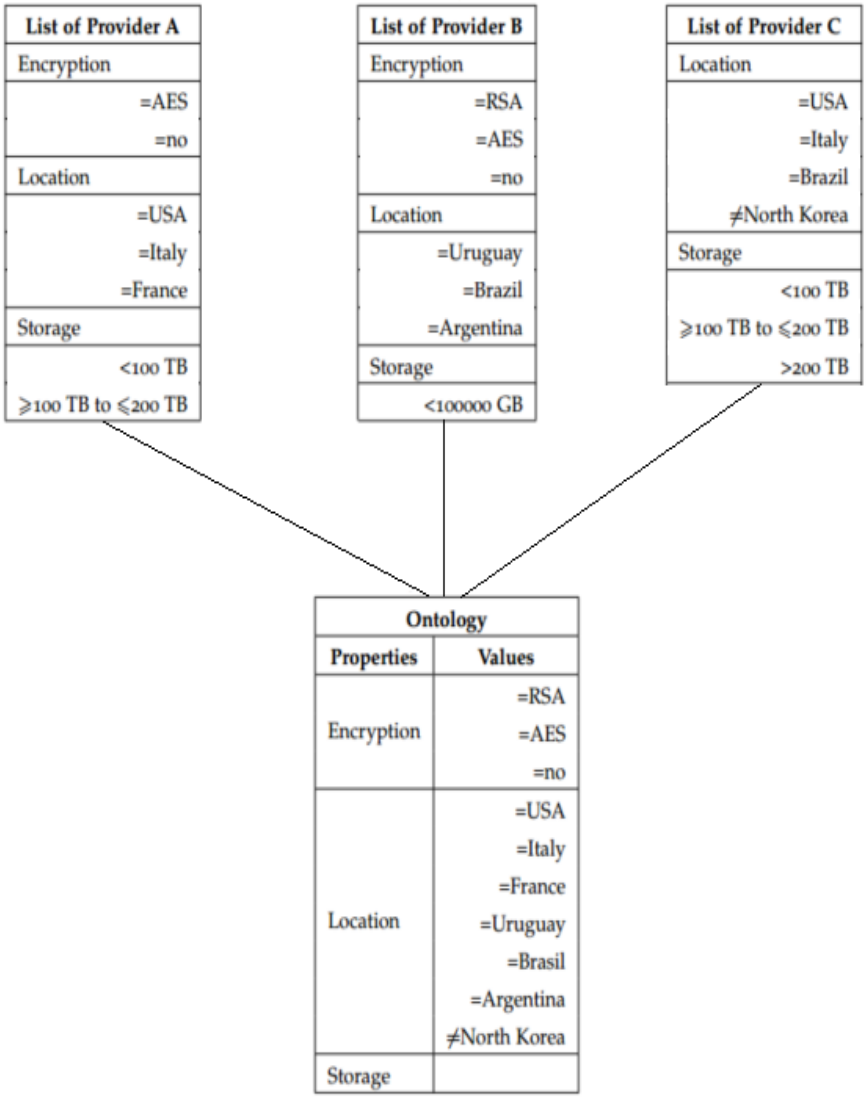


Figure 4.11: Lists of services integrated in a public ontology

The integration of the list of services in a public ontology has great advantages such as standardization and code reuse, and especially the possibility of variation in the construction of specific Service Level Agreements to different fields in Cloud. Thus,

features such as aggregation capability of new services, clarity and overall vision was fulfilled in the best possible way in our approach once the conceptual framework proposed in this work keeps entirely free the possibility of variation for the determination of SLAs.

Another issue to be addressed by the framework is that some values for the same property can be displayed at different scales, the framework needs to determine a common scale so that the values are understood in the same way by the customer. Taking as an example the property *storage* offering values in GB and TB, the framework needs transforms all values on a same basis to be used in the service selection. Therefore, in this case, the value "100000 GB" is converted into "100 TB".

After scaling transformation the list of properties and values can be then translated into a table of properties as shown in Table 4.8.

Table 4.8: Table of Properties

Services	Service ID A	Properties A	Service ID B	Properties B	Service ID C	Properties C
Encryption	11212 11213	c_2 c_3	12125 12126 12127	c_1 c_2 c_3		
Location	11214 11215 11216	c_4 c_5 c_6	12128 12129 12130	c_7 c_8 c_9	13401 13402 13403 13404	c_4 c_5 c_8 c_{10}
Storage	11217 11218	c_{11} c_{12}	12131	c_{11}	13405 13406 13407	c_{11} c_{12} c_{13}

This table of properties is used internally by the framework to determine the service selection, thus the framework has a list of established properties and default values arranged in a generic ontology to manage the properties, so the next step is to determine the SLA composition, this is accomplished by the **SLA Management** module. This module is represented by the structure presented in Figure 4.12.

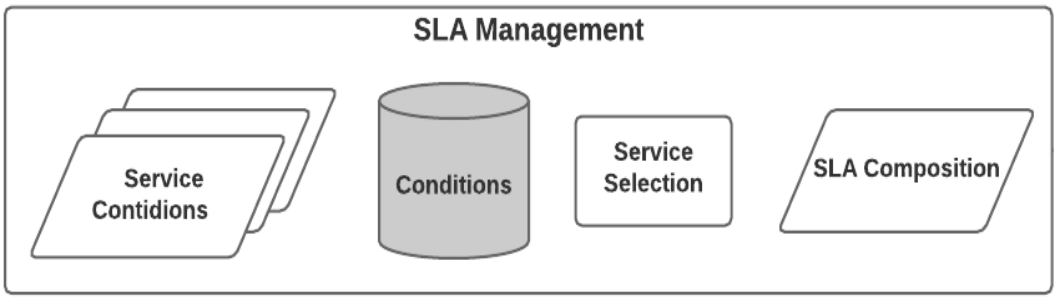


Figure 4.12: SLA Management module

This module receives the provider service conditions and stores them in a repository which is used to compare with the requests of customers, determining thus the selection of services that can matches and consequently creates the SLA composition. Considering the proposed example we assume that, at first, there are no conditions for the execution of the listed services and the customer X requests the following services regardless of price: R= {*Storage*=80 TB, *Location*=Italy, *Encryption*=AES}. As the request for the property *Storage* is equal to 80 TB the framework can assume that the request is for a value less than 100 TB what formally means the property c_{11} and the request may be represented as: R= $\{c_{11}, c_5, c_2\}$.

Based on this information the framework selects the services and returns a list of possibilities that can be used by external MCDM modules to determine the best choice for the customer considering their priorities. For this example, a selected service list can be presented as in Table 4.9.

Table 4.9: Preliminary list of selected services

Provider	Service ID	Properties
A	11217	c_{11}
	11215	c_5
	11212	c_2
B	12131	c_{11}
	12126	c_2
C	13405	c_{11}
	13402	c_5

In this case the only one provider that has services that match the customer's request is the provider A with the services {11217, 11215, 11212}=$\{c_{11}, c_5, c_2\}$.

After the framework performs the selection of services that matches the customer request, it is necessary that the service is accepted by the customer to be possible to make the SLA composition. Since most of the necessary information for the composition of a SLA can be manipulated as properties, and the structure of the ontology presented in our approach enables the identification of other information, the framework can establish that the SLA made for our example can be presented as in Table 4.10.

Table 4.10: Example of SLA composition by framework

<table>
<tr><td colspan="7">Customer: X</td></tr>
<tr><td colspan="7">Provider: A</td></tr>
<tr><td colspan="7">Service ID: 11217</td></tr>
<tr><td colspan="7">Properties</td></tr>
<tr><td>Resp.</td><td>ID</td><td>Name</td><td colspan="2">Value</td><td>Unit</td><td>Dep.</td></tr>
<tr><td rowspan="3">Both Parties</td><td rowspan="3">11</td><td rowspan="3">Storage</td><td colspan="2">Range</td><td rowspan="3">TB</td><td rowspan="3"></td></tr>
<tr><td>Min.</td><td>Max.</td></tr>
<tr><td></td><td>100</td></tr>
<tr><td colspan="7">Service ID: 11215</td></tr>
<tr><td colspan="7">Properties</td></tr>
<tr><td>Resp.</td><td>ID</td><td>Name</td><td colspan="2">Value</td><td>Unit</td><td>Dep.</td></tr>
<tr><td rowspan="2">Provider</td><td rowspan="2">5</td><td rowspan="2">Location</td><td colspan="2">Equal</td><td rowspan="2">country</td><td rowspan="2"></td></tr>
<tr><td colspan="2">Italy</td></tr>
<tr><td colspan="7">Service ID: 11212</td></tr>
<tr><td colspan="7">Properties</td></tr>
<tr><td>Resp.</td><td>ID</td><td>Name</td><td colspan="2">Value</td><td>Unit</td><td>Dep.</td></tr>
<tr><td rowspan="2">Provider</td><td rowspan="2">2</td><td rowspan="2">Encryption</td><td colspan="2">Equal</td><td rowspan="2">name</td><td rowspan="2"></td></tr>
<tr><td colspan="2">AES</td></tr>
</table>

In the example shown in Table 4.10 we can see the customer's identification, the provider's identification and the services compatible with the request, besides the responsible for the maintenance of each value.

Still considering the SLA management module, in another example we assume that these same properties can be grouped into services plans with their prices as shown in Table 4.11.

Table 4.11: Example of service plans from different providers

Provider	Service ID	Location	Storage	Encryption	Contract Time	Price
A	11219	=USA	<100 TB	=AES		=7.9 US$/m.
A	11220	=USA	<100 TB	=no		=4.9 US$/m.
A	11221	=USA	⩾100 TB to ⩽200 TB	=AES		=9.9 US$/m.
A	11222	=USA	⩾100 TB to ⩽200 TB	=AES	>1 year	=8.5 US$/m.
A	11223	=Italy	<100 TB	=AES		=7.9 US$/m.
A	11224	=Italy	<100 TB	=no		=4.9 US$/m.
A	11225	=France	<100 TB	=AES		=7.9 US$/m.
A	11226	=France	<100 TB	=no		=4.9 US$/m.
B	12132	=Uruguay	<1000000 GB	=RSA		=80 US$/y.
B	12133	=Uruguay	<1000000 GB	=AES		=59.99 US$/y.
B	12134	=Uruguay	<1000000 GB	=no		=24.99 US$/y.
B	12135	=Brazil	<1000000 GB	=RSA		=80 US$/y.
B	12136	=Brazil	<1000000 GB	=AES		=59.99 US$/y.
B	12137	=Brazil	<1000000 GB	=no		=24.99 US$/y.
B	12138	=Argentina	<1000000 GB	=RSA		=80 US$/y.
B	12139	=Argentina	<1000000 GB	=AES		=59.99 US$/y.
B	12140	=Argentina	<1000000 GB	=no		=24.99 US$/y.
C	13408	≠North Korea	<100 TB			=24.99 US$/y.
C	13409	=USA	⩾100 TB to ⩽200 TB			=59.99 US$/y.
C	13410	=USA	>200 TB			=80 US$/y.
C	13411	=Italy	⩾100 TB to ⩽200 TB			=59.99 US$/y.
C	13412	=Brazil	⩾100 TB to ⩽200 TB			=59.99 US$/y.

This table is a summary of the service conditions made by each provider separately. Each of these service conditions is received by the framework that performs the equivalences and stores them in one same place to be consulted at the time of service selection.

In this example we demonstrate the occurrence of two dependencies on service conditions. The first is shown in the service 11222 of provider A, where the price of US$ 8.5/m. only will be charged if the contract time is longer than one year, the second occurrence is perceived in the service 13408 of provider C which determines the storage capacity less than 100 TB in places that are different from North Korea.

Considering the equivalence between prices with annual values and monthly values and storage in gigabytes and terabytes we can translate the Table 4.11 as shown in Table 4.12.

Table 4.12: Table of service plans

Provider	Service ID	Properties
A	11219	$c_4; c_{11}; c_2; c_{15}$
A	11220	$c_4; c_{11}; c_3; c_{14}$
A	11221	$c_4; c_{12}; c_2; c_{16}$
A	11222	$c_4; c_{12}; c_2; c_{20} \rightsquigarrow c_{21}$
A	11223	$c_5; c_{11}; c_2; c_{15}$
A	11224	$c_5; c_{11}; c_3; c_{14}$
A	11225	$c_6; c_{11}; c_2; c_{15}$
A	11226	$c_6; c_{11}; c_3; c_{14}$
B	12132	$c_7; c_{11}; c_1; c_{17}$
B	12133	$c_7; c_{11}; c_2; c_{19}$
B	12134	$c_7; c_{11}; c_3; c_{18}$
B	12135	$c_8; c_{11}; c_1; c_{17}$
B	12136	$c_8; c_{11}; c_2; c_{19}$
B	12137	$c_8; c_{11}; c_3; c_{18}$
B	12138	$c_9; c_{11}; c_1; c_{17}$
B	12139	$c_9; c_{11}; c_2; c_{19}$
B	12140	$c_9; c_{11}; c_3; c_{18}$
C	13408	$c_{11} \rightsquigarrow c_{10}; c_{18}$
C	13409	$c_4; c_{12}; c_{19}$
C	13410	$c_4; c_{13}; c_{17}$
C	13411	$c_5; c_{12}; c_{19}$
C	13412	$c_8; c_{12}; c_{19}$

Supposing that a customer needs a storage service of 200 TB, this information is translated as property c_{12} and the services that meet this request are presented as in Table 4.13.

Table 4.13: Services to meet the request by property c_{12}

Provider	Service ID	Properties
A	11221	$c_4; c_{12}; c_2; c_{16}$
A	11222	$c_4; c_{12}; c_2; c_{20} \rightsquigarrow c_{21}$
C	13409	$c_4; c_{12}; c_{19}$
C	13411	$c_5; c_{12}; c_{19}$
C	13412	$c_8; c_{12}; c_{19}$

The result presented in Table 4.13 shows five possibilities for the customer and each one can be described as in the Table 4.14 with the following characteristics:

Table 4.14: Characteristics of selected services for property c_{12}

Service ID	Characteristics
11221	Provider A with data storage capacity between 100 and 200 TB (inclusive) which is located in the USA, using AES encryption at the price of US$ 9.9 per month.
11222	Provider A with data storage capacity between 100 and 200 TB (inclusive) which is located in the USA, using AES encryption at the price of US$ 8.5 per month with contract time more than 1 year.
13409	Provider C with data storage capacity between 100 and 200 TB (inclusive) which is located in the USA at the price of US$ 59.99 per year.
13411	Provider C with data storage capacity between 100 and 200 TB (inclusive) which is located in the Italy at the price of US$ 59.99 per year.
13412	Provider C with data storage capacity between 100 and 200 TB (inclusive) which is located in the Brazil at the price of US$ 59.99 per year.

Based on this result the customer may refer to MCDM techniques with their own utility functions to determine the best choice. We can also observe the dependence presented in the service 11222 that may or may not be accepted by the customer, established a negotiation phase to the composition of the SLA. Once the characteristics are clearly understandable and the service is agreed, this information can be then used to create the SLA composition and the contract between the parties.

4.4 THE FRAMEWORK OPERATION

Basically, the framework uses the inputs from the providers and customers to perform the selection of services and, after negotiation, to show the composition of SLA to players, as shown in Figure 4.13.

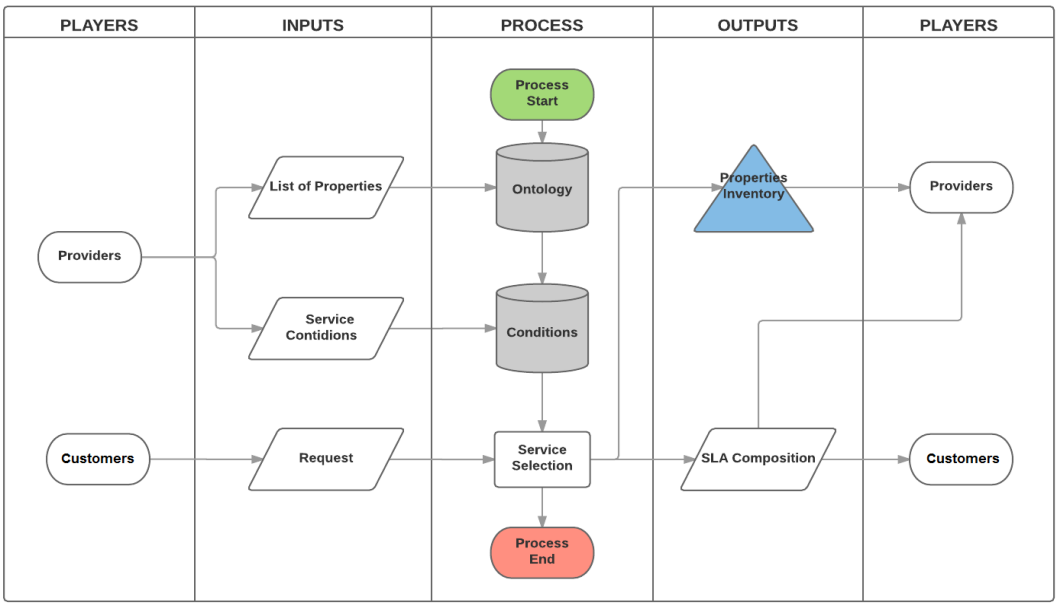


Figure 4.13: Example of inputs and outputs of framework

Figure 4.13 shows the inputs of providers that are lists of properties (which are handled by the Properties Management module and grouped in a single ontology) and the service conditions (which are treated by SLA Management module based on pre-defined ontology) and the customer input that is the service request.

After the selection of services and acceptance by the customer, one of the outputs delivered by the framework is the SLA composition that is sent to both the provider and the customer. Another presented output is the inventory of properties that will be described in the next chapter of this thesis.

The framework provides four services to each user:

- *Registry*: Registration of the SLA models in the system. From a document (XML file) provided by Cloud Services Provider, the lists of established properties and service conditions are stored in their respective databases.
- *Search*: Search SLA models. The customer provides a request encoded query and receives as a result a list of properties that satisfies the query or approaching it (the new properties are handled by properties inventory and will be treated in the next chapter).
- *Compose*: SLA composition. The framework provides an XML file to specify which services should be composed and the amounts agreed for the same. With the agreement between the customer and the provider, a valid SLA (vSLA) is generated and stored in the database.
- *Control*: vSLA Monitoring. The framework compares vSLA with requests that originated them in order to determine any dynamic exchange and also checks the properties in order to indicate possible violations.

