Deriving UML Logic Architectures of Software Product based on a Cloud Reference Architecture: An Experience Report

Francisco Morais[1][a], Tiago F. Pereira[1][b], Carlos Salgado[1][c], Ana Lima[1][d], Manuel Pereira[2][e], João Oliveira[2][f], António Sousa[3][g] and Helena Rodrigues[4][h]

[1]*CCG - Centro de Computação Gráfica, Campus de Azurém, Guimarães, Portugal*

[2]*F3M Information systems S.A., Braga, Portugal*

[3]*HASLab/INESC TEC, Universidade do Minho, Braga, Portugal*

[4]*Centro ALGORITMI, Escola de Engenharia, Universidade do Minho, Guimarães, Portugal*

Keywords: Cloud Computing, Design Methods, Cloud Reference Architecture, Cloud Requirements, Logic Architecture.

Abstract: Companies are nowadays looking for the development of solutions based on public and private clouds capable of interoperating with information sources such as other systems, or devices in an IoT and CPS approach, and subsequently using that information efficiently. However, applying appropriate techniques for requirements engineering and designing logical architectures for that context can be complex. The cloud environments are very dynamic and are difficult to identify, clarify, and systematically manage cloud requirements. The lack of requirements engineering methods for this domain carry risks related to incorrect or unjustified decisions, which result in subjective project developments. This paper presents the use of NIST Cloud Computing Reference Architecture in the eliciting of requirements by employing a new approach (the 2P2S technique), that enables the use of an existing Model-Driven Design method (the 4SRS technique), derive logic architectures for cloud-based solutions, assuring that the system requirements are based on effective client needs.

1 INTRODUCTION

According to Reese (Reese, 2009), Cloud Computing (CC) is the evolution of a variety of technologies that have come together to alter an organization's approach to building out an IT infrastructure. Adopting the cloud and its services brings fundamental changes in how organizations think and engineer their requirements. Moreover, designing logical architectures for that context can be knotted. One of the problems is the natural discontinuity between functional and structural models (Machado et al., 2005). Another is the requirements modeling for CC, in the degree to which requirements are correctly understood by the cloud service providers and consumers. Unlike classical requirements engineering, these processes should cater for scale, decentralization, uncertainties, and heterogeneity making traditional approaches limited in their applicability. Thus, matching between the user requirements and features of the cloud is a key asset for a correct evaluation of cloud services and their adoption (Zardari and Bahsoon, 2011). To accomplish the matching between the user requirements and features of the cloud, proper requirements elicitation and logical architecture are required, to help increase the comprehensibility of requirements for cloud service consumers, and model a technical architecture needed to place the application on the cloud, exploring cloud computing concepts, architectural references, and SaaS business models.

Since standards and reference models enable the use of domain best practices, the cloud requirements should be based on cloud computing models. The National Institute of Standards and Technology (NIST) has developed a logical extension of its cloud definition by developing the NIST Cloud Computing Ref-

[a] https://orcid.org/0000-0001-6579-8509
[b] https://orcid.org/0000-0001-8155-4491
[c] https://orcid.org/0000-0002-5436-0082
[d] https://orcid.org/0000-0003-3077-6662
[e] https://orcid.org/0000-0001-9470-2764
[f] https://orcid.org/0000-0003-0629-4444
[g] https://orcid.org/0000-0001-8639-5346
[h] https://orcid.org/0000-0002-8978-8804

Morais, F., Pereira, T., Salgado, C., Lima, A., Pereira, M., Oliveira, J., Sousa, A. and Rodrigues, H.
Deriving UML Logic Architectures of Software Product based on a Cloud Reference Architecture: An Experience Report.
DOI: 10.5220/0011146700003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 399-409
ISBN: 978-989-758-588-3; ISSN: 2184-2833

erence Architecture (NIST CCRA) (Liu, F., Tong, J.; Mao, J.; Bohn, R.; Messina, J.; Badger, M.; Leaf, D., 2011). First, we discuss some related work in Section 2 on CC requirements engineering approaches. Section 3 shows the proposed approach for eliciting requirements and deriving logic architectures for the cloud. In Section 4, we present our case study. Section 5 describes a brief discussion of the results, and finally, Section 6 concludes the paper.

2 RELATED WORK

To create a software architecture model, one must understand the problem, find a solution, and evaluate the final result, according to (Souza et al., 2019). Finding a solution is related to deriving a software architecture from the requirements specification.

Existing software architecture methods offer ways to derive architectural models from requirements specifications. But creating an architectural model is difficult, especially because requires a balance between the different forces imposed by different application domains and quality attributes. Also, (Souza et al., 2019) refers that the current architectural derivation methods rely heavily on the architect's tacit knowledge (experience and intuition), do not offer sufficient support for inexperienced architects, and lack explicit evaluation mechanisms. This leads to poor requirements understanding process, undetailed decision-making process, inconsistency between artifacts, semantic loss, and lack of traceability between models. (Souza et al., 2019) highlights the lack of standardized terms, reducing understandability, making the communication between stakeholders inefficient and error-prone. While the standardization of terms contributes to an established body of language, beneficial to software architecture researchers and practitioners, also could contribute to the adoption of Architecture Description Languages - ADL (UML, sysML, other). ADLs poor adoption is mainly because they need to capture design decisions judged fundamental to satisfy different stakeholder concerns and also because it is difficult to capture all the different concerns with a unique language. Standardization of terms would facilitate the adoption of ADLs.

The cloud environments are stochastic and dynamic, so it is complex to identify, clarify and manage cloud requirements in a systematic way (Zardari and Bahsoon, 2011), (Iankoulova, 2012). According to (Zalazar A.S., 2017), the CC requirements engineering approaches are mostly concentrated in object-oriented artifacts and tools. And existing requirements engineering processes for CC are generally about nonfunctional requirements, usually focused on particular characteristics, as security, privacy, availability, and other performance aspects. CC is complex to administrate because of dynamism imposed by the context of having elastic resources (scaling, resizing, on/off), requirements depending on business changes (peak and non-peak times), consumers being heterogeneous from different geographic places and jurisdictions, and distributed systems being managed remotely.

Authors like (Schrodl and Wind, 2011) conclude that none of the common models (V-model, Volere, Extreme programming, and Rational Unified Process) is suitable to cover the needs of requirements under cloud computing. Also, there is little information about frameworks and methods for supporting projects for this domain (Klems et al., 2009). Organizations try to adapt existing techniques or create new ones for supporting cloud projects, but missing a systematic guidance (Zardari and Bahsoon, 2011), (Koitz et al., 2013). The lack of engineering methods for this domain carried risks related to incorrect or unjustified decisions, which result in subjective project developments. Thus, (Rimal, 2011), states that a big challenge for CC is the lack of a standard that helps meet the objectives covering many different aspects of cloud services.

(Zalazar A.S., 2017) refers that the recent contributions to the requirements engineering approaches, proposes frameworks to evaluate and to handle requirements for CC projects. However, there are no well-known tools and automatic techniques supporting that CC adoption.

Some approaches pursue the idea of cloud computing as a multidimensional paradigm, related with cloud services and SLA, and used to identify cloud requirements and constraints in natural language, and being the bases for models, and ontologies.

For example, (Repschlaeger et al., 2012) presents six target dimensions based on general objectives (service and cloud management, IT security and compliance, reliability and trustworthiness, scope and performance, costs, and flexibility). (Pichan et al., 2015) consider only three dimensions (technical, organizational, and legal). Also, (Zalazar A.S., 2017) proposes five dimensions (service cloud dimensions of contractual, financial, compliance, operational and technical), considering CC as a multidimensional paradigm where different activities, roles, and service dimensions are integrated. Also, because requirements change frequently, the author states that a streamline is needed, to monitor and trace requirements using traceability mechanisms, but without specifying them.

(Souza et al., 2019) refers to the current practice of deriving architectural models from requirements specification that indicates they have been based on the architect's tacit knowledge, without enough systematization and tool support, lack of standardized terms in the definition of software architecture concepts, and lack of existing architecture evaluation methods. This indicates there is much interest in documenting architectural decisions to reduce the "architectural vaporization knowledge" during the evolution of architecture. Also mention a variety of strategies for specific types of development environments, like cyber foraging, model-driven design, automotive open software architecture, component-based architecture, service-based architecture, decision support for clinical systems, computer game architecture, reference architectures, architectural patterns, architecture erosion. But concludes that reusing knowledge about decision-making and evaluation is still a topic worthy of further research. The current practice of deriving architectural models from requirements specifications still has some gaps.

3 PROPOSED APPROACH

Our approach uses the Use Case and Logical views defined by The Rational Unified Process (RUP) (Reed, 2002). We use the use case view model to gather the cloud requirements, which are used as input to derive a logic architecture as the first view of a cloud architecture.

3.1 Requirements Elicitation for the Cloud

Our approach is aligned with the NIST CCRA framework, through the mapping of the NIST layers components with the product use cases, and it takes into account architectural decisions considering a particular CC deployment model. These decisions are taken by the software product owner, considering the degree of technological dependency he requires from the cloud provider for the service model (from more independent IaaS to more dependent SaaS cloud components), and also the degree of dependency from the cloud infrastructure for the deployment model (from more dependent public cloud, to more independent private cloud infrastructures). NIST CCRA also identifies the Cloud Carrier (communications operator that provides connectivity and transport of services from cloud providers to the cloud consumers), but this is not considered in our requirements elicitation.

The requirements elicitation for the cloud is executed in two phases, each of them comprising two steps (2P2S), as depicted in figure 1. In the first phase, we consider the degree of dependency from the cloud infrastructure provider. In the second phase, we consider the degree of technological dependency from the cloud provider.