The integration of these services with the functional modules of the framework is shown in Figure 4.14.

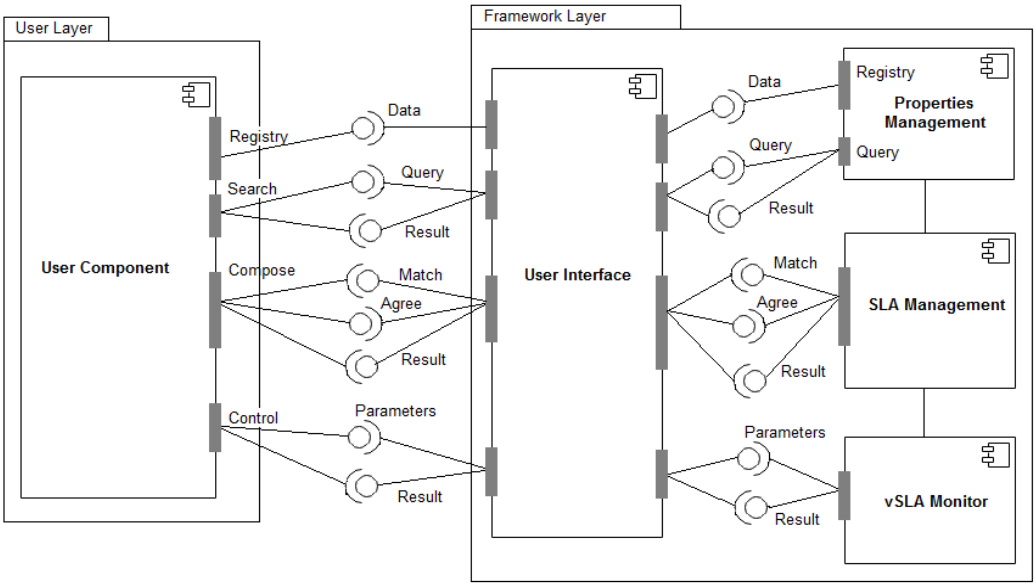


Figure 4.14: Services and framework modules

This approach promotes loose coupling between subsystems and customers allowing them to make changes to the subsystem without affecting the clients. As our approach is based on XML schema, each of the interfaces on the components is either an input point, output or input and output to XML files. These interfaces consist of a description and its corresponding direction.

Simple components are correlated to an implementation. This implementation contains the location of the classes that implement the component. Each implementation must have a main class, but can also refer helper classes necessary for its implementation. Another important point to create the framework is the runtime environment. This execution environment should receive the XML files, control them and return the results.

Regarding the user layer, it can describe the services and operations of the user component as follows:

Registry holds the record of the information contained in the lists of established properties by providers and also the record of service conditions. These data are internally processed by the Property Management module.

Search can be run either by customers or as by service providers. In the case of use by customers this operation performs the search for properties and values, in addition to enabling the query in the form of service request. When used by providers, this operation enables query contained in the property inventory (which will be explained in detail in the next chapter of this thesis).

Compose Receives the list of services that match with service requests by customers. After performing the selection of the services the SLA management module waits for the customer agreement to then carry out the SLA composition.

Control is responsible for controlling over the valid SLA, these operations are performed internally by vSLA Monitor module that will be detailed in the Chapter 7.

The general operation of the framework, including processing requests and properties inventory, requires the use of a prepared database to store the characteristics to be manipulated. The class diagram shown in Figure 4.15 represents the conceptual model proposed to organize these features.

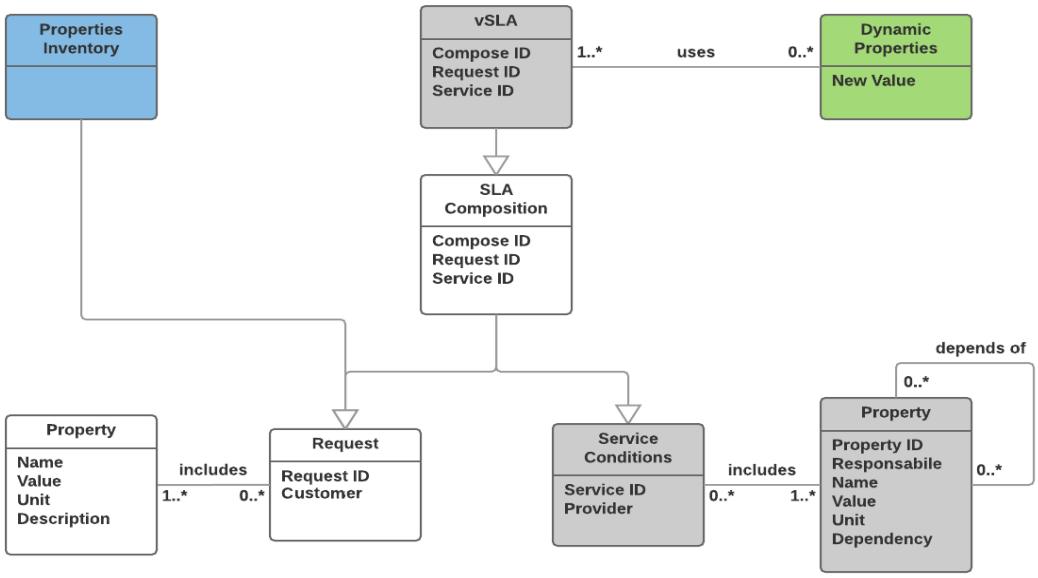


Figure 4.15: Framework Class Diagram

Both the objects created by the class "Request" and "Service Conditions" must comply with the definitions set forth by the XML schema provided by the framework. The class "Request" in addition to contain information relating to the customer and the identification of the request also includes information on the required properties such as name, value and unit, it should be noted that if it is requested a new property (not established by providers) this relationship can also include the description for this new property. Class "Service Conditions" contains the identification of the provider and the service, in addition to include information on the properties that composes the service (whether it be described as a single service or as a services plan). In the case of class "Property" used in the service conditions this class has, besides the identification of the property, its name, value and unit, also has the responsible for the property, that guarantees the aspects of shared liability presented in Chapter 3, and the dependences between properties that are necessary.

The "SLA composition" class is performed by the SLA Management module and after the selection of services and the customer agreement is executed inheriting the information contained in the classes "Request" and "Service Conditions".

The classes "Properties Inventory", "vSLA" and "Dynamic Properties" will be described in the next chapters.

4.5 CHAPTER SUMMARY

In this chapter we present the proposed conceptual framework that is meant as a repository of information about Cloud services. This framework is configured as a storage system, accessible via network, which has mechanisms for managing properties and allows to add objects, search for information relating to it and make them available to the end user. Its architecture enables the definition of the properties, as the definition of rules that composes the service conditions. The handling of data and metadata is made through web services exposed by using interfaces that allow management, access and search. In addition, the framework can be used as an integrated component systems that provides additional functions for organizations or end users. As the framework architecture is designed to manage different types of Cloud services and this management is through web services, it is natural that retrieving information about objects occurs in the same way. The semantic search is also foreseen by SLA composition module that enables the use of internal ontology of the framework itself. Implemented as a web service, it can be used and integrated into already existing applications. The framework structure and operations have been described and examples of use of the framework illustrate their viability.

5

MANAGEMENT OF NOVEL PROPERTIES AND VALUES

In Chapter 3 we have described the Ontology proposed in this thesis, with its structure, features and a detailed description of its elements. In Chapter 4 we have described the conceptual model of the proposed framework for using the Ontology, its architecture and the description of some of its modules. In this Chapter we present the operation of the Property Inventory component used in the Property Management module. This component is responsible for the organization and manipulation of new services required by customers.

5.1 INTRODUCTION

In general, to support the different needs of its users, Service Levels Agreements need to be more flexible, especially in relation to its facilities. As in the context of Cloud computing, the need for customizable feature set of adjustment is an important factor. It is often necessary that the systems allow their customization in terms of SLAs in order to extend its flexibility of adaptation and extension, in addition to changes of the configuration. However, this type of customization is not always feasible because the providers do not provide their internal structure of services and open source systems are usually built on integrated architectures, which customization is complicated by the need to understand the system code as a whole to perform any modification.

Adding to the complexity in architecture, there are other features, available in the SLA, that also hamper their adaptation. For example: fixed roles and permissions on the system; platform dependency and specific databases and set of features designed according to local experiences (usually experiences of developers).

Considering the difficulties of adapting the Service Level Agreements, it becomes relevant to investigate models that offer a higher level of flexibility for these systems. A perfectly adaptable approach to this problem is the use of frameworks and software components. Frameworks are defined as semi-complete and reusable applications that,

when specialized, produce custom applications within a specific domain [161]. Software components are replaceable parts of the system that stress the interface and implementation separation. This separation facilitates communication between system components and the replacement of their implementations, because all communication with the component is made through its interface. Although they are typically meant for reuse, frameworks and components have characteristics, such as dependence on well-defined interfaces, design reuse and architecture, and the use of patterns that can assist in the development of systems with more organized architectures to allow easier adaptation and extension.

In the specific case of our framework, and presented ontology, we use "established" properties and "new" properties that allow customization of the SLA. These properties are handled by a specific module called "Properties Inventory" that will be detailed in this chapter.

5.1.1 CHAPTER OUTLINE

This chapter describes and illustrates how new properties and new values are managed through the Properties Inventory component, shows the scenarios where this module is used in our framework and shows how the new data can be clustered to allow the adaptation and extension for Service Level Agreements. The main contribution of this Chapter is the description of how the proposed framework identifies the demand for new services and how providers can use this information to deploy these services.

5.2 PROPERTIES INVENTORY

The fundamental problem addressed in this thesis is the customization of SLAs based on customer needs. In the previous chapters, we proved that it is possible to simplify the customization process using a generic ontology that can be described in an XML schema. However, in order to make sure that new properties and new values are accepted by the framework, mapping of existing properties and values is necessary. This mapping depends on information that characterizes each property individually and must allow the addition of new values without compromising the initial characteristics of each property. Another important factor is that each new property must be analyzed to restrict ambiguities between similar properties, only after this analysis it will be able to be submitted to the service providers so that they can alert the status of "New" for a "Established" property.

To address this problem, we insert in the proposed framework a component called Properties Inventory. This component is part of the Property Management module and is responsible for the manipulation of alterations in already existing properties values and controls of new properties and values. Basically this component receives all properties listed by service providers and stores it in a common list of properties, which, in time, can be consulted by the providers to determine the inclusion of new properties.

The Figure 5.1 depicts a process (simplified) to determine the inclusion of a property with status "established".

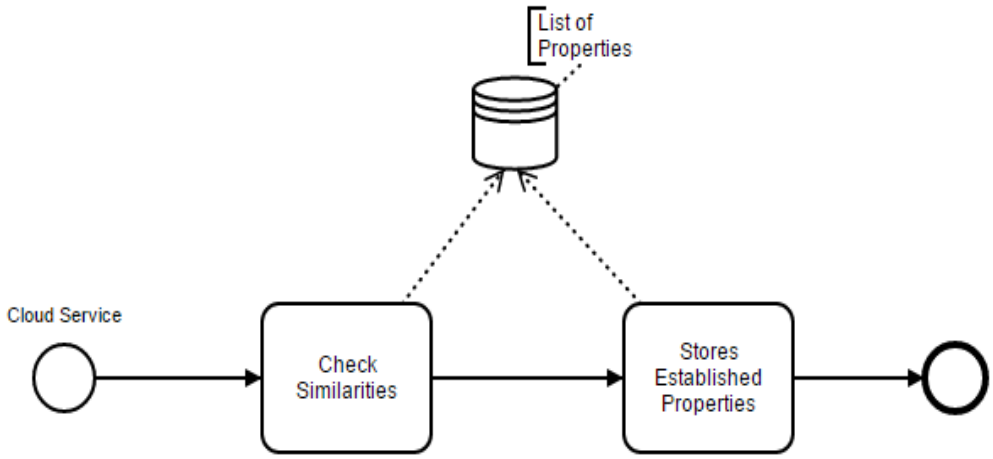


Figure 5.1: Process (simplified) to determine a "Established" property

As the first function of the Properties Inventory component has precisely determined the established properties, it is important to carry out a control on similar properties, so that there are not ambiguous or duplicate properties. As shown in Figure 5.1, when a Cloud Service (as a list of properties) is received by the Properties Inventory it should check for similarities between services previously established and, based on this result, sets new services or integrating similar services. For example if two providers provide the service "Response Time", the first attribution of Property Inventory is to determine whether this is a single property or in their descriptions are presented details that can establish a difference between them, creating two similar properties.

As our premise is to simplify the process of creating a SLA as well as facilitate the use of the framework for the users, the determination of a new established property can be carried out also by the provider. To illustrate this, let us analyze the following situation:

1. Our framework is started with a empty list of properties and values;
2. The provider A provides a list of services among them "Response Time";
3. The framework, through the Properties Inventory component, stores this list of properties with their respective descriptions, where the property "Response Time" is now p_α ("Response Time considering x");
4. The provider B needs to present its list of services, including a property also called "Response Time";

5. The provider B can see the list of properties in the Property Inventory and verify that the description of p_α is consistent with their own description for the property "Response Time";

6. In the case where they are equal and do not exhibit ambiguity, the two properties provided by different providers can be considered the same p_α, regardless of having different values for it;

7. If the provider B checks that the description of p_α is not consistent with their service him can create a new property, called for instance "Response Time considering y".

8. In this case, the Properties Inventory has two different properties for different services: p_α and p_β. Which can be requested by customers.

As in this situation, in which the providers determine the "behavior" of its properties, it is easy to identify that the occurrence of ambiguities is reduced to zero, since a provider knows to determine the difference between its own service and the service offered by another provider. This situation is a little different when we consider the customers of services and their needs. So that customers require that the list of properties and their descriptions need to be clear.

Since the property "Response Time" is not a trivial property and may consider several factors, the difference between properties p_α ("Response Time considering x") and p_β ("Response Time considering y") makes it clear to the provider and the customer which service is being hired and added to the SLA.

As seen in Chapter 3, a customer can request services using an ontology that allows the composition of a SLA according to their needs. In this sense a customer can request established properties with defined values (or new values) or requesting new properties. In case of established properties we can present this as described in Figure 5.2.

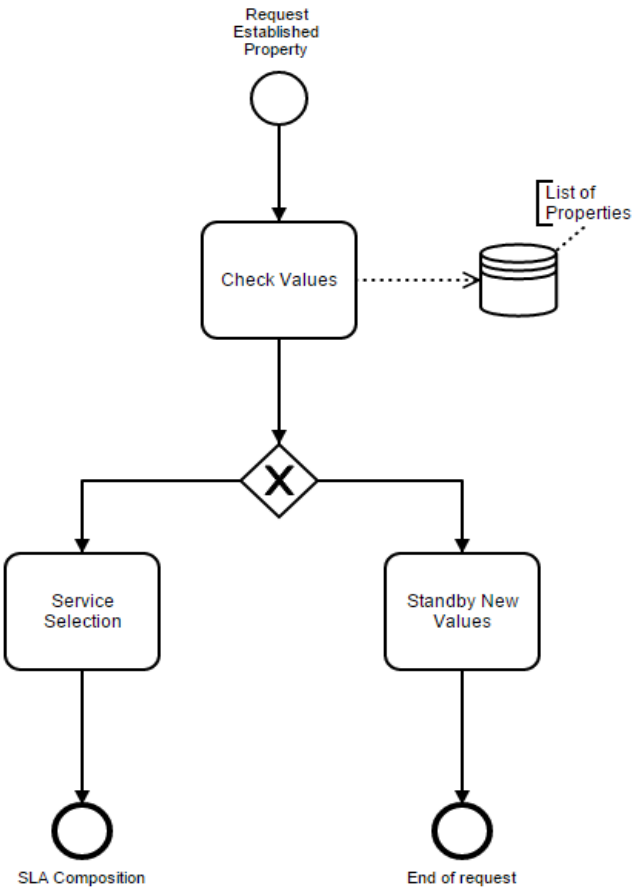


Figure 5.2: Process for requesting established properties

In the example shown in Figure 5.2, we have a request of a customer seeking services based on the list of established properties, and this list is presented to the customer using our XML schema (as in Figure 5.3):

```
<xs:element name="Established">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Encryption" />
      <xs:enumeration value="Location" />
      <xs:enumeration value="Storage" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 5.3: Fragment of XML schema with the list of established properties

Taking as example the property "Location", our framework provides a set of values determined for this property (Figure 5.4):

```
<xs:element name="Set">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="USA" />
      <xs:enumeration value="Italy" />
      <xs:enumeration value="France" />
      <xs:enumeration value="Uruguay" />
      <xs:enumeration value="Brazil" />
      <xs:enumeration value="Argentina" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 5.4: Fragment of XML schema with absolute values for property Location

If the customer requires the property Location with a value like "Spain", for example, but this value is not part of the set of possible values, the Properties Management module can determine the selection of possible services for the customer. Moreover, Properties Inventory can put the new value in a standby situation, to be further assessed by the providers of this type of service.

Another situation considered by the Properties Inventory concerns the new properties, an example of this process is demonstrated in Figure 5.5.

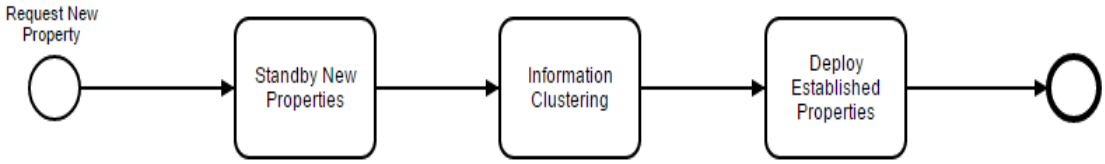


Figure 5.5: Process for requesting new properties

As shown in Figure 5.5 a customer can request a property not yet established, such as a new property called "Video Streaming", this new property is put on standby and the information related to their description need to be clustered to reduce ambiguities. After that a service provider can query the inventory of new properties with the option to deploy new services.