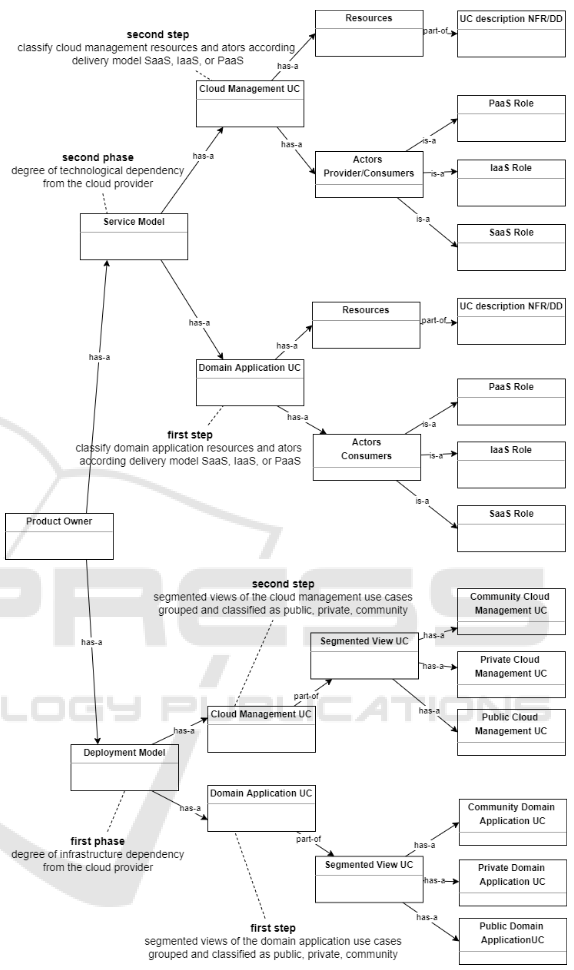

Resources
UC description NFR/DD
part-of
second step
classify cloud management resources and ators according delivery model SaaS, IaaS, or PaaS
has-a
Cloud Management UC
PaaS Role
is-a
has-a
second phase
degree of technological dependency from the cloud provider
has-a
Actors Provider/Consumers
is-a
IaaS Role
is-a
Service Model
SaaS Role
Resources
UC description NFR/DD
part-of
has-a
has-a
has-a
Domain Application UC
PaaS Role
has-a
is-a
first step
classify domain application resources and ators according delivery model SaaS, IaaS, or PaaS
Actors Consumers
is-a
IaaS Role
is-a
Product Owner
SaaS Role
second step
segmented views of the cloud management use cases grouped and classified as public, private, community
Community Cloud Management UC
has-a
Segmented View UC
has-a
Private Cloud Management UC
has-a
part-of
has-a
Cloud Management UC
Public Cloud Management UC
has-a
Deployment Model
Domain Application UC
has-a
part-of
Community Domain Application UC
first phase
degree of infrastructure dependency from the cloud provider
has-a
Segmented View UC
has-a
Private Domain Application UC
has-a
first step
segmented views of the domain application use cases grouped and classified as public, private, community
Public Domain ApplicationUC

Figure 1: 2P2S concept diagram.

Phase 1 Step 1: Domain Application Deployment Model. The aim of the first step is to build segmented views of the application use case model (domain application). The use cases are grouped and classified as public, private, or community, producing one segmented view per each of these classifications, named as Public Domain Application use cases (UC), Private Domain Application UCs, and Community Domain UCs. This classification is based on the product-owner decisions on the deployment model for the application functionality. The result of this step is depicted in figure 2 (left side part), which considers a situation where domain application functionality is split across the three deployment models. The prefix letter of the UC numbering is only used here for readability, distinguishing between the several segmented views. The UC numbering should follow known techniques as the Volere Template (Robertson and Robertson, 2000).

Phase 1 Step 2: Cloud Management Deployment Model. In the second step, the designer adds to the UC model the cloud management functionality (configuration and administration) as described by the NIST CCRA. These Cloud Management UCs include all functions related to services necessary for the management and operations of the services requested or proposed to customers, according to the service model of the provider and the consumers. In the same way as above, if we consider a situation where domain application functionality is split across the three deployment models, we will have the corresponding cloud management functionality for each deployment model of the domain application. The result of this step is depicted in figure 2 (right side part). Graphically, the Use-Case refinement can be represented as a tree structure. In the level 0 (root level), it is represented the most abstract requirements that are related to a lower level of refinement. As template for describing the cloud management use cases we propose the following descriptions for level 0 use cases:

- {dU.C.1}, {eU.C.1}, {fU.C.1} Cloud account support management: involves the set of services supporting the contract between cloud provider and cloud consumer in the public cloud. It includes the components used to perform these business operations, such as account management, contract management, catalog management, billing management. These use cases are detailed in Account management, Contract Management, Service catalog management, and invoicing management level 1 use cases.
- {dU.C.2}, {eU.C.2}, {fU.C.2} Cloud services provisioning and configuration management: includes the set of services for provisioning, monitoring, and reporting on cloud services, measuring resource usage and service level management, such as Quality of Service (QoS) parameters for the Service Level Agreements (SLA). These use cases are detailed in Application management, Platform monitoring, Resource utilization measurement, and SLA management level 1 use cases.
- {dU.C.3}, {eU.C.3}, {fU.C.3} Portability and interoperability management of the cloud: includes the mechanisms to support data portability, service interoperability, and system portability. From the cloud consumers' perspective, the activities of data portability and service interoperability between public cloud data and external information systems are performed. These use cases are described in Systems Integration and Data Synchronization level 1 use cases.
- {dU.C.4}, {eU.C.4}, {fU.C.4} Cloud security and privacy management: perform a set of functionalities that allow the platform administrator to manage backups, monitor activities, and configure access controls. In a logic of shared responsibilities between the Cloud Provider and Cloud Consumers, these components are provided by the Provider for privileged access to resource services. The management of application user accounts is the responsibility of the Cloud Consumer. These use cases are detailed in Backup management, Configuration of data access control, and Activity monitoring level 1 use cases.

Phase 2 Step 1: Domain Application Service Model. In this step the objective is to classify the Domain Application resources and actors according to the product-owner decisions on the delivery model (SaaS, IaaS, or PaaS). The classification of the actors promotes the discussion with stakeholders in order to validate roles and raise functional requirements. The classification of the resources appears in the description of the use cases to be used as non-functional requirements or design decisions later on as modeling decisions on the Logic Architecture Design. The result of this step is depicted in figure 2 (actors on the left side).

Phase 2 Step 2: Cloud Management Service Model. The purpose of this last step is to classify the Cloud Management resources and actors according to the product-owner decisions on the delivery model (SaaS, IaaS, or PaaS). As in the previous step, this classification promotes the discussion with stakeholders, in order to validate roles and modeling decisions considering the technology dependency from the cloud providers. The classification of the resources appears in the description of the use cases to be used as non-functional requirements or design decisions later on the Logic Architecture Design. The result of this step is depicted in figure 2 (actors on the right side).

3.2 Logic Architectural Model for the Cloud

The 4SRS technique, in its original version (Machado et al., 2005), allows the transformation of the user requirements model into the first logical system architecture, based on mapping of UML use-case diagrams

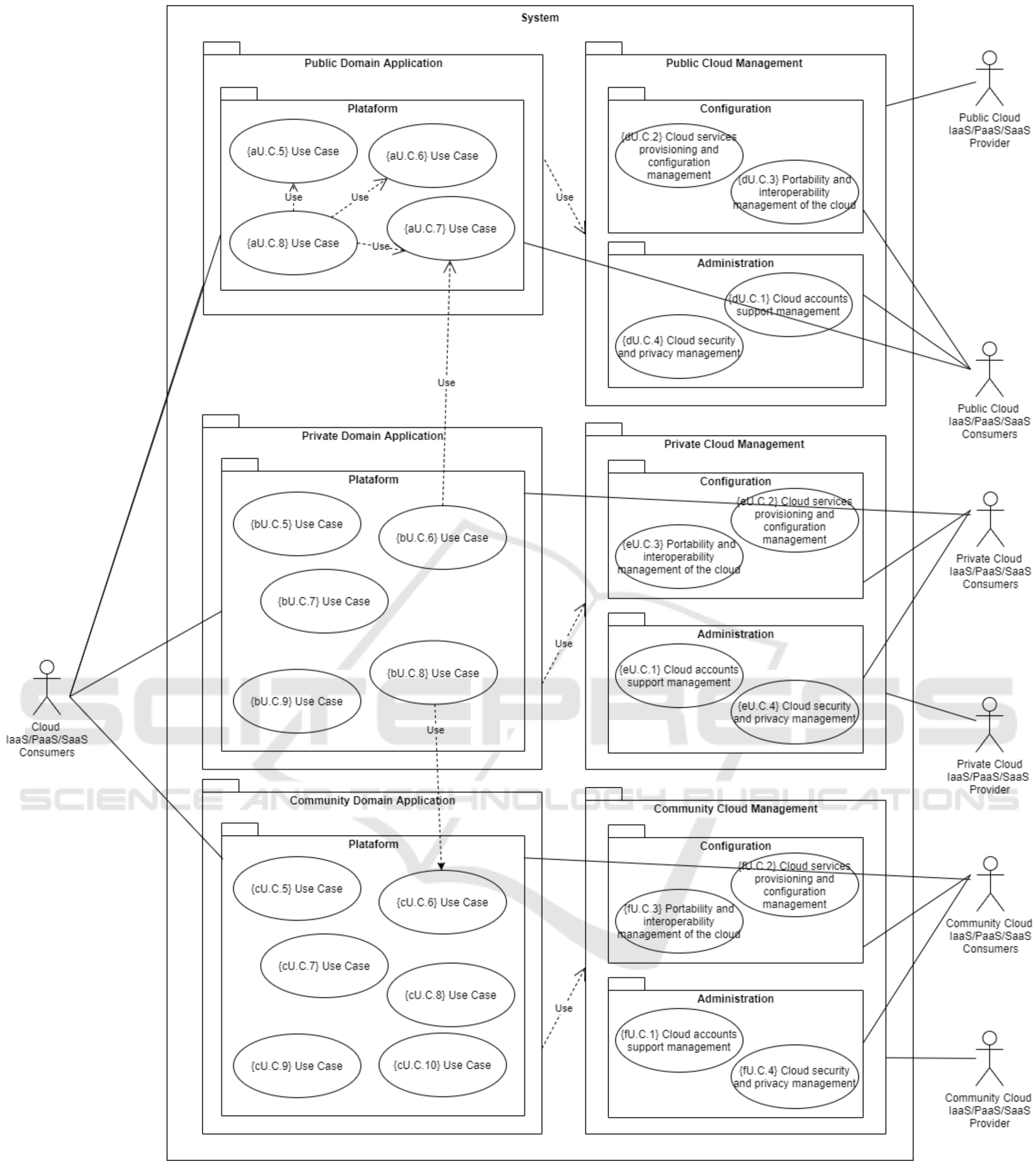

System
Public Domain Application
Plataform
{aU.C.5} Use Case
{aU.C.6} Use Case
{aU.C.7} Use Case
{aU.C.8} Use Case
Use
Public Cloud Management
Configuration
{dU.C.2} Cloud services provisioning and configuration management
{dU.C.3} Portability and interoperability management of the cloud
Administration
{dU.C.1} Cloud accounts support management
{dU.C.4} Cloud security and privacy management
Public Cloud IaaS/PaaS/SaaS Provider
Public Cloud IaaS/PaaS/SaaS Consumers
Private Domain Application
{bU.C.5} Use Case
{bU.C.6} Use Case
{bU.C.7} Use Case
{bU.C.8} Use Case
{bU.C.9} Use Case
Private Cloud Management
{eU.C.2} Cloud services provisioning and configuration management
{eU.C.3} Portability and interoperability management of the cloud
{eU.C.1} Cloud accounts support management
{eU.C.4} Cloud security and privacy management
Private Cloud IaaS/PaaS/SaaS Consumers
Private Cloud IaaS/PaaS/SaaS Provider
Cloud IaaS/PaaS/SaaS Consumers
Community Domain Application
{cU.C.5} Use Case
{cU.C.6} Use Case
{cU.C.7} Use Case
{cU.C.8} Use Case
{cU.C.9} Use Case
{cU.C.10} Use Case
Community Cloud Management
{fU.C.2} Cloud services provisioning and configuration management
{fU.C.3} Portability and interoperability management of the cloud
{fU.C.1} Cloud accounts support management
{fU.C.4} Cloud security and privacy management
Community Cloud IaaS/PaaS/SaaS Consumers
Community Cloud IaaS/PaaS/SaaS Provider