In summary the Properties Inventory component has four functions:

1. Keep the control on established properties;
2. Put new values and properties in a standby state;
3. Information clustering;
4. Control of changes.

These functions can be used in different scenarios, as described in the next section.

5.3 USAGE SCENARIOS

As the Properties Inventory component is part of the property management module and this composes the proposed framework, we can describe some scenarios that can be handled directly by the functions assigned to the component. So, the following scenarios present, briefly, the usage possibilities of the framework:

- $\mathbf{S_1}$: From a service or feature of interest, the user searches in the generic ontology the semantic representation and sends a request to the framework. After obtaining this representation, the framework searches in a repository which models are somehow associated with the service or resource. From the list of SLA candidate models, the user selects the most suitable one to make the composition of the SLA.
- $\mathbf{S_2}$: The user can request a new service model by composing new properties or existing properties with new values. These new properties and/or values are stored and inventoried.
- $\mathbf{S_3}$: Based on the Properties Inventory, a service provider can generate new SLA models. These new models of Service Level Agreements can be created considering the new properties and new values demanded by customers.
- $\mathbf{S_4}$: After some change in the status, the framework can inform the users in order to fulfill a preliminary request.

Because these scenarios are specific to the use of our framework, obviously it does not exist algorithms that address these situations, so that we present our own algorithm and for a better understanding we illustrate this as follows.

5.3.1 APPLICATION EXAMPLE FOR THE SCENARIO $\mathbf{S_1}$

The first usage scenario is the simplest of all: the user searches for services that have properties and values already established. In this case, the Property Management module starts an algorithm that searches in the Properties Inventory (the first part of the algorithm is represented by the pseudocode in Algorithm 1) .

Algorithm 1 Properties Inventory (part 1)

Input: *RP*: requested property, *EP*: established properties, *rv*: requested value, *sv*: setted values
Output: *SS*: list of selected services

```
1: procedure SEARCH(RP, EP, rv, sv)
2:     if RP = EP then
3:         sv' ← sv ∈ EP
4:         if rv = sv' then
5:             SS ← (EP, sv')
6:             return SS
7:         end if
8:     end if
9: end procedure
```

As described in Algorithm 1, the inputs for the algorithm are the requested property by the customer (*RP*), the list of established properties (*EP*), the requested value (*rv*) and the setted values for the established properties (*sv*). As output, we have the list of selected services (*SS*). After the search of information, that can be made by customers using our XML schema, the request (*RP*) is compared with the list of established properties (*EP*) (in line 2) and existing compatibilities are compared the requested values (*rv*) with the set of values (*sv*) assumed by the property (line 4). Considering the service conditions determined beforehand by the provider, the framework can return the list of selected services (*SS*) that match with the solicitation (lines 5 and 6).

Using the example shown in section 5.2 we can consider the request for a service of location in USA like *R*:(Location=USA) where "Location" is the established property and "USA" is one of the absolute values for this property. As we saw in chapter 3 this tuple (*p op val*) in the provider side is a service condition, then hypothetically we can say that this tuple takes the identification c_4 to the framework.

With the compatibility of the properties and values required, the framework returns the list of selected services considering the service conditions made by each provider, as shown in Table 5.1.

Table 5.1: List of compatible services for the property Location

Provider	Service ID	Properties
A	11219	$c_4; c_{11}; c_2; c_{15}$
A	11220	$c_4; c_{11}; c_3; c_{14}$
A	11221	$c_4; c_{12}; c_2; c_{16}$
A	11222	$c_4; c_{12}; c_2; c_{20} \rightsquigarrow c_{21}$
C	13409	$c_4; c_{12}; c_{19}$
C	13410	$c_4; c_{13}; c_{17}$

This list of services enables the customer to hold the choice of service plans based on their priorities.

5.3.2 APPLICATION EXAMPLE FOR THE SCENARIO s_2

The second usage scenario presents two situations, the first one is the solicitation of new properties and the second is the solicitation of new values for properties already established. To control these two situations is necessary to perform some changes to our initial part of the algorithm, complementing it as shown in Algorithm 2:

Algorithm 2 Control of new properties and new values

Input: *RP*: requested property, *EP*: established properties, *rv*: requested value, *sv*: setted values

Output: *SS*: list of selected services, *PS*: property in standby, *vS*: value in standby

```
1:  procedure SEARCH(RP, EP, rv, sv)
2:      if RP = EP then
3:          sv' ← sv ∈ EP
4:          if rv = sv' then
5:              SS ← (EP, sv')
6:              return SS
7:          else
8:              vS ← rv
9:              SS ← (EP, sv)
10:             return (SS, vS)
11:         end if
12:     else
13:         PS ← RP
14:         return PS
15:     end if
16: end procedure
```

As inputs we keep the same information (*RP*, *EP*, *rv* and *sv*). As output, in addition to display the list of selected services (*SS*), the algorithm can set properties and values in standby status (*PS* and *vS*). Until the line 6 the procedure is the same presented in the first part of the algorithm after this the algorithm needs to control two situations. The first situation occurs when the customer requests a property that has not yet been established by the framework (lines 12). In this case, the new property is placed on standby by the Properties Inventory (lines 13 and 14). The second situation occurs when the customer requests a new value for a property already established (line 7). In this case, in addition to put the new value in standby (line 8), the framework also returns all possible values for the property requested (line 9). In this way, it is possible to present some alternatives to the customer, since their original request can not be met.

Regarding the first situation we can consider as example the new property "Video Streaming", as this property is not in the list of established properties is impossible for the framework to determine the compatibility of this new property to the list of

known properties, having the Properties Inventory put this new property on standby for further evaluation by providers.

For the second situation we take as an example a request consisting of two properties (Encryption and Location) with the following values: (Encryption=RSA) and (Location=Spain). Since this two properties are already established, the framework needs to control only its values. In this case the framework locates the compatible value for the property Encryption (c_1) but does not have a value that matches the value requested for the property Location, therefore the Properties Inventory puts the new value on standby and returns a service selection considering the possible values for both properties, obviously prioritizing the property that showed value compatible with the request, as shown in Table 5.2.

Table 5.2: List of compatible services considering the property Encryption

Provider	Service ID	Properties
B	12132	$c_7; c_{11}; c_1; c_{17}$
B	12135	$c_8; c_{11}; c_1; c_{17}$
B	12138	$c_9; c_{11}; c_1; c_{17}$

As can be seen in Table 5.2 the framework suggests other values for the requested property also considering other properties present in the service condition.

5.3.3 APPLICATION EXAMPLE FOR THE SCENARIO s_3

The third usage scenario considers the repositories established by Properties Inventory (Figure 5.6).

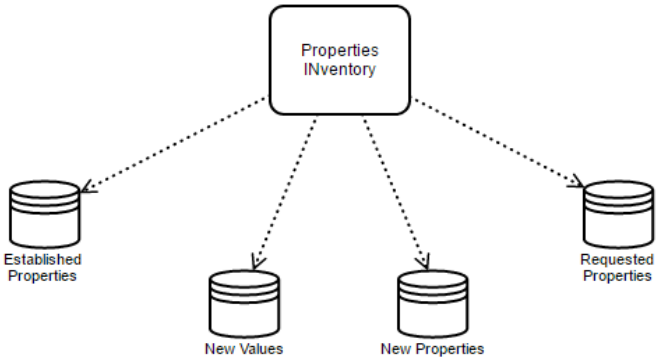


Figure 5.6: Repositories used by Properties Inventory

As can be seen in Figure 5.6 the Properties Inventory component is responsible for four different repositories: a) Established Properties; b) New Values; c) New Properties and d) Requested Properties. These four repositories are linked together and the

repositories "New Values" and "New Properties" are considered transient repository since their information tend to migrate to the repository of Established Properties. The repository "Requested Properties" will be described in the next subsection.

The proposed framework, in addition to allow customization and freedom to the customer requesting services according to their needs, also appears as a source of information for service providers. Once a provider can see the demand for new services (in the form of properties and values) this provider can consider deploying these requests to meet the market demand.

We can describe this scenario as the representation of service models that can be found, made and changed by the providers according to their market perception. To illustrate this situation take as example the Table 5.3:

Table 5.3: Example of information maintained by the properties repository

Established Properties	New Values	New Properties
Location USA Italy	Location Spain	Video Streaming
Encryption RSA AES		

As we can see in the example shown in Table 5.3, our approach presents a set of information in an organized and simplified manner. This simplification allows service providers to consult these data (mostly New Values and New Properties) allowing its implantation if they wish.

We assume that a provider, fulfilling the needs of customers in services located in Spain, resolves to meet this demand by establishing this service. This new situation is delivered to the framework when the provider presents a new service. Thus the Properties Inventory component passes a value, that was previously set in standby for an established property, with the new value (Table 5.4)

Table 5.4: Example of reallocation of property values

Established Properties	New Values	New Properties
Location USA Italy Spain		Video Streaming
Encryption RSA AES		

The same can happen with new properties, a provider can simulate the composition of their services to determine their plans and implement new services with their values, as shown in Table 5.5.

Table 5.5: Example of reallocation of services

Established Properties	New Values	New Properties
Location USA Italy Spain		
Encryption RSA AES		
Video Streaming 720p		

With this approach, the proposed framework can be established as an interfacing channel between the customer market of Cloud services (that need more freedom and control for the composition of their SLAs) and service providers (which need to be alert to new markets).

5.3.4 APPLICATION EXAMPLE FOR THE SCENARIO s_4

As the Properties Inventory also tracks changes status of properties and values (standby for established), we can use the information regarding these changes to inform customers in order to be able to meet any previously requested service.

For this, it is necessary to have a control over the Service Agreements established previously, this control is carried out by vSLA Monitor module that will be detailed in chapter 7. This module uses the information provided by the Properties Inventory that informs whenever a new value or a new property is established.

Considering the example shown in subsection 5.3.2, let us say that the customer 00452 has chosen the service 12132 with the following properties: (c_7, c_{11}, c_1, c_{17}) where c_7 is (Location=Uruguay) but their first request has property like (Location=Spain). As previously seen the value contained in the request was not met and this value then goes to standby status, so we have the following situation (Table 5.6).

Table 5.6: Example of status repository

Established Properties	**Requested Properties**		**Standby Values**	**Selected Properties**	
Location	Location: Spain			Location: Uruguay	
	User ID	SLA ID	Location	User ID	SLA ID
USA	00452	1521		00452	1521
Italy	01254	1678	Spain	Location: Italy	
Uruguay	00687			User ID	SLA ID
France				01254	1678

The information presented in Table 5.6 are:

1. Established Properties: the list of properties by providers;
2. Requested Properties: properties requested by customers;
3. Standby Values: list of property values in standby controlled by Properties Inventory module;
4. Selected Properties: properties selected by the customer and subsequently stored in a valid SLA (vSLA).

The items 1 and 3 have been described previously while items 2 and 4 needs to. For this, we observe what it happens when a property has its status changed from "Standby" to "Established" (Table 5.7).

Table 5.7: Example of status change

Established Properties	**Requested Properties**		**Standby Values**	**Selected Properties**	
Location	Location: Spain			Location: Uruguay	
	User ID	SLA ID		User ID	SLA ID
USA	00452	1521		00452	1521
Italy	01254	1678			
Uruguay				Location: Italy	
France				User ID	SLA ID
Spain				01254	1678

When a property passes to the status "Established", the Properties Inventory component checks the repository of Requested Properties if this property was required previously (the yellow items in Table 5.7), if the result is positive the Properties Management module informs the vSLA Monitor module through mapping of the SLA ID (the green items in Table 5.7).

Note that this mapping is also carried out in relation to the User ID, this approach allows the retrieval of information on the market needs and can identify customers who have requested these services. This is possible because the proposed framework are mapping new values in already known properties, but we must pay attention when it comes to new properties, for it, it is necessary to organize and cluster the new information so that they can be handled in the best way by the framework modules, this approach will be described in the next section.

5.4 INFORMATION CLUSTERING

As seen in previous sections the management of new values and new properties is responsibility of Properties Inventory component. This management is based on a structure of repositories that keeps information on the properties used in different services. The information on these services are from generic Ontology that was presented in Chapter 3 where the information on established properties are presented by on Cloud service providers and the information about new properties and new values are presented by customers in the form of service requests.

Regarding the information on new properties and new values so that they can be analyzed and established by the service providers, it is necessary that the framework produces it in a qualified way. It is not enough to simply store this information, they need to present some value to providers.

Considering the occurrence of new values is relatively easy to introduce an approach that allows a decision making by the provider. Since these are new values for established properties, it is enough that the framework applies a counter for each new displayed value. An example of this is shown in Table 5.8.

Table 5.8: Counter table for new values

Property: Location	
Value	**N. Occurrences**
Spain	16423
Chile	538
Poland	612
Pakistan	45

According to the example shown in Table 5.8, we can see an higher occurrence of requests requesting services located in Spain. In this way, the counter table allows the service provider to review this information and take their decision about it. This approach can not be used for new properties because it is not a trivial situation. For

this the framework must present an exploratory analysis of the descriptions for each new property requested.

Different customers may submit service requests with different properties names but with similar descriptions. Of course, if the list of established properties contains a comprehensive number of possible services that reduce the need to request new services by customers but the framework needs to consider this new situation. The framework, through the Properties Inventory, have the function to establish mechanisms that can organize this information. For this we suggest the use of text mining and clustering techniques which are a way to explore data analysis. In essence, the cluster analysis provides a means to explore and verify present structures in the data, organizing them into clusters of similar objects.

The purpose of this section is not to present or create techniques for extraction of knowledge, but to prove that our approach, based on the framework and generic ontology, can be perfectly used for these situations.

The goal of the clustering is to find a structure of clusters in the data, where each cluster contains objects that share some characteristics considered relevant to the field of data studied. As the service requests used in our Ontology receive descriptions of the properties in textual form through XML files, we can say that the texts contained in those files are important repositories and the intelligent organization of these textual collections is of great importance for the proposed framework as it streamlines the search process and information retrieval. In this context, the application of Text Mining techniques allows the transformation of this volume of textual unstructured data into useful knowledge.

The use of Text Mining techniques can extract knowledge from raw textual data (unstructured), providing support elements to knowledge management, which refers to how knowledge is created, used, shared, stored and rated. Technologically, the support of Text Mining for knowledge management takes place in the transformation of content information repositories for knowledge to be analyzed and shared among stakeholders.

For the unsupervised extraction and organization of the knowledge from textual data, the differential is in the standard extraction step, in which grouping texts methods to organize collections of documents in cluster are used. Then, they need to apply some selection technique descriptors for the clusters formed, that are, words and phrases that assist the interpretation of the groups. After validation of the results, the hierarchical clustering and their descriptors can be used as a topical hierarchy for exploratory analysis of texts tasks and support of information retrieval systems.

The three main steps are included in Text Mining (as shown in Figure 5.7): 1) Preprocessing of documents; 2) Standards Extraction with Texts Grouping and 3) Knowledge Assessment. In this section, our focus is on the first stage and how the files provided by the Properties Inventory can help to facilitate the extraction of knowledge by providers.

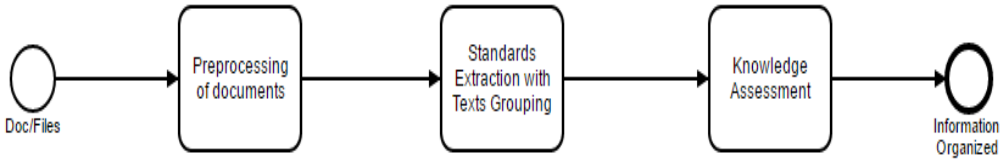


Figure 5.7: Stages of Text Mining process

The preprocessing stage is the main difference between the Text Mining processes and Data Mining processes: the structuring of the text in a format suitable for knowledge extraction. Many authors consider this step the most time consuming throughout the Text Mining cycle. The purpose of preprocessing is to convert text written in natural language (inherently unstructured) into a structured representation, concise and manageable by grouping text algorithms. For such task, standardization activities are performed on the collection of texts, selection of terms (words) most significant and, finally, the textual representation collection in a structured format that preserves the main characteristics of the data.

In this sense, the proposed framework, especially with the Ontology, enables this phase to be simplified since the terms to be analyzed and structured are in a common source provided by the structure of XML files.

One way to perform the selection of terms is evaluate them by simple measures statistics such as term frequency (known as TF) and document frequency (known as DF). The term frequency records the absolute frequency of a particular term throughout the text collection. The document frequency, in turn, counts the number of documents in which a term appears.

Once selected the most representative terms of our text collection (which are descriptions of new properties presented in the XML file) must seek the structure of documents, in order to make them processable by clustering algorithms that are used for bundling texts. The most widely used model for representing textual data is the space-vector model, in which each document is a vector in a multidimensional space, and each dimension is a term of the collection [162]. To do this, we can structure the text in a bag-of-words, in which the terms are considered independent, forming a disorderly assembly in which the occurrence of word order does not matter. The bag-of-words is a document-term table as illustrated in Table 5.9 where d_i corresponds to the i^{th} document, t_j is the j^{th} term and a_{ij} is a value that relates the i^{th} document with the j^{th} term. Note that in the representation shown here does not have class information, since the learning task grouping method is unsupervised.

Table 5.9: Table of document-term

	t_1	t_2	$\cdots$	t_m
d_1	a_{11}	a_{12}	$\cdots$	a_{1m}
d_2	a_{21}	a_{22}	$\cdots$	a_{2m}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
d_n	a_{n1}	a_{n2}	$\cdots$	a_{nm}

Using the table of document-term presented in Table 5.9, each document can be represented as a vector $\vec{d_i} = (a_{i1}, a_{i2}, ..., a_{im})$. The value of a_{ij} measure can be obtained in two ways:

- Using a value that indicates whether a term is present or not in a given document; and
- Using a value that indicates the importance of the term and distribution along the collection of documents, for example, the value of TF.