Figure 2: Segmented Views of the Use Case UML Diagram.

for UML object diagrams. Our approach is based on the 4SRS method and takes as input a set of use cases describing the requirements for the cloud management services and domain application services, representing both the intended application and cloud concerns based on NIST CCRA, of the involved business and technological stakeholders. This technique is organized in four steps, to transform use cases into objects, being this transformation supported by tabular representations, constituting the main mechanism to automate the decisions assisted by these four steps. Each column of the tabular representation, depicted in the figure 3, corresponds to a step (or micro-step) in the execution of the 4SRS method, and shows an example of this transformation. Next, each of these four steps is described, applied to the use case diagrams mapped to the NIST CCRA reference architecture, generating a specification for a cloud service-oriented platform (SOA).

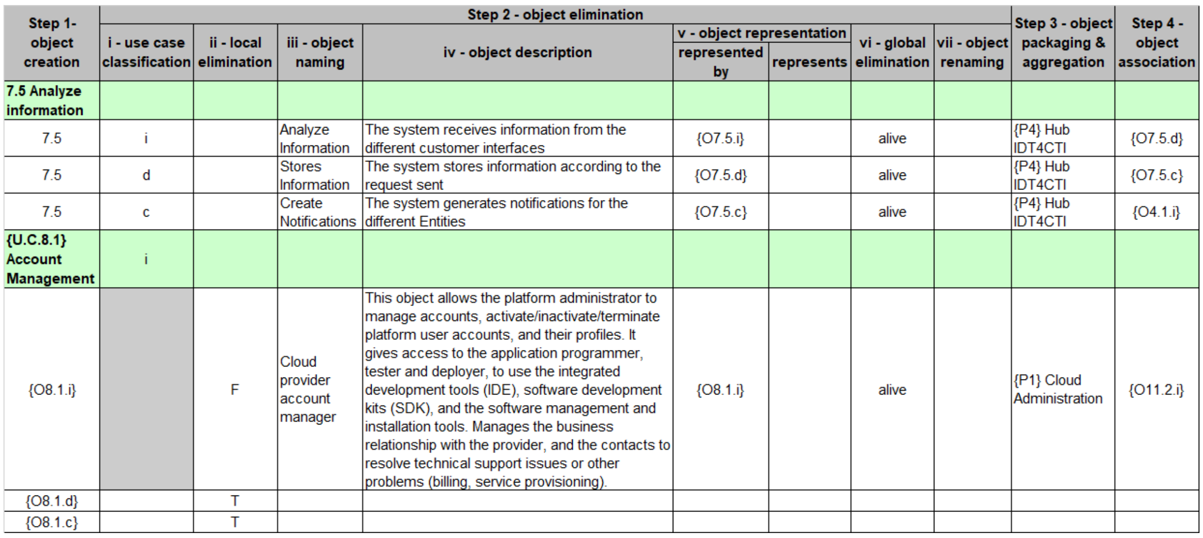
Step 1: Creating Objects. In the first step each use case is transformed into three objects, an interface object, a data object, and a control object. Each object receives the reference or numbering of the use case preceded by the letter "O", and a suffix i, d, or c, indicating the category of the object. From this step onwards there are only objects as modeling entities, although use cases are still used to introduce requirements in the object model.

Step 2: Eliminating Objects. In this step, a decision is made on which of the three objects should be kept, to computationally represent each use case, taking into account its textual description, and the system as a whole. In addition to this decision to keep objects from step 1, decisions are also made to eliminate redundant user requirements, and decisions to discover new requirements, and in this step, the definitive system-level entities are defined. To reduce complexity, this step is composed of seven micro-steps:

- Micro step 2i, classification of use cases: each use case can generate one of eight patterns, if ignoring the link between objects, by the different combination of its category (ϕ, i, c, d, ic, di, cd, icd). The goal is to transform use cases into objects, giving clues as to which categories of objects to use and how to link them together through associations;
- Micro step 2ii, elimination of objects: in this step, the objects assigned in step 1 are eliminated, since the assignment of categories i, c, d does not consider the problem domain. It takes into account the classification assigned in micro-step 2i, and the description of the use case in the context of the problem domain to be solved;
- Micro step 2iii, naming of objects: the objects not previously eliminated are named, reflecting the use case and the specific role of the object in the main object;
- Micro step 2iv, description of the objects: each object resulting from the previous micro-step 2iii is described so that the requirements representing these objects are included in the object model. These descriptions should be based on the original use case descriptions, except if they are classified as non-functional requirements (NFR) or design decisions (DD);
- Micro step 2v, object representation: a decision is made to eliminate redundancies of user requirements as well as discover new requirements. It constitutes an internal validation that ensures semantic consistency of the object model and discovers anomalies in the use case model;
- Micro step 2vi, global elimination: it is based on the result of the previous micro-step. Objects that have come to be represented by others are eliminated since their system requirements no longer belong to them. In this way a coherent object model is maintained from the requirements point of view;
- Micro step 2vii, object renaming: the purpose of this last micro-step of step two, is to rename the objects that were not deleted in the previous micro-step, and that represent additional objects. These new names must fully describe the system requirements they represent.