In our approach, we suggest the use of the second way. In order to perform the extraction of patterns after the representation of texts in a structured format, we should use methods of grouping texts for organizing the documents. The various grouping strategies are in practice algorithms that seek an approximate solution to the clustering problem. To illustrate a brute force algorithm that seeks the best partition of a set of n documents into k groups need to evaluate $k^n/k!$ possible partitions.

There are several tools that provide algorithms to clustering texts. Among the most relevant, we can mention the Cluto (Clustering Toolkit [163]), which supports the main clustering algorithms present in the literature, various measures of similarity, and graphical interfaces for analysis of results. It is important to note that due to extensive use of clustering algorithms, various mathematical and statistical software, such as Matlab [164] and R [165] also provide clustering algorithms that can be employed in standard extraction tasks.

Given the above, we can say that based on the files provided by our framework the service providers can apply their own techniques and tools to determine the demand for new services.

5.5 CHAPTER SUMMARY

In this chapter, we present the Properties Inventory component that integrates the Property Management module and works as a set of repositories. This component is responsible for the management of information on the properties previously established by providers and also the information on new values and new properties requested by customers. We present how the storage of this information is organized and how the status changes enable the interaction between the framework and service customers especially when there is a change in the set of services provided. Also, we present how

the information provided by the framework can be used to facilitate understanding of new demands by service providers.

6

USE OF FUZZY PARAMETERS

In the previous chapters we define the control of properties in SLAs using absolute values (or crisp values) but this traditional solutions for the SLA composition assume that the values required for the services to be static and punctual, not considering the needs or the differences in perceptions between the parties, which implies that a simple difference of values makes the customer's request unanswered and that the provider loses market or have no knowledge of it. Such an assumption may result in a market restricted to the providers. We solved this limitation by proposing an approach using values described in fuzzy logic [89]. This allows an approximation between the parties since they can use natural language to describe certain values and allows a wider range of service opportunities for providers.

6.1 INTRODUCTION

Despite the growth of fuzzy control applications in several areas, in many cases, it remains the difficulty in assessing the choice between fuzzy logic rather than the classic one. According to [166], there are two main reasons for the choice of fuzzy logic and not a classic. The first, when the process is unable to provide accurate information which can be quantified and used and the second, when the process supports imprecision. These two features can be seen in service requests demanded by customers as well as in certain resources controlled by providers.

Fuzzy sets do not give absolute relevance to its elements. Any element of the universe of discourse belongs to the fuzzy set, which obtains a degree of membership (membership function) that refers as how the element belongs to the set. Therefore, the interpretation given to the fuzzy set of elements is not "if the element belongs to the set" but "as it belongs to the set".

The fuzzy sets are important because they can represent inaccurate or diffuse knowledge, which can be expressed and manipulated to generate decisions. Human language

uses often concepts (linguistic terms) for the description of knowledge. The conditional statements, based on the theory of fuzzy sets, involving logical product (minimum) of the background blocks "IF", which are combined within the logical sums (maximum) of subsequent blocks "THEN".

Since in our approach the service providers can present their solutions in the form of service conditions we can see that these models are typically multidimensional and with multiple condition, related to the dependencies between the properties. In this sense, the fuzzy logic control algorithms can use linguistic terms to describe the process variables to safety, economy, effectiveness, ease and applicability, without the need for mathematical models.

Due to difficulties in process control and the complexity of simultaneous phenomena, the mathematical modeling and the model accuracy, the actuation time of control algorithms, dynamic conditions and knowledge of the process, the fuzzy logic becomes convenient. Such convenience works because it features the facilities on the appropriateness of human control strategy, simplicity of control laws, the flexibility of linguistic variables and accuracy for computational implementation.

For a given property Fuzzy logic can be used in a control system using the service conditions, it is necessary to take the result of the inference of the various rules (which are fuzzy sets) and turn it into numerical values to control signals associated with variables language used in the inferred propositions.

6.1.1 CHAPTER OUTLINE

In this chapter, we will describe where and how we can use the fuzzy logic within our framework and solutions presented by him. We address the problem of interpretation of values in Service Level Agreements considering the possibility of using fuzzy logic at different times in the determination of a SLA. Then, we illustrate our approach to the use of natural language services to customer requirements and how this approach can also be used on the provider side, ending with a practical example of this approach. The main contribution of the Chapter is the description of the utilization of fuzzy logic to model property values that can be requested by customers, while allowing wider flexibility in controlling these properties for the providers.

6.2 FUZZY LOGIC IN CLOUD SYSTEMS MANAGEMENT

Cloud systems management can exploit the advantages of fuzzy logic in different ways, from representing knowledge and observed facts in a way similar to humans (i.e., with natural language and its imprecisions), to approximate reasoning including incomplete or inaccurate information (like humans are able to do). We identify and analyze the different uses of fuzzy logic in cloud systems management, pointing out its adoption in various components, and the advantages it provides.

Fuzzy specification of customer's preferences on requirements. In many practical situations it may not be possible to satisfy all the requirements that the user specified in the SLA

because they are conflicting or there are not enough resources. Customer's requirements do not always have the same relevance and impact on the correct and timely execution of the applications. Fuzzy logic can be used to enable users to easily define the relative relevance or their preferences among requirements. For instance, in a big data application, a user can specify that storage requirement has "high importance", application performance has "medium importance", and user interface and interaction have "low importance". Merging requirements taking into account their relative preferences may guide the resource allocation engine in finding a feasible solution, by solving a multi-objective optimization problem. Considering that our approach presents another vision for the utilization of fuzzy parameters by the customer, this vision will be demonstrated in the next section.

Fuzzy allocation of resources to tasks. The allocation of cloud resources to the tasks of customer's applications is a complex problem, which needs to take into account various aspects (e.g., applications performance, customers and provider costs, energy consumption, resource usage, security, data protection, and system reliability and availability). This problem has conflicting goals and is usually addressed using a multiobjective decision making approach. In dynamic scenarios, where tasks are continuously activated/deactivated, a strategy that reaches the global optimum at each update is not practical as it may imply moving tasks and data. To overcome this issue, incremental approaches allocate (as optimally as possible) only the new tasks according to their SLAs. Fuzzy logic can be useful to support flexible reasoning during mapping of the requests of new tasks onto available cloud resources, especially by merging all requirements similarly to adaptable human reasoning. In particular, fuzzy similarity can be used to measure the distance between solutions and find the optimal allocation of resources. This approach will be demonstrated in Section 6.4.

Fuzzy monitoring and prediction of cloud status. Cloud systems evolve mainly because of workload activation and dynamic operation changes, faults or malfunctioning components, security threats and attacks, changing application needs, and need for energy saving. Monitoring the cloud system permits to understand and possibly anticipate critical situations, ensuring dependability based on continuity and quality of service. In fact, starting from observed behavioral trends, prediction tools are able to infer the future system behavior from the past condition and current situation. In critical situations, applications reallocation may be suggested to maintain the desired level of service quality and fulfill the SLAs. Monitoring and prediction of the cloud system operation can benefit from flexible reasoning based on fuzzy logic, suited to understand the cloud status and identify (nearly-) critical situations. This approach will be demonstrated in the next Chapter.

Fuzzy dynamic allocation of resources to tasks. Dynamic re-allocation of resources to applications tasks can be adopted in dependable cloud system management to ensure dynamic adaptation to changing operating conditions. This could in fact guarantee continuous complete fulfillment of the SLAs. The resource (re-)allocation algorithm

should then be executed, either periodically or when the status monitor/predictor points out a significant change, which may impair (or is shortly expected to impair) the normal system operation. Fuzzy logic can be useful also to support flexible multi-objective reasoning for dynamic adaptation of the resource mapping to the evolving operating conditions. Also this approach will be demonstrated in the next Chapter.

Finally, we can say that fuzzy logic can be applied to meet the customer's needs and to define parameters that can be used by services providers. To demonstrate this situation the next two sections show how fuzzy logic can be applied on the customer's and provider's sides.

6.3 FUZZY CUSTOMER REQUIREMENTS

Customers often tend to overestimate their requirements in SLAs, to prevent possible problems during the working of their applications. However, this leads to a waste of paid resources and reduces resource availability in the cloud. To limit this waste of resources (and money for final users), we propose to use fuzzy logic to support SLA specification. Fuzzy logic allows the customers to define their applications needs in a flexible way, capturing natural linguistic expressions, when customers are not specialists in information systems and technologies and when requirements are not so crispy defined or easily definable. Fuzzy logic can support the definition of customer's requirements as *fuzzy parameters* and *fuzzy concepts* as follows.

6.3.1 FUZZY PARAMETERS

Fuzzy parameters permit customers to define their requirements when they are unable to determine an exact, specific value of a characteristic of the cloud environment, but they are fully conscious of the required size of the considered characteristic and are linguistically able to describe it (e.g., with adjectives of periphrases). For instance, a customer who would like to use a large key for the encryption of her data, may not have a precise idea of the needed key length. In this case, the customer may prefer to state that a long key is needed, accepting a conventionally defined concept of "Long" as a fuzzy range of values. Mutual agreement about the significance of linguistic expressions is essential to understand and satisfy the customer's requirement.

As there can be differences of perception between what a customer meant by "Long" and what a provider defines a "Long", we must ensure that these differences are eliminated. That our framework can accept service requests containing requests with fuzzy values a provider must first establish these values, as seen in Chapter 3, then a property in a service conditions must present a fuzzy description as seen in Figure 6.1.

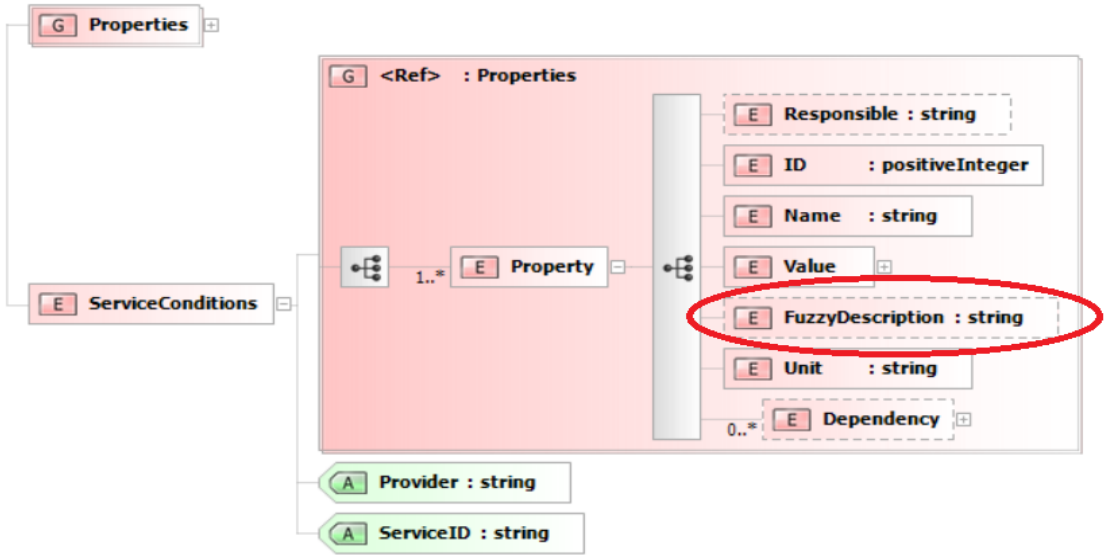


Figure 6.1: Element Fuzzy Description in a Service Conditions XML Schema

The Element "FuzzyDescription" shown in Figure 6.1 should describe the range of values determined for the fuzzy value.

For example a property "Key Length" defined in the conventional crisp way has the following typical form: *Key Length* = 384 bit and may be better replaced by a fuzzy definition like: *Key Length* should be "Long" provided that the customer follows the fuzzy definition of sizes defined by the provider (e.g., Figure 6.2).

Fuzzy label	Range
very short	1-32 bit
short	16-128 bit
medium	64-256 bit
long	128-1024 bit
very long	512-2048 bit

Figure 6.2: Example of fuzzy specification of key length parameter

The separation between ranges of key length is not crisp, but ranges may overlap. This fuzzy specification of parameters in SLAs allows cloud providers to manage with higher elasticity their resources, without living them unused when applications do not explicitly demand for them. This enables for better resource allocation, with higher quality of service at lower costs for both the provider and the customers.

It might be difficult for a customer to define the exact value of a given resource, that is facilitated when the provider has a set of options with their descriptions. As an example we can describe the following situation:

A service provider provides the property "Support Response Time" in fuzzy values presented as follows: High (12 hours $<$ 48 hours), Medium (5 hours $\leqslant$ 12 hours) and

Low (< 5 hours). Then, a customer can request this property in two ways: one using crisp values such as (Support Response Time < 12 hours) or by using the fuzzy determination displayed by the provider (Support Response Time = Medium). In this way, the provider can determine their service plans considering these rules and in the case of property "Support Response Time" can also control its service priority.

We can considers that the Service Agreement established between the customer and the provider consider the following tuple: (Support Response Time = Medium). Thus, any request that has a Support Response Time between 5 and 12 hours has been in accordance with the SLA. If in a particular demand the customer needs a faster response, for example in 1 hour, this would be a change of plan and could be charged by the provider. If, on the other hand, the provider answers any demand in more than 12 hours this would be a violation of the SLA. This approach allows the flexibility of SLA composition process and ensures its management in a more transparent manner between the parties.

6.3.2 FUZZY CONCEPTS

Fuzzy concepts operate at abstract level, allowing customers to define features that do not directly correspond to a cloud characteristic or parameter, but map on an appropriate combination of them. Fuzzy logic can provide the mathematical foundation for merging real characteristics and metrics, by translating the linguistic high-level description given by the customer. For example, if a customer wants to protect his data but does not know the characteristics that are relevant for data protection, he may prefer to request "high data security". The provider then can relies on fuzzy logic for formalizing and processing this concept, translating it into proper parameter values (e.g., see Figure 6.3).

Fuzzy concept	**Parameters**
high data security	encryption: Elliptic Curve
	min. key length: 384 bit
	redundancy: 5 copies
	HMAC: SHA-512
	hash key length: 512 bit

Figure 6.3: Example of fuzzy specification of data security concept

Fuzzy parameters and fuzzy concepts can then be transformed in a format that the SLA Management module can process in a homogeneous way with other crisp requirements, to take all of them into account in a comprehensive resource allocation strategy and optimization. Thus the service customers can define their requests using natural language since the fuzzy concepts can be described for the properties specified in fuzzy way.

As you can see from the example shown in Figure 6.3 this structure can be characterized as our definition of dependence on a service condition, this approach will be detailed in the next section.

6.4 FUZZY CONCEPTS AND FUZZY PARAMETERS ON THE PROVIDER SIDE

Once we identify the possible use of fuzzy logic in various stages in the SLA composition, we can easily infer that the rules used in the fuzzy logic can also be replicated in the rules established by the providers to determine their service conditions. For this we can define a fuzzy proposition as the combination of a fuzzy set representing a concept as a linguistic variable. A fuzzy proposition can be represented generally by the following construct (x is θ) where x is a linguistic variable, and θ is a fuzzy set.

Based on this premise we can define a fuzzy rule as a production that uses fuzzy propositions representing a concept. Each rule consists of an antecedent, representing a condition, and a consequent representing an action structured in terms of an association of fuzzy propositions. The basic structure of a fuzzy rule is as follows:

$$\text{IF } (u_1 \text{ is } \theta_1) \text{ AND } (u_2 \text{ is } \theta_2) \text{ AND ... AND } (u_n \text{ is } \theta_n) \text{ THEN } (y \text{ is } \theta)_N.$$

A rule is usually represented in the form R: ***if*** <complex> *then* <class> = C_i, where C_i is one of the possible class values, i.e. $C_i \in \{C_1, C_2, ..., C_n\}$ and <complex> is a disjunction of conjunctions of conditions for the attributes of the form X_i *op value*, with $X_i \in X$.

Any rule can be generally represented by $Body \rightarrow Head$ or, briefly, $B \rightarrow H$. Thus, it is said that an example E_i is covered by a rule R, if and only if the sample meets all the rule conditions. That is, an example E_i is covered by a rule R, if and only if B is true.

As an example we can present to the request of a property "Response Time" rules of the following type:

$$\text{IF } (x \text{ is Low}) \text{ AND } (y \text{ is Low}) \text{ THEN } (z \text{ is Low})$$

Where x is a variable representing the property "Request Rate", y a variable representing the property "Use of Resources" and z representing the property "Response Time". Meaning that if the "Request Rate" is low and "Use of Resources" is low then the provider can guarantee a low "Response Time".

Note that this rule is nothing more than the translation of our concept of dependence on a service condition by applying fuzzy parameters to determine the values of the properties, as can be seen in the following example:

$$d_1\text{: } (response\_time = \text{Low}) \rightsquigarrow (req\_rate = \text{Low}) \wedge (use\_resources = \text{Low})$$

It is interesting to note that in this example we present the dependence of internal properties to the provider (Use of Resources) and also demonstrate the shared liability using the dependence of an external property to the provider (Request Rate).

The definition of what is "High" or "Low" is given by the functions of relevance of each set, which is given for each type of linguistic variable. The connective THEN used

in the description of fuzzy rules corresponds to the fuzzy implication operator. Thus the inference procedure processes the fuzzy input data, along with the rules, in order to infer fuzzy control actions by applying the fuzzy implication operator and the rules of inference of fuzzy logic. The knowledge base consists of a rule base, characterizing the control strategy and goals of each provider. The membership functions define the commonalities of the universes of discourse, the fuzzy partitions and forms of input and output spaces. The advantage of using fuzzy rules is to be close to the form of presentation of human language.

As the ontology used in our framework establishes a control for determining SLAs we can say that this structure is configured as a control system. For certain information to be used in a control system, it is necessary to take the result of the inference of the various rules (which are fuzzy sets) and turn it into numerical values corresponding to the control signals associated with the linguistic variables used in the inferred propositions.

In conventional process control, assumptions and simplifications are often used to construct a mathematical model that can be far from real phenomenon. In contrast, the fuzzy control is capable of operating in complex processes in which knowledge is restricted and the mathematical models are not available.