Step 3: Packaging and Aggregation. Packaging is a technique that introduces a (tenuous) semantic cohesion between objects, which can be easily reversed in the modeling phase. In other words, it is used in a flexible way to obtain a temporary (or definitive) understanding of the object model. Aggregation, on the other hand, imposes a strong semantic cohesion between objects and is more difficult to be reversed in the following modeling phases. In other words, aggregation should be used based on modeling options that are aware of their impact on the considered objects. Aggregation is typically used when there is a part of the system that is part of a pre-existing subsystem, or when modeling takes into account an architectural reference model that constrains the object model. In this way, and this step, the objects maintained from the previous step and which offer advantages in being considered together, give rise to aggregations or packages of objects with semantic consistency (stronger or weaker respectively. Thus obtaining the construction of a coherent object model, since it introduces a semantic layer at an abstraction level that works as a functional link between the objects.

Step 4: Association of Objects. The last step of the 4SRS technique introduces associations in the object model, based on the existing information in the use cases, and on the information generated in the micro-step 2i. In case the textual description of the use cases provides clues about the sequence of actions, this information is used to generate associations between objects. At the same time, the use case diagram in UML can itself include associations between use cases and be used for the object model. In the case of the micro-step 2i of object classification, this generated information should be used to include associations between the objects originally obtained from the same use case.



Step 1- object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object association
	i - use case classification	ii - local elimination	iii - object naming	iv - object description	v - object representation		vi - global elimination	vii - object renaming		
					represented by	represents				
7.5 Analyze information										
7.5	i		Analyze Information	The system receives information from the different customer interfaces	{O7.5.i}		alive		{P4} Hub IDT4CTI	{O7.5.d}
7.5	d		Stores Information	The system stores information according to the request sent	{O7.5.d}		alive		{P4} Hub IDT4CTI	{O7.5.c}
7.5	c		Create Notifications	The system generates notifications for the different Entities	{O7.5.c}		alive		{P4} Hub IDT4CTI	{O4.1.i}
{U.C.8.1} Account Management	i									
{O8.1.i}		F	Cloud provider account manager	This object allows the platform administrator to manage accounts, activate/inactivate/terminate platform user accounts, and their profiles. It gives access to the application programmer, tester and deployer, to use the integrated development tools (IDE), software development kits (SDK), and the software management and installation tools. Manages the business relationship with the provider, and the contacts to resolve technical support issues or other problems (billing, service provisioning).	{O8.1.i}		alive		{P1} Cloud Administration	{O11.2.i}
{O8.1.d}		T								
{O8.1.c}		T								

Figure 3: Steps of the 4SRS method.

The execution of the 4SRS method ends with the construction of a logical diagram, which represents the logical architecture of the functionalities of the software system. This logical architecture arises as a result of the application of the 4SRS method to the user requirements of the cloud management services and domain application services, and that expresses the system requirements. The user requirements gave rise to textual descriptions of each object in the model and to non-functional intentions classified as NFR and DD modeling decisions, if applicable, in micro-step 2iv [4], as bounded contexts, namely: intra-service behavior; interfaces; data models separation; inter-service communication; and messaging requirements.

4 CASE STUDY

4.1 The Problem

To face some business challenges in the Textile and Clothing Industry, the development of Interoperability and Digital Thread for a More Competitive Textile Industry domain (IDT4CTI) project, will be supported on a platform based on the Industry 4.0 paradigm, mainly adopting cloud computing models, adopting SOA approaches and interoperability solutions. In this context, the different steps of the garment manufacturing process leading to numerous iterations and dependencies were designed, with consequent communication and interoperability needs between the textile company and its subcontractors. To address the difficulties encountered in the current process and meet the interoperability needs between the different actors of the Company ecosystem, the main interactions, and respective sub-processes have been identified. The Business processes were characterized, as well as the domain functional and non-functional requirements. The requirements for this system were acquired using requirements engineering techniques, and the result was a collection of artifacts, including UML Use Case diagrams. To address the challenge of deriving a cloud architecture from the modeled requirements, we describe the application of the proposed approach in this paper, first mapping the use case diagram to NIST CCRA and obtaining cloud requirements, and then applying the 4SRS method to derive a first logical architecture for the cloud.

4.2 Cloud Requirements

The cloud requirements result from the application of the two-phase two-step method (2P2S) described in section 3.1. In the first phase, we discussed with the software product owner the dependency on the cloud infrastructure. In the first step, the software product owner decided to deploy its software product in a cloud infrastructure with computing resources available to the general public over the public internet network, and owned and managed by a Public Cloud Provider organization. We grouped the domain application use cases in one segmented view and classified it as a Public Domain Application UC. In a second step, we added a segmented view classified as Public Cloud Management, containing the functions related to services offered by the public cloud provider, used by the Domain Application, to be configured and administrated in the Cloud Provider Infrastructure.