The design of a fuzzy system consists in defining the set of terms used for the input and output variables, their respective membership functions and a database of rules representing the expert knowledge of the system. The design of a system using fuzzy logic involves the following steps:

1. Characterization of the range of values that the input and output variables can assume;

2. Definition of a set of functions, called membership functions, which map the input and output variables in the interval [0,1]. These functions take "labels" looking verbally translate some meaning to the modeled physical phenomenon (linguistic variables);

3. Definition of a set of rules by using logical operators, seeking to establish a relationship between the input and output values;

4. Once defined the rules derived from a symbolic language and with very intuitive meaning to the designer, going to the mathematical translation stage built symbolic language. This is conducted by using logical operators defined by the theory of fuzzy sets. This task is divided into three sub-stages: the first transforms the actual values of the input variables in degrees of relevance to a particular set (*fuzzyfication*); the second operates with the rules, "labels" and the result of *fuzzyfication* phase generating a set of fuzzy variables through the inference engine; the third and final sub-step transforms the results of the inference in a numerical output (*defuzzyfication*).

It should be noted that if a problem is not well understood and can not be mathematically represented, but has a good practical understanding, a system with fuzzy logic

can be used successfully. The utilization of fuzzy logic is inherently non-linear, unlike other methods, such as neural networks, it is characterized by easy to determine the action that should be taken for a given situation as it presents analytical structures. The rule base is constructed by general observation and knowledge of the problem, being simple to design.

Based on the assumptions presented we can prove that use of fuzzy logic in our framework is feasible, through the example shown in the next section.

6.5 APPLICATION OF FUZZY LOGIC IN THE FRAMEWORK PROPOSED

As seen in previous chapters each property has its value assigned by the provider according to the conditions in which they are available. On the other hand each property has the value requested by the user according to their needs. This approach allows the establishment of a model that considers aspects recognized by both parties through the service conditions and the customer requests.

Summarising the given definitions, we have that, in the provider side, a "service condition" can be described as follows:

Cond = $\alpha \mid \alpha = (p\ op\ v)$ where $p \in P$, *op* is an operator and v is the value available. As example:

Cond = (Availability $\geqslant$ 99.95%), (Location = USA), (Number of Replicas $\leqslant$ 5), (Storage Capacity $\leqslant$ 2 TB)

On the user side we can use the same description to describe a request for services, as follows:

Req = $\beta \mid \beta = (p\ op\ v)$. As example:

Req = (Availability $\geqslant$ 99.9%), (Location = USA), (Number of Replicas = 5), (Storage Capacity $\leqslant$ 1 TB)

So, we can say that a SLA is a merging of the customer request and the service condition (*Req* = *Cond*) or, as definition: SLA = $\forall\alpha \in Req \mid \exists\beta \in Cond$

As the results of both expressions (α and β) can present a range of values this range can be represented by fuzzy specifications. The use of fuzzy logic allows the framework to treat information of a non-numerical way. These fuzzy inputs (the customer request and the properties dependencies) are used in the SLA Management module to determine whether the SLA can be established (or not) based in a fuzzy output which is determined by the result of service condition.

Usually, the determining of a SLA can support numerical requirements but in our case we can also specify the treatment of linguistic requirements, which can be displayed in two ways: fuzzy parameters and fuzzy concepts. We can also assign to properties some fuzzy parameters which include all features that these resources can take. And we can assign to the services the fuzzy concepts that comprise all necessary resources for the implementation of service.

According to this specification, we can use fuzzy values in the service request from the customers as follow:

Req = (Storage Capacity = Medium), (Location = USA), (Number of Replicas = High), (Key Length = Long), (Security = High)

The behavior presented by fuzzy logic has great similarities to human form to process the information and is not Boolean bringing inferences and approximations. Thus, we can determine that the service conditions can be presented in the form of rules "if-then". For this to happen it is necessary that all possible properties are mapped in the same way, for this the provider can use a table of properties as shown in Table 6.1.

Table 6.1: Dependence of properties for fuzzy concepts

Property	**Dependence**
Storage	Storage Time Storage Capacity Location
Performance	Processing Time
Security	Key Length Hash Key Length
Availability	Redundancy

In the representation shown in the Table 6.1 we can describe the first column as the request of the customer and the second column as the necessary properties (dependencies) to determine the fuzzy rules by provider. Both levels can present fuzzy scales like: Very Low, Low, Medium, High, Very High, Big or Very Big, depending on the nature and the field of property.

The process consists in the fuzzy inputs are treated by the fuzzifier that describes the inputs by the linguistic variables. After this the fuzzy inference engine takes these inputs and applies the rules to give the results. Finally the defuzzification phase converts the fuzzy output set in a single number or a range of possible solutions.

The SLA Management module is used in our approach to verify the compatibility of the properties in the composition of a SLA. If the SLA is supported means that the provider can meet the customer requirements. The rules "if-then" based on service conditions by the provider can be represented like this:

Cond = (Storage = High), (Storage Time = Low), (Storage Capacity = Very High), (Location = USA)

This service condition gives us a rule like this:

- Dependencies - *if* Storage Time = Low *and* Storage Capacity = Very High *and* Location = USA

- Solution - *then* Storage = High

If the condition is not satisfied, the rule returns an empty result. In this way the weighted requirements, with the properties required by the customer and their dependencies, shows all possible values to be submitted by properties: crisp and fuzzy requirements, fuzzy concepts and fuzzy parameters.

In addition, changes in the rule base, changes in the provisions of the membership functions and the allocation of earnings to the inputs and outputs of the fuzzy system are techniques that can be used to simulate the ability to perform services by providers.

The mapping of service conditions in fuzzy rules allows monitoring and prediction of cloud services beyond the possibility of dynamic allocation of resources. This approach will be demonstrated in the next Chapter.

6.6 CHAPTER SUMMARY

This Chapter introduced the use of fuzzy logic for determining values in properties in Cloud services illustrating how these values can be displayed by providers using our ontology and also how these values can be requested by customers keeping their freedom of choice. The presented approach proves that the used ontology allows flexibility and adaptation of values handled by different providers, guaranteeing freedom of choice by customers and maintaining the control to the stakeholders involved, since customers can request services according to their needs while that providers can establish the relations of internal and external dependencies for the properties.

7

DYNAMIC RESOURCES MANAGEMENT

In order to use the generic Ontology presented in Chapter 3, we create a conceptual framework that deals with the information presented. As seen in Chapter 4, this framework has three complementary modules: Property Management, SLA Management and vSLA Monitor. The first two modules were described with in the previous chapters (4 and 5), this Chapter presents the vSLA Monitor module that completes the framework, in addition to the chapters describing the framework we also present the utilization of fuzzy logic in Chapter 6. This approach is also used for monitoring and control of the values determined in the valid Service Level Agreements.

7.1 INTRODUCTION

The conceptual framework proposed in this thesis aims to support advanced management of Service Level Agreements in Cloud Computing. For this management to be carried out, it is necessary that information about services is collected and organized. In addition, the framework should ensure that agreements are not violated and also keep monitoring on the changing scenarios.

From a SLA management perspective, it appears that the use of services in Cloud has expanded to several areas. However, models of agreements only consider the requirements of each domain by setting specific elements to be used in each agreement individually. That is, there are no conceptual models that consider the abstraction of general characteristics to be instantiated and reused in different applications, including preserving portability across domains and also, they are not models that have service integration with contextual information.

Thus, this thesis inspects specific aspects presented in each type of service and their behavior in different settings, obtaining a conceptual modeling approach to Service Levels Agreements representing this information in the form of a generic ontology. With this approach we could manage this information separately allowing the creation

of a structure that can control each specificity without compromising the composition of agreements.

For that the proposed framework provides a module called vSLA Monitor that controls the agreements and enables the monitoring of the contexts where different properties are located. To be able to perform the Service Agreements monitoring, resource management has to be considered. This resource management, in the scope of our framework, must be generated from independent external modules based on business process models of each provider and built using our definition of properties.

7.1.1 CHAPTER OUTLINE

This Chapter shows how the framework presented in this work can be used to control the changes of properties values in the Service Level Agreements. In addition to describing how the dynamic behavior of certain services is handled by our approach. Also, it describes the parameters that must be monitored to ensure the stability of the agreement between the parties and also how this feature can be used for dynamic prediction and prospecting for new Cloud services. The main contribution of this chapter is the description of the vSLA Monitor module that integrates the proposed framework and how this module manages the dynamic characteristic presented by different properties allowing the advanced management of Service Level Agreements with a method for stochastic modeling of resource management. The chapter is finishes with an overview of our approach.

7.2 VSLA MONITOR MODULE

As seen in Chapter 4, our framework has three complementary modules: Property Management, SLA Management and vSLA Monitor. The vSLA Monitor module is responsible for controlling the SLAs and monitoring the change of scenarios and possible violations. This module consider the structure shown in Figure 7.1.

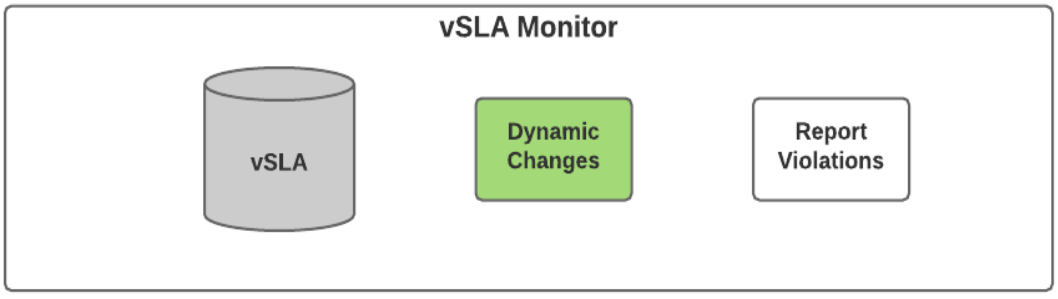


Figure 7.1: vSLA Monitor module

The vSLA Monitor module receives (from the SLA Management module) the valid Service Agreements that were previously accepted and agreed between customers and providers and maintains a repository containing the information inherent in each of these agreements, as mentioned in Chapter 5 and shown by example in Table 7.1.

Table 7.1: Information about valid SLAs in the vSLA repository

SLA ID	User ID	Provider ID	Service ID	Service Conditions
1521	00452	2542	12132	c_7, c_{11}, c_1, c_{17}
1678	01254	1367	11222	c_4; c_{12}; c_2; $c_{20} \rightsquigarrow c_{21}$
934	05363	6985	13410	c_4; c_{13}; c_{17}
15231	00896	1367	11219	c_4; c_{11}; c_2; c_{15}
123	00125	3674	13408	$c_{11} \rightsquigarrow c_{10}$; c_{18}

This repository contains the information for each SLA that has been established between the parties, which are: the SLA Identifier (SLA ID), the user identification (User ID), identification of the service provider (Provider ID), the service identifier (Service ID) and the properties that composes each service plan (Service Conditions). In addition to this repository, the vSLA Monitor has a component that monitors the changes that may occur in properties that have dynamic characteristics: this component is called Dynamic Changes.

The Dynamic Changes component receives information about the values of all properties contained in service agreements and considers monitoring tools that can be applied in different properties.

Since some properties may suffer dynamic alterations and these alterations can influence the agreements established the framework must have a component that reports the possible violations. Considering this situation, we analyze what are dynamic properties and how effectively the vSLA Monitor module performs this monitoring.

7.3 DYNAMIC PROPERTIES

As previously seen, services in Cloud computing typically rely on different resources to be realized. The expected performance for a particular service depends on how its resources are provisioned and used. The resource management defines: (1) what are the necessary resources for the execution of the service, (2) how many they are, (3) what are the job skill/requirements, and (4) how and what services they are accessed.

According to the definitions in Chapter 3, we can state that a resource may also be considered as a property. In this section we define a proposal to use the dynamic features of the property with information about managing resources. This proposal aims to facilitate the evaluation of services in Cloud considering the necessary tasks for the deployment of services.

The proposed method uses our previous settings to adapt a business process that considers the use of resources. This method is composed of:

1. a notation that allows the use of properties like tasks of services to be modeled;
2. a method to include in the models the quantitative information that can be drawn from those properties.

This approach is part of the proposed framework and allows our generic ontology to be used for monitoring and control of cloud resources. To describe better how our monitoring of dynamic properties works, we can divide it into two phases:

1. description of the available resources;
2. description of dynamic needs of resources.

In the Subsection 7.3.1, we define the notation to consider the resource management in the modeling of the business process. This notation, although simple, is detailed enough to capture much of the resource requirements that a SLA can have. The Subsection 7.3.2 addresses the problem of resource allocation from the perspective of stochastic modeling. In this section, we also define a method to model the management capabilities that can be used by providers as well as a simulation tool for new services.

7.3.1 DESCRIPTION OF THE AVAILABLE RESOURCES

In Chapter 3, we present a definition for the generalization of information on SLAs for Cloud computing. The model reflects the behavior of an instance of a business process, without considering the resource requirements and the degradation that the contention for resources causes to the performance of the requested service (especially when several instances of processes are executed in parallel). However, if we want a analysis that really approximates the expected values for the real-world applications the model should contain a resource management policy associated with the service considered. For this to happen, we must map the properties of each service to the resources needed for its execution. In the sequence, we formalize the notion of resource used in this work.

Definition 7.1 *(Resource). A Resource R is a tuple: R = {c, φ} , where c is a service condition that specifies the type of resource being modeled together with a value that determines the amount of resource that can be accessed, given an operator and a value in the form of (p op v) this variable can consider values expressed in crisp mode and in fuzzy mode; and φ is the strategy that determines how the task instances are assigned to the feature.*

Regarding the task assignments there are several possible ways of access. We can cite as an example the terminology commonly found in the Queueing theory [167]:

- *first-in-first-out (FIFO)*: first come, first out, the first to arrive will be the first to be attended by the resource;
- *last-in-first-out (LIFO)*: last come, first-out, where the last to arrive, will be first to be attended by the resource;
- *random choice*: a task instance will be randomly chosen among the others who are waiting for the access to the resource;
- *timesharing*: the resource processing time is divided equally between all parallel instances that need to access it;

- *priority systems*: the instances of tasks are selected to access the resource according to their level of priority. A priority system can be preemptive (when instances of requests of higher priority can interrupt the instances that are already using the feature) or non-preemptive (when the instances that are already using the resource can not be interrupted).

In the context proposed in this thesis, we present examples using only two of these modes: time sharing and priority systems.

Definition 7.2 *(Resource Set). A Resource Set RS of a service s, denoted by RS(s), is a set RS(s) = {R | R is a necessary resource for any task of the service s}.*

As an example for a Resource Set we can have:

RS(*s*) = ("server 1"=2; "Timesharing"), ("server 2"=1; "Timesharing"), ("support operator"=3; "Priority Systems")
This example of a Resource Set corresponds to a business process model for a given service *s* with three different types of resources, such that:

- There are two resources of type "server 1", each one has as access mode the timesharing;
- There is only one resource of type "server 2" which has as access mode the timesharing;
- There are 3 resources of type "support operator" who have as access mode the priority system.

Based on these definitions we can then describe the available resources and determine how the modeling of properties in resources can be used in business models contributing to the monitoring of dynamic properties.

7.3.2 DESCRIPTION OF DYNAMIC NEEDS OF RESOURCES

Once the vSLA Monitor module considers the monitoring of external properties to the framework (considering shared liability and resources dependencies of provider) we should describe the needs of monitored resources. For this, we make three assumptions about the utilization of resources considered in this work:

1. the resource requirements do not vary with time, i.e., the amount of work and/or the number of resources needed for each service does not vary with time;
2. the resource requirements are state-dependent, i.e., the necessary resources for each service should consider the state changes of the properties;
3. when a service requires more than one resource, it only will be executed when all the resources it requires are available.

Given these assumptions, the following definitions formalize our concept of resource requirements for services in Cloud:

Definition 7.3 *(Simple Resource Requirement (SRR)). A Simple Resource Requirement (SRR) of a service can be expressed as a simple service condition c, where c is the identifier of a type of resource necessary for the service considering a value that determines how many work units will be processed by the resource, i.e., c = {p op v}, this value considers information in crisp mode and fuzzy mode.*

Definition 7.4 *(Composed Resource Requirements (CRR)). The Composed Resource Requirements (CRR) of a given service* $\boldsymbol{a}$*, denoted by CRR(*$\boldsymbol{a}$*), can be described by an expression involving SRRs and two logical operators of composition:* $\wedge$ *(AND) or* $\vee$ *(OR). In an expression of CRR, the AND operator has a higher precedence than the OR operator and parentheses can be used to force the order of evaluation of operations.*

As an example, let us consider the following situation: A service $\boldsymbol{a}$ as the solicitation of a property "Response Time" and the following Composed Resource Requirements (CRR) for this service:

$$CRR(\boldsymbol{a}) = (\text{Request Rate=Low}) \wedge ((\text{server 1=2}) \vee (\text{server 2=2}))$$

The Composed Resource Requirements of the service $\boldsymbol{a}$ determine that this service requires 1 resource of type "Request Rate" which can consider a low rate of requisitions and 1 resource of type "server 1" or "server 2 " that should consider the execution of two work units.

If a task for a service requires more than one unit of the same resource type, then we can express this fact in its expression of CRR through different SRRs with the same type of resource, as shown in the following example:

$$\text{Consider the service } \boldsymbol{b}, \text{ with } CRR(\boldsymbol{b}) = (\text{"server 1"=4}) \wedge (\text{"server 1"=2})$$

The Resource Requirements of the service $\boldsymbol{b}$ determine that this service requires 2 features like "server 1": one to hold four work units and the other to perform two work units.

We chose to represent a requirement for x units of a resource R as x SRR to R (as shown in the previous example) because this approach allows the provider set as the service demanded of R will be divided between the x resource units. Thus, the resource requirements of the set of all the necessary tasks for the realization of a service can be translated using the same definitions used by the service conditions of our framework.

7.3.3 MAPPING OF RESOURCES AND PROPERTIES

A Service Level Agreement should consider the fact that some services depend on different resources to run. For this reason, before the execution of a service, it is necessary that their resources are assigned. In this thesis we do not directly deal with the allocation of resources but we provide mechanisms through our framework for this to

be done, so that this allocation can also allow monitoring of the agreements reached. Once we allow providers to express the values of their properties also in fuzzy way, we can use this approach also to monitor the necessary resources for this. To illustrate this situation we analyze the mapping between properties and resources shown in Table 7.2:

Table 7.2: Mapping between Properties and Resources

List of Properties P	**Resource Set RS**
Storage Capacity	• ("Server 1"=2; "Timesharing") • ("Server 2"=1; "Timesharing") • ("HD"=100TB; "Timesharing") • ("Location"=USA; "Timesharing")
Response Time	• ("Request Rate"=Low; "FIFO") • ("Resource Usage"=Medium; "Timesharing")
Location	• ("Location"=Italy; "Timesharing") • ("Location"=France; "Timesharing") • ("Location"=Brazil; "Timesharing") • ("Location"=USA; "Timesharing")
Security	• ("Key Length"=High; "Random Choice") • ("Hash Key Length"=High; "Random Choice")
Storage	• ("Storage Time"=High; "Timesharing") • ("Storage Capacity"=Very High; "Timesharing")
Resource Usage	• ("Server 1"=10; "Timesharing") • ("Server 2"=10; "Timesharing")

The example shown in Table 7.2 illustrates the list of properties displayed by a provider for the provision of its services and the necessary resources so that these services can be realized. Considering these information, we can determine the service conditions of which are presented to the framework and the resource requirements observed by the provider (Table 7.3).

Table 7.3: Mapping between Service Conditions and Resource Requirements

Service	Service Condition	Resource Requirements
a: c_1: *response_time*=Low	$c_1 \rightsquigarrow c_7 \wedge c_{17}$	$CRR(\boldsymbol{a})$ = (Request Rate=Low) $\wedge$ (Resource Usage=Low)
b: c_2: *storage*$\leqslant$10TB	$c_2 \rightsquigarrow c_8 \wedge c_{10}$	$CRR(\boldsymbol{b})$ = (Storage Time=Low) $\wedge$ (Storage Capacity=10TB)
c: c_3: *security*=High	$c_3 \rightsquigarrow c_{20} \wedge c_{25}$	$CRR(\boldsymbol{c})$ = (Key Length=Very Long) $\wedge$ (Hash Key Length=512bit)

Note that, in both Table 7.2 and Table 7.3, we have the property "Resource Usage" (c_{17}). Obviously, this property is not available as a service by the provider but serves to illustrate that the use of our approach can allow an effective control over resources provider since the provider can control the internal use of its resources considering the same rules scheme introduced by our approach. Disregarding this property into a real example of service "Response Time" could be made available as follows:

$$\text{service } a\text{: } c_1\text{: } \textit{response\_time}\text{=Low} \mid c_1 \rightsquigarrow c_7 \mid SRR(\boldsymbol{a}) = \text{(Request Rate=Low)}$$

We can see that for the service b, presented as a solicitation in Table 7.3, the provider needs to allocate an amount equivalent to the amount of solicitation otherwise could not meet this demand.

Furthermore, it is clear that both the service a and c shown in Table 7.3, the provider has the opportunity to present the same service considering fuzzy concepts, since our approach presented in Chapter 6 allows it.

Considering the assumptions presented in Subsection 7.3.1 we have:

- The resource requirements necessary to meet the service b exemplifying the first assumption, i.e., "the resource requirements do not vary with time" as the amount of resource available to meet this demand can not be changed. As noted in the example, to set up 10TB of storage, the provider must provide a storage capacity equal to 10TB, if this requirement changes with time, for example providing 8TB after a certain period of time, that characterize a violation of the agreement.

- Precisely for this feature, we consider the second assumption ("the resource requirements are state-dependent"), such as to meet the service a, the customer (having shared liability) must maintain a rate of requests equal to the values assigned to the fuzzy parameter "Low". Obviously this property undergoes state changes as it is possible the customer does not make any request keeping this rate equal to zero, the same way that the customer can perform requests with a request rate equal to "Medium" and even "High", featuring the dynamic nature of the property.

- As all services shown as an example in Table 7.3 we can say that the third assumption is also served by that "one service only will be executed when all the resources it requires are available."

Furthermore, we must consider that in the case of a timesharing resource exists the possibility that, at some time in the service execution, the number of instances

accessing the resource can be greater than the number of units of that resource. For this reason, the vSLA Monitor module should monitor the dynamic changes that occur during execution, i.e. to monitor the context in which property is located.

7.4 CONTEXT MONITORING

As seen in Chapter 5 the information about the changes of new values of property already established can serve as suggestion of new service agreements, since vSLAs are stored and the relevant properties are monitored. Using this same approach we can monitor the dynamic changes in property values in SLA already established (vSLA). To do this, we must consider the intrinsic knowledge in the service conditions determined to establish the SLA in question.

The main source of knowledge to build an algorithm of control and monitoring comes from the control experience of human operator, in our case the service provider. Given this premise setting up a monitoring system for dynamic values is a set of conditional (if - then), where the first part is called "antecedent" (conditions) and the second part, called "consequent" deals with an action (control) that has to be performed. Thus, in the same way that human strategy, fuzzy rule bases express how the control should be performed when a certain state of controlled value is observed, from the service provider's knowledge.

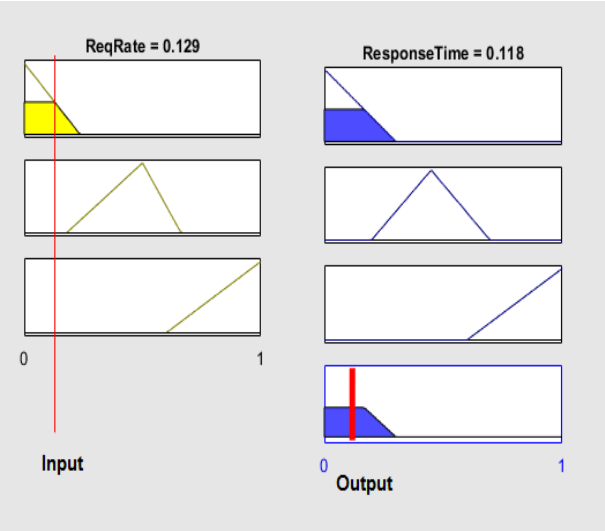
The construction of a fuzzy controller to be used in the control and monitoring of a service should consider the service behavior in response to change the value of some variable. This allows the monitoring of property values presented for each service agreement and allows the provider to qualitatively predict the results that can be displayed.

As an example, we consider the following service condition:

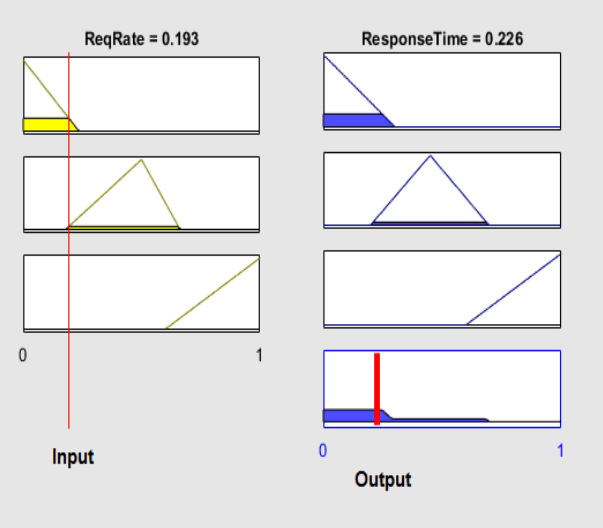
Given a service "Response Time"=Low which considers the following condition "Request Rate"=Low we have ($c_1 \rightsquigarrow c_7$)

Applying our approach we have as "antecedent" (conditions) the property c_7 (which is external and has shared liability) and as "consequent" (control) we have the property c_1.

Once in possession of the knowledge base formed by the provider and the set of generated rules, it is possible to distinguish which service conditions are being met through a Fuzzy Inference System that observes entries for each vSLA, as shown in Figures 7.2(a) and 7.2(b).



(a)



(b)

Figure 7.2: Fuzzy Inference System with input c_7=0.129 like "Low" parameter (a) and Fuzzy Inference System with input c_7=0.193 like "Low" parameter (b)

As shown in Figure 7.2, the change of the input values for the property c_7 does not compromise the agreement reached because the monitored property values remains within the range set by the rule. Consider that in a given period of time the monitored values of this property have the following situation (Figure 7.3):

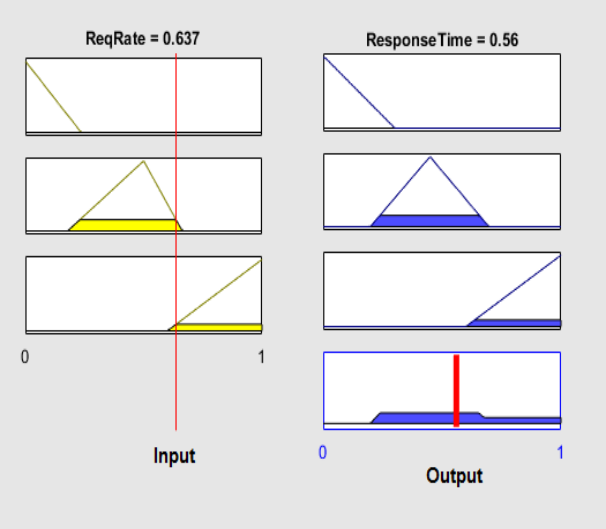


Figure 7.3: Fuzzy Inference System with input c_7=0.637 like "Medium" parameter

In the case shown in Figure 7.3, the request rate presented by the customer is greater than the value permitted in this agreement, configured as a violation of the SLA.

In this sense we can apply this approach to monitor the different contexts presented for different vSLA according to the example shown in Table 7.4.

Table 7.4: Different Contexts in the vSLA repository

SLA ID	User ID	Provider ID	Service ID	Service Conditions
1522	00452	6985	12135	$c_1 \rightsquigarrow c_7$
1679	00896	6985	12135	$c_1 \rightsquigarrow c_7$

Note that the same service condition is presented in a service established in two vSLAs for two different customers. Each property is then monitored according to the context presented to each customer, that is, the values presented by each customer should be related to their respective agreements, exemplifying:

The USER 00452 have a "Request Rate"=0.193 so the SLA 1522 is OK but the USER 00896 have a "Request Rate"=0.637 causing the SLA 1679 is violated. This violation can then be informed by the Report Violations component that is part of the vSLA Monitor module. It is important to mention that to make a correct monitoring of the property values, sometimes, it can be necessary to use external monitoring tools, since many properties can belong to external environments to the framework.

In addition to allowing monitoring of dynamic changes and report possible violations, the approach presented to the vSLA Monitor module can also be used for simulation services for providers.

7.5 PREDICTION SYSTEM

As seen throughout this chapter, monitoring the use of resources is of utmost importance for the service provider. Also, predicting how many resources will be necessary

to perform a particular service helps to better determine the allocation of these resources. Therefore, a demand predictor can exert direct influence on the mesh of a provider resources and, by consequence, the strategy of control and provision of new services. Thus, a service provider, before a request for new services, can determine the necessary resources to meet the new demand and predict how these resources will be necessary to provide the new service. Thus, the predictors developed by each provider can simulate to meet the new demands. This approach also allows the prediction of violations of agreements before it occurs.

A prediction system based on our approach can consider the logic fuzzy demonstrated in Chapter 6, where the inputs are the resources needed to perform the service and the output will be the demand service capacity. For example, consider the following service condition used by the provider (considering its internal resources to meet a service):

$$c_1 \rightsquigarrow c_7 \wedge c_{17}, \text{ where } CRR(\boldsymbol{a}) = \text{(Request Rate=Low)} \wedge \text{(Resource Usage=Low)}$$

Considering the set of rules established by the provider to meet the property c_1 we can have a situation as shown by Figure 7.4:

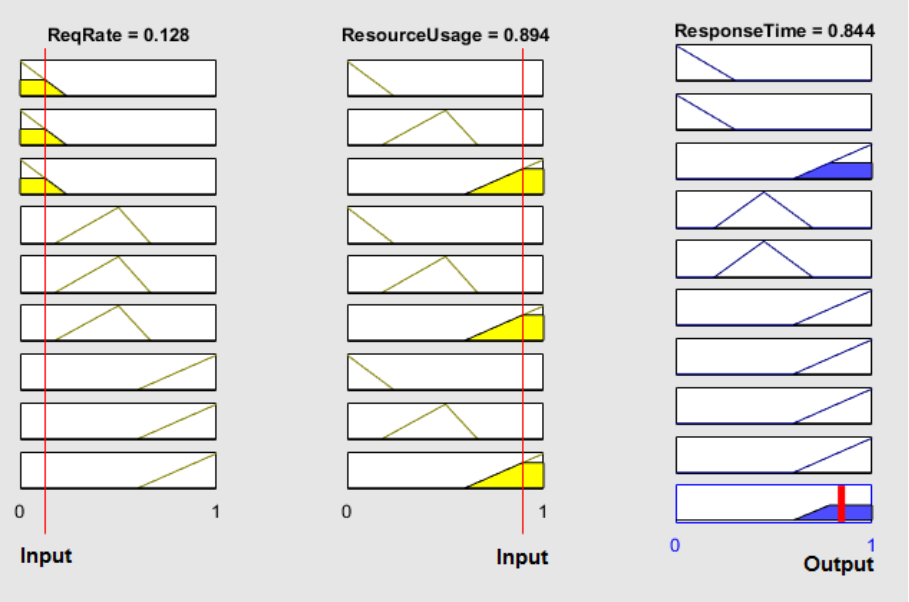


Figure 7.4: Fuzzy Inference System to Simulate the Resource Usage

In the example shown in Figure 7.4 the provider simulates the input value for the property "Resource Usage" by setting it as fuzzy parameter "High" while keeping the value for the "Request Rate" property to "Low". Obviously this situation presents a result in which the provider can not meet the SLA established. As the property "Resource Usage" may be based on the resources available by the provider, such as number of servers, the provider can apply the same approach to create rules that simulate these features. Thus, the provider can predict the amount of resource that needs to be allocated to meet specific demand.

7.6 APPROACH OVERVIEW

Once the vSLA Monitor module completes the proposed framework, we can then present an overview of our approach. In this section we present a simplified structure of the proposed framework showing all the features described in previous chapters, thereby enabling a complete understanding of our approach. For this we divided our sample into four distinct and complementary parts, as shown in Figure 7.5.

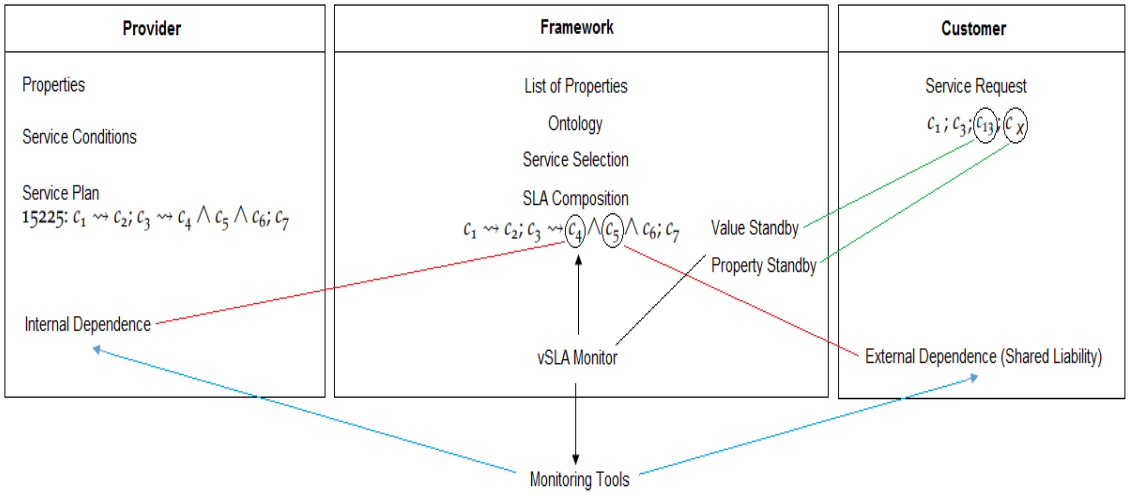


Figure 7.5: Approach Overview

The features reported in Figure 7.5 can be described according to the following sequence:

1. First, the service provider, using our XML Schema, offers its property list containing all services that it might offer, containing values both in crisp mode as in fuzzy mode, including the conditions for each service can be executed, and consequently their own service plans.

2. Then, the framework maintains a repository containing a list of all services provided by different providers and organizes in a generic ontology containing the established properties and their values.

3. Based on this information, the customer can consult the services provided and submit his service request that can contain even the solicitation of new values and/or new services (properties).

4. The framework performs the search for the services that match the customer's request and presents the service selection.

5. Based on customer choice, the framework presents the SLA composition and stores the valid SLA (vSLA) which is monitored by vSLA Monitor module.

6. Considering the customer's freedom to order services according to their needs, if a request contains the solicitation of new values and/or new properties, this information will be placed on standby and controlled by Properties Inventory component.

7. The information related to new services and new values demanded by customers can be accessed by providers to establish services that can meet this new market demand.

8. Using our approach to service conditions and properties dependence, the provider can simulate their resource allocation to meet new demands.

9. When new values and new properties are presented by the provider, the Property Inventory component can refer to the valid SLA and report these situations to customers.

10. To ensure the performance of services, the provider may submit plans that may contain the dependence of internal properties to its own structure and also external properties, featuring thus shared liability for the service.

11. Considering our concept of property dependences the vSLA Monitor module can be combined with external monitoring tools to signal violations in valid SLAs.

Finally, we conclude the description of our approach to Advanced SLA Management in Cloud computing, also reporting some show cases as examples.