In the second phase, we discussed with the soft-

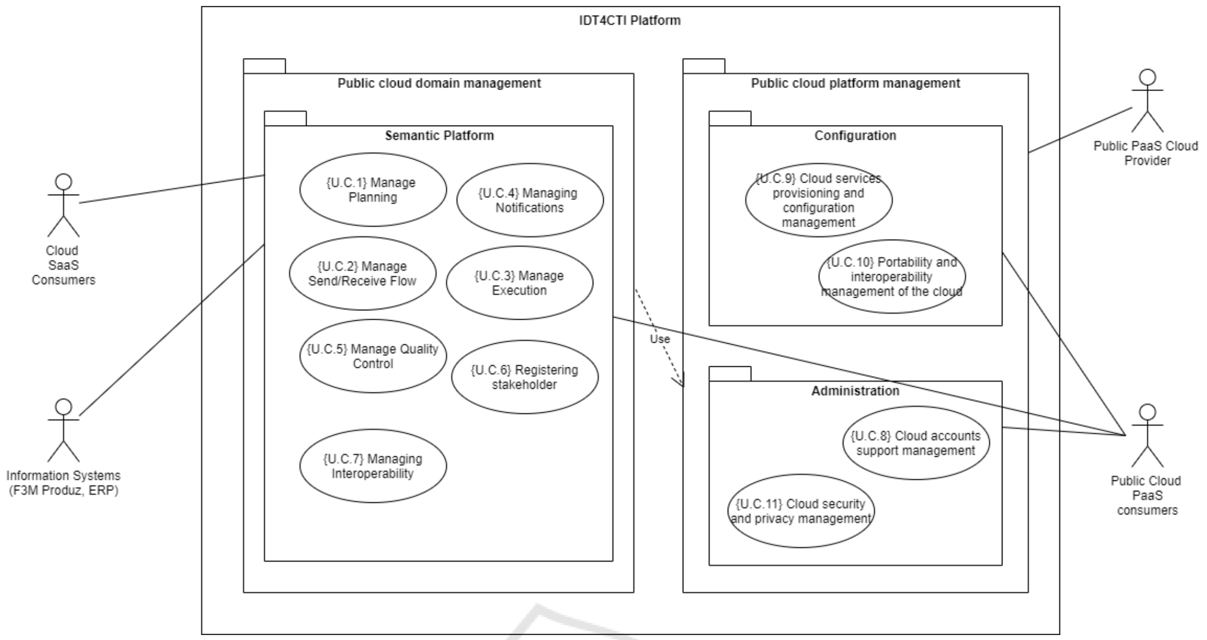

IDT4CTI Platform
Public cloud domain management
Semantic Platform
{U.C.1} Manage Planning
{U.C.4} Managing Notifications
{U.C.2} Manage Send/Receive Flow
{U.C.3} Manage Execution
{U.C.5} Manage Quality Control
{U.C.6} Registering stakeholder
{U.C.7} Managing Interoperability
Public cloud platform management
Configuration
{U.C.9} Cloud services provisioning and configuration management
{U.C.10} Portability and interoperability management of the cloud
Use
Administration
{U.C.8} Cloud accounts support management
{U.C.11} Cloud security and privacy management
Cloud SaaS Consumers
Information Systems (F3M Produz, ERP)
Public PaaS Cloud Provider
Public Cloud PaaS consumers

Figure 4: IDT4CTI Platform Level 0 Use Cases UML Diagram.

ware product owner the degree of technological dependency from the cloud provider. In the first step, the decision was to deliver the software product as a SaaS application. In this case, SaaS Consumers should be billed based on the number of end-users, time of use, network bandwidth consumed, the amount of data stored, and the duration of data stored. We classified the actors using the Public Domain Application as Public Cloud SaaS Consumers. In a second step, the decision was to deploy the SaaS application directly into the Cloud PaaS resources of the cloud provider. In this case, the cloud provider is responsible to manage the computing infrastructure, i.e., managing the runtime software execution stack, databases, and other middleware components. Additionally, the cloud provider is responsible to support the development, installation, and management, by providing tools such as integrated development environments (IDEs), cloud software development versions, software development kits (SDKs), installation and management tools.

On the other hand, the product owner has decided for the control over the domain application and also for the configuration of the hosting environment, excepting for the configuration of the underlying infrastructure of the platform, such as network, servers, operating systems (OS) or storage. In this case, product owners may use the tools and execution resources provided by Cloud Providers to develop, test, install and manage applications hosted in a cloud environment. Following this decision, we classified actors of the Public Cloud Management View as Public Cloud PaaS Consumers and Public PaaS Cloud Provider.

Figure 4 presents the use case diagram for the highest level of abstraction, level 0, as a result of the requirements elicitation. This top-level UC diagram was further refined into a total of thirty-three level-one UCs. The description of these UCs was omitted due to space limits. In Figure 5, we present the matching between the thirty-three use cases, that capture the elicited user requirements (Public Domain Application UC and Public Cloud Management UC), and the features of the NIST CCRA layers components, for a correct evaluation of cloud services and their adoption. The UC model describes the main functionalities of the platform and their various interactions with different actors, promoting the association of PaaS providers, and PaaS and SaaS consumers. The Public Domain Application UCs refers to the functions that support the business needs of the Cloud SaaS Consumers actors. The Public Cloud Management View includes all the functions related to the management and operations requested or proposed to customers (Cloud SaaS Consumers actors). It refers to resources related to the management of user accounts, monitoring cloud resources, portability and interoperability, and reliability of a public cloud.