7.7 CHAPTER SUMMARY

In this chapter, we present the vSLA Monitor module used in our conceptual framework for monitoring and control of the properties established in each Service Level Agreement. This module uses information from monitoring tools (which may be external) to verify if the dynamic changes that occur in property values do not change what was established in the agreement, thus enabling monitoring of violations that can be perceived both by customers service as the providers. This chapter also shows how our approach can be used to predict the use of resources and the prospecting of new cloud services. In addition to presenting concepts of dynamic properties the chapter also presents a description of dynamic needs for Cloud resources and how these resources can be made available by the providers through our approach. Concluding the Chapter

we present the overview of the proposed approach for Advanced SLA Management in Cloud Computing.

8

CONCLUSION AND FUTURE WORKS

In this thesis, we define a new approach to advanced management of Service Level Agreements in Cloud Computing. This approach consists of the rating of a generic ontology that allows specification of services, attributes and resources in the form of properties and this definition is then used in a conceptual framework that allows the management of these properties. We address the problem of transparency and control of information in Service Level Agreements and verify sensitive information presented in different scenarios of selection and SLAs composition in Cloud Computing. After some introductory remarks and a discussion of related works, we focused on three specific issues: the greater flexibility in SLA trading by defining of an Advanced SLA in Cloud Computing, the support for future markets in Cloud Computing, and the description of a set of adaptive methods for Dynamic SLA Management. In this chapter, we summarize the contributions of this thesis, and outlined possible directions for future works, concluding with our final remarks.

8.1 SUMMARY OF THE CONTRIBUTIONS

The contribution of this thesis is threefold.

Greater flexibility in SLA trading.
We proposed our solution to enable the application of different services by customers guaranteeing freedom to request services according to their needs in addition to provide the effective application of the concept of XaaS. Our approach is based on the definition of concepts that considers each and every service attribute or resource in Cloud environment as properties with the same importance, it enables the the model requirements to make each service and observe the dependencies that may exist between different properties. We provide a generic ontology to support the specific characteristics of each property, allowing greater flexibility in negotiating SLAs and present the

modeling of the problem in a generic way considering the freedom of customer choice and shared liability present in some dependencies. The efficiency of our approach is based on the use of XML schemas that are used to facilitate the access to information and the SLA models presented by each service provider.

Support for future markets in Cloud Computing.
We provide our solution to capture and store information related to new market demands considering properties and values already established by the service providers and use our generic ontology to enable the discovery of new customer needs. For this we describe the structure of our conceptual framework that allows the application of our ontology to describe the features presented for each situation. Our solution considers the monitoring of information in order to provide important data for the Advanced SLA Management allowing flexibility and adaptation of services.

Description of a set of adaptive methods for Dynamic SLA Management.
We define an information control solution in Cloud SLA that allows the use of different methods already used in Cloud Computing, such as external certification services and the use of fuzzy logic. In our approach, we deal with dynamic changes present in values of different properties as an information base for control of services. With this, we can use control rules to ensure both the simulation of new market demands as well as to simulate the allocation of resources by providers. Our approach also allows the application of techniques of text mining and clustering information to enable service providers to the identification of these new markets. Thus, our proposal allows the use of different methods and techniques already consolidated in Cloud environment and guarantees the freedom choice for customers and a more effective control by providers.

8.2 FUTURE WORKS

To continue the work of this PhD thesis, we have two suggestions. The first one can be developed in the short term, while the second requires further research efforts. The proposals are described below.

Adaptation of the necessary algorithms in BPMN (Business Process Model and Notation).
For the algorithms presented in Chapter 5, and the use of fuzzy logic presented in Chapters 6 and 7 considered as valid entries the properties presented in each service condition, although already describe, in a way the business processes, it would be interesting to extend these algorithms to process diagrams containing other objects or even other types of BPMN models like collaboration diagrams.

For example, in process diagrams and collaboration diagrams of BPMN, there are several objects that can be activated with the occurrence of an external event to the business process (such as alterations of values in certain properties). If we can associate a rate of occurrence of these external events, then we can also model these objects.

When these external events are generated by business processes whose model is known, it can, in some cases, estimate the rate of generation of events through the performance of this model analysis.

Motivated by this notion of decomposition, our proposal is to decompose a business process model even before it is established as a service. Thus, the semantics of the constructs used to model the flow control business processes could help in the development of a new method of specific breakdown for each application domain.

Framework implementation with the provision of different services and methods. As seen in the examples, the monitoring of properties is a subject related to the analysis of subarea business processes and can be performed by external tools. The objective of this monitoring is to check the behavior of certain properties and establish the consequences that alterations in their value can result in Service Level Agreements. In addition, the monitoring methods should consider the dependencies presented in each service condition.

Motivated by the possibility of creating different techniques of monitoring enabled by the generalization concept presented, we propose the implementation of the framework to support the integration of new control methods looking for the scope of new services. Thus, the proposed framework can serve as a basis for testing for research on new services in the cloud, considering the aspects of the consolidation of Service Level Agreements.

8.3 CLOSING REMARKS

The main result of the study presented in this thesis was to demonstrate that the generalization of information in Cloud Computing can enable a multitude of composition of services. Since the entire information regarding services available in Cloud can be presented in the same way, it allows different aspects of services, resources and attributes to be combined to form new services. This result corroborates the main purpose of this thesis: Allow freedom of choice of services for customers while maintaining control over the agreements reached. It is also shown that the combination of different techniques in a conceptual framework allows the approach presented to be implemented to a model that effectively supports the concept of XaaS (everything as a service).

[1] ISO/IEC, "ISO/IEC 15408-1:2009 - evaluation criteria for it security," ISO/IEC - International Standartization Organization/International Engineering Consortium, Tech. Rep., 2009. *(Cited on page 1)*

[2] K. Jeffery and B. Neidecker-Lutz, *The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010*, K. E. Jeffery and B. S. R. Neidecker-Lutz, Eds. European Commission - Information Society and Media, 2010. *(Cited on page 1)*

[3] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: A system-level perspective," *IEEE Systems Journal*, vol. 7, no. 2, pp. 288–297, 2013. *(Cited on pages 1, 2, and 29)*

[4] A. Van der Wees, D. Catteddu, J. Luna, M. Edwards, N. Schifano, M. Scoca Lucia, and S. Tagliabue, "Cloud Service Level Agreement Standardisation Guidelines," Cloud Select Industry Group - Subgroup on Service Level Agreement (C-SIG-SLA), Tech. Rep., 2014. *(Cited on pages 2 and 62)*

[5] "Amazon CloudWatch." [Online]. Available: http://aws.amazon.com/cloudwatch/?nc2=h_ls *(Cited on pages 2 and 13)*

[6] V. R. Wadhe and V. A. Bharadi, "Review on Existing Cloud Platforms," *International Journal of Applied Information Systems*, vol. 6, no. 8, pp. 21–26, feb 2014. *(Cited on pages 2 and 16)*

[7] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing: State-of-the-Art and Research Challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, apr 2010. *(Cited on page 2)*

[8] P. Samarati and S. De Capitani di Vimercati, "Cloud security: Issues and concerns," in *Encyclopedia on Cloud Computing*, S. Murugesan and I. Bojanova, Eds. Wiley, 2016. *(Cited on page 3)*

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009. *(Cited on page 3)*

[10] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," in *ICA3PP'10 Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing*, 2010, pp. 13–31. *(Cited on page 5)*

[11] R. Buyya, J. Broberg, and A. Goscinski, *Cloud Computing: Principles and Paradigms*. Wiley, 2011. *(Cited on page 11)*

[12] B. Halpert, *Auditing Cloud Computing: a security and privacy guide*. Wiley, 2011. *(Cited on page 11)*

[13] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud Computing Pricing Models: A Survey," *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013. *(Cited on page 12)*

[14] W. Iqbal, M. Dailey, and D. Carrera, "SLA-Driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud," in *CloudCom '09 Proceedings of the 1st International Conference on Cloud Computing*, 2009, pp. 243–253. *(Cited on page 12)*

[15] P. Mell and T. Grance, "The nist definition of cloud computing," 2011. *(Cited on page 12)*

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," University of California at Berkeley, Tech. Rep. 2, mar 2009. *(Cited on pages 12 and 29)*

[17] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy, survey, and issues of cloud computing ecosystems," in *Cloud Computing*. Springer, 2010, pp. 21–46. *(Cited on page 12)*

[18] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the cloud landscape," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society, 2009, pp. 23–31. *(Cited on page 12)*

[19] B. Sosinsky, *Cloud computing bible*. John Wiley & Sons, 2010, vol. 762. *(Cited on page 13)*

[20] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends," *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, pp. 621–628, 2015. *(Cited on page 13)*

[21] "Google App Engine (GAE)." [Online]. Available: https://cloud.google.com/appengine/ *(Cited on page 13)*

[22] "Microsoft Windows Azure." [Online]. Available: https://azure.microsoft.com *(Cited on page 13)*

[23] S. Ried, H. Kisker, P. Matzke, A. Bartels, and M. Lisserman, "Sizing the cloud, understanding and quantifying the future of cloud computing," *Forrester Research, Inc*, vol. 21, 2011. *(Cited on page 13)*

[24] K. Breitman, M. Endler, R. Pereira, and M. Azambuja, "When tv dies, will it go to the cloud?" *Computer*, vol. 43, no. 4, pp. 81–83, 2010. *(Cited on page 13)*

[25] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *INFOCOM'10 Proceedings of the 29th conference on Information communications*. IEEE, mar 2010, pp. 1–9. *(Cited on page 13)*

[26] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro, "A Security Analysis of Amazon's Elastic Compute Cloud Service," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12*. ACM Press, 2012, p. 1427. *(Cited on page 13)*

[27] T. Redkar, T. Guidici, and T. Meister, *Windows Azure Platform*. Springer, 2011, vol. 1. *(Cited on page 13)*

[28] C. Severance, *Using Google App Engine*. " O'Reilly Media, Inc.", 2009. *(Cited on page 14)*

[29] M. Abualkibash and K. Elleithy, "Cloud Computing: The Future of IT Industry," *International Journal of Distributed and Parallel systems*, vol. 3, no. 4, pp. 1–12, jul 2012. *(Cited on page 14)*

[30] P. Sempolinski and D. Thain, "A comparison and critique of eucalyptus, opennebula and nimbus," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. Ieee, 2010, pp. 417–426. *(Cited on pages 14 and 16)*

[31] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 2009, pp. 124–131. *(Cited on pages 14 and 16)*

[32] N. Sadashiv and S. M. D. Kumar, "Cluster, Grid and Cloud Computing: A Detailed Comparison," *2011 6th International Conference on Computer Science Education (ICCSE)*, no. Iccse, pp. 477–482, aug 2011. *(Cited on page 14)*

[33] J. S. Ward and A. Barker, "Observing the Clouds: a survey and taxonomy of cloud monitoring," *Journal of Cloud Computing*, vol. 3, no. 1, p. 24, 2014. *(Cited on page 14)*

[34] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008. *(Cited on page 15)*

[35] A. Undheim, A. Chilwan, and P. Heegaard, "Differentiated availability in cloud computing slas," in *2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE, 2011, pp. 129–136. *(Cited on page 15)*

[36] A. Arenas and M. Wilson, "Contracts as trust substitutes in collaborative business," *Computer*, no. 7, pp. 80–83, 2008. *(Cited on page 15)*

[37] M. Alhamad, T. Dillon, and E. Chang, "Conceptual SLA Framework for Cloud Computing," in *4th IEEE International Conference on Digital Ecosystems and Technologies*. IEEE, 2010, pp. 606–610. *(Cited on pages XIII, 15, and 30)*

[38] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, no. 1, pp. 57–81, 2003. *(Cited on pages 15 and 17)*

[39] M. Smit and E. Stroulia, "Maintaining and evolving service level agreements: Motivation and case study," in *2011 International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*. IEEE, 2011, pp. 1–9. *(Cited on page 15)*

[40] L. Wu and R. Buyya, "Service Level Agreement (SLA) in Utility Computing Systems," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, 2011, ch. 1, pp. 1–25. *(Cited on page 16)*

[41] L. Ye, H. Zhang, J. Shi, and X. Du, "Verifying Cloud Service Level Agreement," in *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE, dec 2012, pp. 777–782. *(Cited on page 16)*

[42] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "Towards QoS-Oriented SLA Guarantees for Online Cloud Services," *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp. 50–57, may 2013. *(Cited on page 16)*

[43] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification," IBM Corporation, Tech. Rep., 2002. *(Cited on page 17)*

[44] A. Andrieux, K. Czajkowski, K. Keahey, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification (WS-Agreement)," in *Global Grid Forum GRAAP-WG*, vol. 192, 2006, pp. 1–80. *(Cited on page 17)*

[45] A. Pichot, P. Wieder, O. Wäldrich, and W. Ziegler, "Dynamic SLA-Negotiation Based on WS-Agreement," Tech. Rep., 2007. *(Cited on page 17)*

[46] A. Maarouf, A. Marzouk, and A. Haqiq, "A Review of SLA Specification Languages in the Cloud Computing," *10th International Conference on Intelligent Systems: Theories and Applications (SITA), 2015*, pp. 1–6, 2015. *(Cited on pages XIII, 17, 18, and 27)*

[47] R. B. Uriarte, F. Tiezzi, and R. De Nicola, "SLAC: A Formal Service-Level-Agreement Language for Cloud Computing," *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, pp. 419–426, 2014. *(Cited on page 17)*

[48] K. T. Kearney, F. Torelli, and C. Kotsokalis, "SLA*: An Abstract Syntax for Service Level Agreements," *2010 11th IEEE/ACM International Conference on Grid Computing*, pp. 217–224, oct 2010. *(Cited on page 17)*

[49] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A Language for Defining Service Level Agreements," in *Proceedings of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems-FTDCS*. *(Cited on page 17)*

[50] V. Tosic, K. Patel, and B. Pagurek, "WSOL - Web Service Offerings Language," in *Lecture Notes in Computer Science - CAiSE 2002 and WES 2002*, 2512nd ed., C. Bussler, S. McIlraith, M. Orlowska, B. Pernici, and J. Yang, Eds. Springer, 2002, vol. 2512, pp. 57–67. *(Cited on page 17)*

[51] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low Level Metrics to High Level SLAs - LoM2HiS Framework: Bridging the Gap Between Monitored Metrics and SLA Parameters in Cloud Environments," in *2010 International Conference on High Performance Computing Simulation*. IEEE, jun 2010, pp. 48–54. *(Cited on pages XI, 18, 19, and 25)*

[52] V. C. Emeakaroha, R. N. Calheiros, M. a. S. Netto, I. Brandic, and C. De Rose, "DeSVi: An Architecture for Detecting SLA Violations in Cloud Computing Infrastructures," *Proceedings of the 2nd International ICST Conference on Cloud Computing (CloudComp 2010)*, 2010. *(Cited on pages XI, 18, 20, 21, and 25)*

[53] A. Kertesz, G. Kecskemeti, and I. Brandic, "An SLA-based Resource Virtualization Approach for On-demand Service Provision," *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pp. 27–34, 2009. *(Cited on pages XI, 18, 22, and 25)*

[54] O. Niehörster, A. Brinkmann, G. Fels, J. Krüger, and J. Simon, "Enforcing SLAs in Scientific Clouds," *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, pp. 178–187, 2010. *(Cited on pages XI, 18, 23, and 25)*

[55] V. C. Emeakaroha, T. C. Ferreto, M. A. S. Netto, I. Brandic, and C. A. F. De Rose, "CASViD: Application Level Monitoring for SLA Violation Detection in Clouds," *2012 IEEE 36th Annual Computer Software and Applications Conference*, pp. 499–508, jul 2012. *(Cited on page 18)*

[56] M. A. T. Rojas, N. M. Gonzalez, V. Fernando, F. F. Redígolo, and T. Carvalho, "A Framework to Orchestrate Security SLA Lifecycle in Cloud Computing," in *CISTI'2016 - 11ª Conferencia Ibérica de Sistemas y Tecnologías de Información*, 2016, pp. 414–420. *(Cited on page 18)*

[57] A. Chazalet, "Service Level Checking in the Cloud Computing Context," *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 297–304, jul 2010. *(Cited on page 26)*

[58] S. Cimato, E. Damiani, F. Zavatarelli, and R. Menicocci, "Towards the Certification of Cloud Services," *Proceedings - 2013 IEEE 9th World Congress on Services, SERVICES 2013*, pp. 92–97, 2013. *(Cited on page 26)*

[59] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud Monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, jun 2013. *(Cited on pages 26 and 27)*

[60] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with lattice," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*. IEEE, 2010, pp. 239–246. *(Cited on page 26)*

[61] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf, "A flexible architecture integrating monitoring and analytics for managing large-scale data centers," in *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 2011, pp. 141–150. *(Cited on page 26)*

[62] R. Mian, P. Martin, and J. L. Vazquez-Poletti, "Provisioning data analytic workloads in a cloud," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1452–1458, 2013. *(Cited on page 26)*

[63] P. Hasselmeyer and N. d'Heureuse, "Towards holistic multi-tenant monitoring for virtual data centers," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*. IEEE, 2010, pp. 350–356. *(Cited on page 27)*

[64] G. Katsaros, R. Kübert, and G. Gallizo, "Building a service-oriented monitoring framework with rest and nagios," in *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 426–431. *(Cited on page 27)*

[65] J.-C. Laprie, "From dependability to resilience," in *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*. Citeseer, 2008, pp. G8–G9. *(Cited on page 27)*

[66] R. W. Shirey, "Internet security glossary, version 2," 2007. *(Cited on page 27)*

[67] E. Amazon, "Cloudwatch," 2014. *(Cited on page 27)*

[68] "AzureWatch." [Online]. Available: https://www.paraleap.com/AzureWatch *(Cited on page 27)*

[69] "Monitis." [Online]. Available: http://www.monitis.com/ *(Cited on page 27)*

[70] "Logicmonitor." [Online]. Available: http://ls.logicmonitor.com/monitoring/storage/netapp-filers/ *(Cited on page 27)*

[71] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka Platform and QoS-Driven Resource Provisioning for Elastic Applications on Hybrid Clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, jun 2012. *(Cited on page 27)*

[72] "GroundWork." [Online]. Available: http://www.gwos.com/features/ *(Cited on page 27)*

[73] "Nagios." [Online]. Available: https://www.nagios.org/ *(Cited on page 27)*

[74] "OpenNebula." [Online]. Available: http://opennebula.org/ *(Cited on page 27)*

[75] "Nimbus." [Online]. Available: http://www.nimbusproject.org/ *(Cited on page 27)*

[76] S. A. De Chaves, R. B. Uriarte, and C. B. Westphall, "Toward an Architecture for Monitoring Private Clouds." *(Cited on page 27)*

[77] A. Corradi, L. Foschini, J. Povedano-Molina, and J. M. Lopez-Soler, "Dds-enabled cloud management support for fast task offloading," in *Computers and Communications (ISCC), 2012 IEEE Symposium on*. IEEE, 2012, pp. 000 067–000 074. *(Cited on page 27)*

[78] "Hyperic-hq." [Online]. Available: https://sourceforge.net/projects/hyperic-hq/ *(Cited on page 27)*

[79] "Sensu." [Online]. Available: http://www.sonian.com/sensu/ *(Cited on page 27)*

[80] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A Survey of Cloud Monitoring Tools: Taxonomy, capabilities and objectives," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2918–2933, oct 2014. *(Cited on pages XI and 28)*

[81] "mOSAIC Project." [Online]. Available: http://www.mosaic-cloud.eu/ *(Cited on pages 28 and 36)*

[82] F. Moscato, R. Aversa, B. Di Martino, T.-F. Fortis, and V. Munteanu, "An Analysis of mOSAIC Ontology for Cloud Resources Annotation," *Computer Science and Information Systems (FedCSIS)*, pp. 973–980, 2011. *(Cited on pages 28, 37, and 39)*

[83] W. K. Hon, C. Millard, and I. Walden, "Negotiating cloud contracts-looking at clouds from both sides now," 2012. *(Cited on page 29)*

[84] D. Kyriazis, "Cloud computing service level agreements: exploitation of research results," *European Commission Directorate General Communications Networks Content and Technology Unit, Tech. Rep*, vol. 5, 2013. *(Cited on pages 29 and 40)*

[85] E. Feller, L. Rilling, and C. Morin, "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds," *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pp. 482–489, may 2012. *(Cited on page 29)*

[86] K. V. Vishwanath and N. Nagappan, "Characterizing Cloud Computing Hardware Reliability," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*. ACM Press, 2010, p. 193. *(Cited on page 29)*

[87] R. Jhawar and V. Piuri, "Fault Tolerance and Resilience in Cloud Computing Environments," in *Computer and Informaion Security Handbook*, 2nd ed., J. Vacca, Ed. Morgan Kaufmann, 2013, vol. 2. *(Cited on page 29)*

[88] L. S. Meyers, G. Gamst, and A. J. Guarino, *Applied multivariate research: Design and interpretation*. Sage, 2006. *(Cited on page 30)*

[89] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic*. Prentice hall New Jersey, 1995, vol. 4. *(Cited on pages 30 and 115)*

[90] S. Thomas *et al.*, "The analytic hierarchy process: planning, priority setting, resource allocation," *Pittsburgh PA: University of Pittsburgh*, 1980. *(Cited on page 30)*

[91] N. Mary and K. Jayapriya, "An Extensive Survey on QoS in Cloud Computing," *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 1–5, 2014. *(Cited on pages XIII, 30, and 31)*

[92] M. M. K. Saravanan and M. L. Kantham, "an enhanced qos architecture based framework for ranking of cloud services," *International Journal of Engineering Trends and Technology (IJETT)*, vol. 4, no. 4, pp. 1022–1031, 2013. *(Cited on page 31)*

[93] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 5, pp. 936–943, 2012. *(Cited on page 31)*

[94] B. Chitra, M. Sreekrishna, and A. Naveenkumar, "A survey on optimizing the qos during service level agreement in cloud," *International Journal of Emerging Technology and Advanced Engineering (ISSN 2250-2459, ISO 9001: 2008 Certified Journal, Volume 3, Issue 3*, 2013. *(Cited on page 31)*

[95] G. Feng, S. Garg, R. Buyya, and W. Li, "Revenue maximization using adaptive resource provisioning in cloud computing environments," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE Computer Society, 2012, pp. 192–200. *(Cited on page 31)*

[96] X. Liu, Y. Yang, D. Yuan, G. Zhang, W. Li, and D. Cao, "A generic qos framework for cloud workflow systems," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. IEEE, 2011, pp. 713–720. *(Cited on page 31)*

[97] Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang, "Qos ranking prediction for cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1213–1222, 2013. *(Cited on page 31)*

[98] S. K. Garg, S. Versteeg, and R. Buyya, "Smicloud: A framework for comparing and ranking cloud services," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE, 2011, pp. 210–218. *(Cited on page 31)*

[99] R. Yu, X. Yang, J. Huang, Q. Duan, Y. Ma, and Y. Tanaka, "Qos-aware service selection in virtualization-based cloud computing," in *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*. IEEE, 2012, pp. 1–8. *(Cited on page 31)*

[100] L. Pan, "Towards a ramework for automated service negotiation in cloud computing," in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*. IEEE, 2011, pp. 364–367. *(Cited on page 31)*

[101] K. Xiong and H. Perros, "Service performance and analysis in cloud computing," in *2009 Congress on Services-I*. IEEE, 2009, pp. 693–700. *(Cited on page 31)*

[102] L. Wu, S. K. Garg, and R. Buyya, "Sla-based resource allocation for software as a service provider (saas) in cloud computing environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011, pp. 195–204. *(Cited on page 31)*

[103] V. C. Emeakaroha, I. Brandic, M. Maurer, and I. Breskovic, "Sla-aware application deployment and resource allocation in clouds," in *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*. IEEE, 2011, pp. 298–303. *(Cited on page 31)*

[104] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya, "Pricing cloud compute commodities: a novel financial economic model," in *Proceedings of the 2012 12th IEEE/ACM international symposium on cluster, cloud and grid computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 451–457. *(Cited on page 31)*

[105] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected areas in communications*, vol. 14, no. 7, pp. 1228–1234, 1996. *(Cited on page 31)*

[106] J. Huang, X. Huang, and Y. Ma, "An effective approximation scheme for multiconstrained quality-of-service routing," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–6. *(Cited on page 31)*

[107] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013. *(Cited on page 31)*

[108] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed systems*, vol. 22, no. 6, pp. 931–945, 2011. *(Cited on page 31)*

[109] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 1–14. *(Cited on page 31)*

[110] V. X. Tran, H. Tsuji, and R. Masuda, "A new qos ontology and its qos-based ranking algorithm for web services," *Simulation Modelling Practice and Theory*, vol. 17, no. 8, pp. 1378–1398, 2009. *(Cited on page 31)*

[111] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012. *(Cited on page 31)*

[112] M. Macías Lloret, J. O. Fitó, and J. Guitart Fernández, "Rule-based sla management for revenue maximisation in cloud computing markets," in *Proceedings of*

the 2010 International Conference on Network and Service Management. IEEE Computer Society Publications, 2010, pp. 354–357. *(Cited on page 31)*

[113] L. Wu, S. K. Garg, and R. Buyya, "Sla-based admission control for a software-as-a-service provider in cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280–1299, 2012. *(Cited on page 31)*

[114] C. A. Ardagna, R. Jhawar, and V. Piuri, "Dependability Certification of Services: A Model-Based Approach." *(Cited on page 31)*

[115] G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, and T. Varvarigou, "Dynamic, Behavioral-Based Estimation of Resource Provisioning Based on High-Level Application Terms in Cloud Platforms," *Future Generation Computer Systems*, vol. 32, no. 1, pp. 27–40, 2014. *(Cited on page 31)*

[116] F. a. Omara and M. M. Arafa, "Genetic Algorithms for Task Scheduling Problem," *Journal of Parallel and Distributed Computing*, vol. 70, no. 1, pp. 13–22, jan 2010. *(Cited on page 31)*

[117] J. Malczewski, "Gis-based multicriteria decision analysis: a survey of the literature," *International Journal of Geographical Information Science*, vol. 20, no. 7, pp. 703–726, 2006. *(Cited on page 31)*

[118] National Institute of Standards and Technology, "Cloud Computing Service Metrics Description," Tech. Rep., 2015. *(Cited on pages 32 and 68)*

[119] E. Marjomaa, "Necessary conditions for high quality conceptual schemata: Two wicked problems," *Journal of Conceptual Modeling*, vol. 1, no. 27, 2002. *(Cited on page 32)*

[120] H. Kangassalo, "On the concept of concept for conceptual modelling and concept detection," *Information modelling and knowledge bases*, vol. 3, pp. 17–58, 1992. *(Cited on page 32)*

[121] M. P. Singh and M. N. Huhns, *Service-oriented computing: semantics, processes, agents*. John Wiley & Sons, 2006. *(Cited on pages 32 and 35)*

[122] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004. *(Cited on page 32)*

[123] V. W. Setzer, "Data, information, knowledge and competency," *Ciência da Informação, DataGramaZero, n. zero, dez*, 1999. *(Cited on page 33)*

[124] N. Bontis, "Managing organisational knowledge by diagnosing intellectual capital: framing and advancing the state of the field," *International Journal of technology management*, vol. 18, no. 5-8, pp. 433–462, 1999. *(Cited on page 33)*

[125] E. A. Smith, "The role of tacit and explicit knowledge in the workplace," *Journal of knowledge Management*, vol. 5, no. 4, pp. 311–321, 2001. *(Cited on page 33)*

[126] J. Cardoso, "The syntactic and the semantic web," *IGI Global*, p. 21, 2007. *(Cited on pages 33 and 34)*

[127] J. Cardoso and A. Sheth, "The semantic web and its applications," in *Semantic Web Services, Processes and Applications*. Springer, 2006, pp. 3–33. *(Cited on page 33)*

[128] T. R. Gruber *et al.*, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993. *(Cited on page 34)*

[129] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies*, vol. 43, no. 5, pp. 907–928, 1995. *(Cited on page 34)*

[130] N. Borstw, "Construction of engineering ontologies," *Enschede: University of Twente*, 1997. *(Cited on page 34)*

[131] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web. a new form of web content that is meaningful to computers will unleash a revolution of new possibilities," *Scientific American*, vol. 284, no. 5, pp. 1–5, 2001. *(Cited on pages 34 and 35)*

[132] O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez, "Ontological engineering: what are ontologies and how can we build them?" 2007. *(Cited on page 34)*

[133] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004. *(Cited on page 35)*

[134] M. B. Almeida, "An Introduction to XML, its use on the Internet and some Complementary Concepts," *Ci. Inf. vol.31 no.2*, 2002. *(Cited on pages 35 and 50)*

[135] G. Klyne and J. J. Carroll, "Resource description framework (rdf): Concepts and abstract syntax," 2006. *(Cited on page 35)*

[136] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean *et al.*, "Swrl: A semantic web rule language combining owl and ruleml," *W3C Member submission*, vol. 21, p. 79, 2004. *(Cited on page 35)*

[137] H. Boley, S. Tabet, and G. Wagner, "Design rationale for ruleml: A markup language for semantic web rules." in *SWWS*, vol. 1, 2001, pp. 381–401. *(Cited on page 35)*

[138] A. Horn, "On sentences which are true of direct unions of algebras," *The Journal of Symbolic Logic*, vol. 16, no. 01, pp. 14–21, 1951. *(Cited on page 35)*

[139] J. Kang and K. M. Sim, "Cloudle: an ontology-enhanced cloud service search engine," in *International Conference on Web Information Systems Engineering*. Springer, 2010, pp. 416–427. *(Cited on pages XIII and 36)*

[140] Y. B. Ma, S. H. Jang, and J. S. Lee, "Ontology-based resource management for cloud computing," in *Asian Conference on Intelligent Information and Database Systems*. Springer, 2011, pp. 343–352. *(Cited on page 36)*

[141] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0," *University of Manchester*, 2004. *(Cited on page 37)*

[142] G. Di Modica and O. Tomarchio, "Matching the business perspectives of providers and customers in future cloud markets," *Cluster Computing*, vol. 18, no. 1, pp. 457–475, 2015. *(Cited on pages XI, 37, 38, and 39)*

[143] M. Rak, R. Aversa, S. Venticinque, and B. Di Martino, "User centric service level management in mosaic applications," in *European Conference on Parallel Processing*. Springer, 2011, pp. 106–115. *(Cited on pages XI, 39, and 40)*

[144] F. DAndria, S. Bocconi, J. G. Cruz, J. Ahtes, and D. Zeginis, "Cloud4soa: multi-cloud application management across paas offerings," in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*. IEEE, 2012, pp. 407–414. *(Cited on page 40)*

[145] A. J. Ferrer, F. HernáNdez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame *et al.*, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012. *(Cited on page 40)*

[146] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman *et al.*, "Reservoirâwhen one cloud is not enough," *IEEE computer*, vol. 44, no. 3, pp. 44–51, 2011. *(Cited on page 40)*

[147] "4CaaSt Project." [Online]. Available: http://4caast.morfeo-project.org/ *(Cited on page 40)*

[148] R. G. Cascella, C. Morin, P. Harsh, and Y. Jegou, "Contrail: A reliable and trustworthy cloud platform," in *Proceedings of the 1st European Workshop on Dependable Cloud Computing*. ACM, 2012, p. 6. *(Cited on page 40)*

[149] "IRMOS Project." [Online]. Available: http://www.irmosproject.eu/ *(Cited on page 40)*

[150] "SLA@SOI." [Online]. Available: http://sla-at-soi.eu/ *(Cited on page 40)*

[151] "ETICS Project." [Online]. Available: https://www.ict-etics.eu/ *(Cited on page 40)*

[152] A.-F. Antonescu, P. Robinson, L. M. C. Murillo, J. I. Aznar, S. Soudan, F. Anhalt, and J. A. G. Espín, "Towards cross stratum sla management with the geysers architecture." in *ISPA*, 2012, pp. 527–533. *(Cited on page 40)*

[153] "VISION Cloud Project." [Online]. Available: http://www.visioncloud.eu/ *(Cited on page 40)*

[154] "CumuloNimbo Project." [Online]. Available: http://www.cumulonimbo.eu/ *(Cited on page 40)*

[155] D. L. Moody, "Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: current state and future directions," *Data Knowl. Eng.*, vol. 55, no. 3, pp. 243–276, 2005. *(Cited on page 44)*

[156] L. J. Campbell, T. A. Halpin, and H. A. Proper, "Conceptual Schemas with Abstractions - Making flat conceptual schemas more comprehensible," *Data and Knowledge Engineering*, vol. 20, no. 1, pp. 39–85, 1996. *(Cited on page 44)*

[157] B. E. Bargmeyer and D. W. Gillman, "Metadata standards and metadata registries: An overview," in *International Conference on Establishment Surveys II*, vol. 19, 2000. *(Cited on page 44)*

[158] A. Gali and C. Chen, "From Ontology to Relational Databases," *Conceptual Modeling for . . .*, pp. 1–12, 2004. *(Cited on page 50)*

[159] E. Tsang, "Foundations of constraint satisfaction," 1995. *(Cited on pages 65 and 66)*

[160] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Fragmentation in presence of data dependencies," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 11, no. 6, pp. 510–523, November/December 2014. *(Cited on page 66)*

[161] R. E. Johnson and B. Foote, "Designing reusable classes," *Journal of object-oriented programming*, vol. 1, no. 2, pp. 22–35, 1988. *(Cited on page 98)*

[162] R. Feldman and J. Sanger, *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge University Press, 2007. *(Cited on page 112)*

[163] W. L. Kuechler, "Business applications of unstructured text," *Communications of the ACM*, vol. 50, no. 10, pp. 86–93, 2007. *(Cited on page 113)*

[164] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011. *(Cited on page 113)*

[165] D. Sullivan, *Document warehousing and text mining: techniques for improving business operations, marketing, and sales*. John Wiley & Sons, Inc., 2001. *(Cited on page 113)*

[166] L. A. Zadeh, "Fuzzy logic= computing with words," *IEEE transactions on fuzzy systems*, vol. 4, no. 2, pp. 103–111, 1996. *(Cited on page 115)*

[167] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984. *(Cited on page 130)*

A

PUBLICATIONS

Some ideas and significant results present in this thesis were published in:

1. **"On the Use of Fuzzy Logic in Dependable Cloud Management"**

   S. Foresti, V. Piuri, G.A. Soares

   in Proc. of the 3rd IEEE Conference on Communications and Network Security (CNS 2015), Florence, Italy, September 28-30, 2015.

   Abstract: The effective and efficient use of dependable cloud infrastructures requires the agreement between users and cloud providers on resources, services, operating conditions, and features as well as the mapping of usersâ requirements onto the cloud architecture. In this paper, we identify the different ways in which fuzzy logic can be profitably adopted in performing these tasks, providing flexibility in capturing usersâ needs and dealing with complex architectures and conflicting or hardly-satisfiable requirements. We specifically put forward the idea of using fuzzy logic at the user-side, to enable the specification of usersâ needs in crisp or fuzzy ways and their homogenous processing.

2. **"Supporting Application Requirements in Clod-based IoT Information Processing"**

   P. Samarati, G.A. Soares, V. Piuri, G. Livraga, S. De Capitani di Vimercati

   in Proc. of the International Conference on Internet of Things and Big Data (IoTBD 2016), Rome, Italy, April 23-25, 2016.

   Abstract: IoT infrastructures can be seen as an interconnected network of sources of data, whose analysis and processing can be beneficial for our society. Since IoT devices are limited in storage and computation capabilities, relying on external cloud providers has recently been identified as a promising solution for

storing and managing IoT data. Due to the heterogeneity of IoT data and applicative scenarios, the cloud service delivery should be driven by the specific IoT applications. In this paper, we propose a novel approach for supporting application requirements for cloud-based IoT data processing. Our solution allows an IoT infrastructure authority to formulate conditions that the provider must satisfy in service provisioning, and computes a SLA based on these conditions while also accounting for possible dependencies among them. We also illustrate a CSP-based formulation of the problem of computing a SLA, which can be solved adopting off-the-shelves CSP solvers, possibly accommodating user preferences in the computation of the solution.