4.3 Cloud Architecture

The IDT4CTI Logic Architecture is the result of the 4SRS applied to the use case model (level-one)

NIST Layers	Use Cases
1. Cloud Consumers	not applied
2. Cloud Auditor	not applied
3. Cloud Broker	not applied
3.1 Service Intermediation	not applied
3.2 Service Aggregation	not applied
3.3 Service Arbitrage	not applied
4. Cloud Provider	
4.1 Service Layer	
4.1.1 SaaS	{U.C.1.1} Propose planning
	{U.C.1.2} Accept planning
	{U.C.2.1} Send raw material/product
	{U.C.2.2} Recall raw material/product
	{U.C.3.1} Report quantities produced
	{U.C.3.2} Report backlog in production
	{U.C.4.1} Send message
	{U.C.4.2} Receive message
	{U.C.5.1} Submit quality control
	{U.C.5.2} Receive quality control report
	{U.C.6.1} Register entity
	{U.C.6.2} Invite organisation
	{U.C.6.3} Manage Entity Profile
	{U.C.7.1} Subscribe to interoperability services
	{U.C.7.2} Manage information history
	{U.C.7.3} Submit information model
	{U.C.7.4} Consult indicators
4.1.2 PaaS	Cloud Infrastructure Responsibility
4.1.3 IaaS	Cloud Infrastructure Responsibility
4.2 Resource Abstraction and Control Layer	Cloud Infrastructure Responsibility
4.3 Physical Resource Layer	Cloud Infrastructure Responsibility
4.3.1 Hardware	Cloud Infrastructure Responsibility
4.3.2 Facility	Cloud Infrastructure Responsibility
4.4 Cloud Service Management	
4.4.1 Business Support	
4.4.1.1 Customer Management	{U.C.8.1} Account management
4.4.1.2 Contract Management	{U.C.8.2} Contract management
4.4.1.3 Inventory Management	{U.C.8.3} Service catalogue management
4.4.1.4 Accounting & Billing	{U.C.8.4} Invoice management
4.4.1.4 Reporting & Auditing	{U.C.9.2} Platform monitoring
4.4.1.4 Pricing & Rating	{U.C.8.3} Service catalogue management
4.4.2 Provisioning & Configuration	
4.4.2.1 Rapid Provisioning & Configuration	{U.C.9.1} Application management
4.4.2.1 Resource Change & Configuration	{U.C.9.1} Application management
4.4.2.3 Monitoring & Reporting Change	{U.C.9.2} Platform monitoring
4.4.2.4 Metering	{U.C.9.3} Measuring the use of services
4.4.2.5 SLA Management	{U.C.9.4} SLA management
4.4.3 Portability & Interoperability	
4.4.3.1 Data Portability & Interoperability	
4.4.3.1.1 Copy Data To-From	U.C.11.1 Data synchronisation
4.4.3.1.2 Bulk Data Transfer	U.C.11.1 Data synchronisation
4.4.3.2 Service Interoperability	
4.4.3.2.1 Unified Management Interface	U.C.11.2 Systems integration
4.4.3.3 System Portability	
4.4.3.3.1 VM Images Migration	Responsibility of Cloud Infrastructure
4.4.3.3.2 App/Svc Migration	Responsibility of Cloud Infrastructure
4.4.4 Security	{U.C.11.1} Backup management
	{U.C.11.2} Configuration of data access control
4.4.5 Privacy	{U.C.8.1} Account management
	{U.C.11.2} Configuration of data access control

Figure 5: Mapping NIST layers components with IDT4CTI UC.

elicited in section 4.2. The IDT4CTI architecture, depicted in Figure 6 is constituted by forty components. The logical architecture, represented as an object diagram, identifies the system entities, their responsibilities, and the relationships between the objects. Its purpose is to show the decomposition of the overall system without going into detail, and which represents the logical architecture of the functions of the IDT4CTI system.

The objects derived from the cloud administration use case were aggregated into a single ”black box” package since it will be implemented by native cloud components from Public Cloud Provider, relating to account administration in the cloud provider, account management, contract, billing, service catalog provisioning, backup management, and data access permissions and the tracking of user and service activity in the use of cloud resources. The cloud configuration use case objects were aggregated in another package and included the configuration of cloud services and resources provisioned from the Public Cloud Provider component catalog. These objects include the application management services and the SLA QoS parameters of the provisioned services, the measurement and usage of cloud resources, monitoring of cloud notifications and events, and report generation, and the integration of external systems and data into the platform. This Package is implemented by a native cloud component from Public Cloud Provider and therefore presented as a ”black box” package.

In order to group the different architectural components we divide the architecture into packages that represent the main processes of IDT4CTI. This packaging defines decomposition regions, which contain semantically cohesive objects, being grouped in the following 9 packages: P1 Cloud Administration; P2 Cloud Configuration; P3 Message Delivery Management; P4 Entity management; P5 Semantic Manager; P6 Notification management; P7 Information model management; P8 Authentication Management; P9 Message Query Management.

5 DISCUSSION

In this work, we based the elicitation of cloud requirements in the NIST CCRA framework and systematized a method (2P2S) to be executed in the requirements elicitation phase when building the Use Case Diagram. The Use Case model allows representing the process visualization of a system and can be considered as a suitable requirement gathering method for the cloud. The method 2P2S adds segmented views to the domain application’s use case diagram, allowing the deployment model for the application functionality to be classified. In the case of the IDT4CTI case study, we have introduced the cloud requirements in the use case model following the product owner perspective of a public cloud deployment model and compliant with the NIST CCRA framework. Following this, the 2P2S method provided a segmented view with the associated cloud management capability as defined by NIST CCRA, which encompasses all functions related to services required for the domain application’s management and operations. In the case of the IDT4CTI case study, this has resulted in a public cloud platform management view, containing the use cases functionality to configure and administrate the public cloud provider which have later given origin, in the logical architecture, to black box packages (P1 and P2 in figure 6). It only remained to the product owner the discussion of the actors required to operate, configure, or administer these software packages, such as administrators, program-

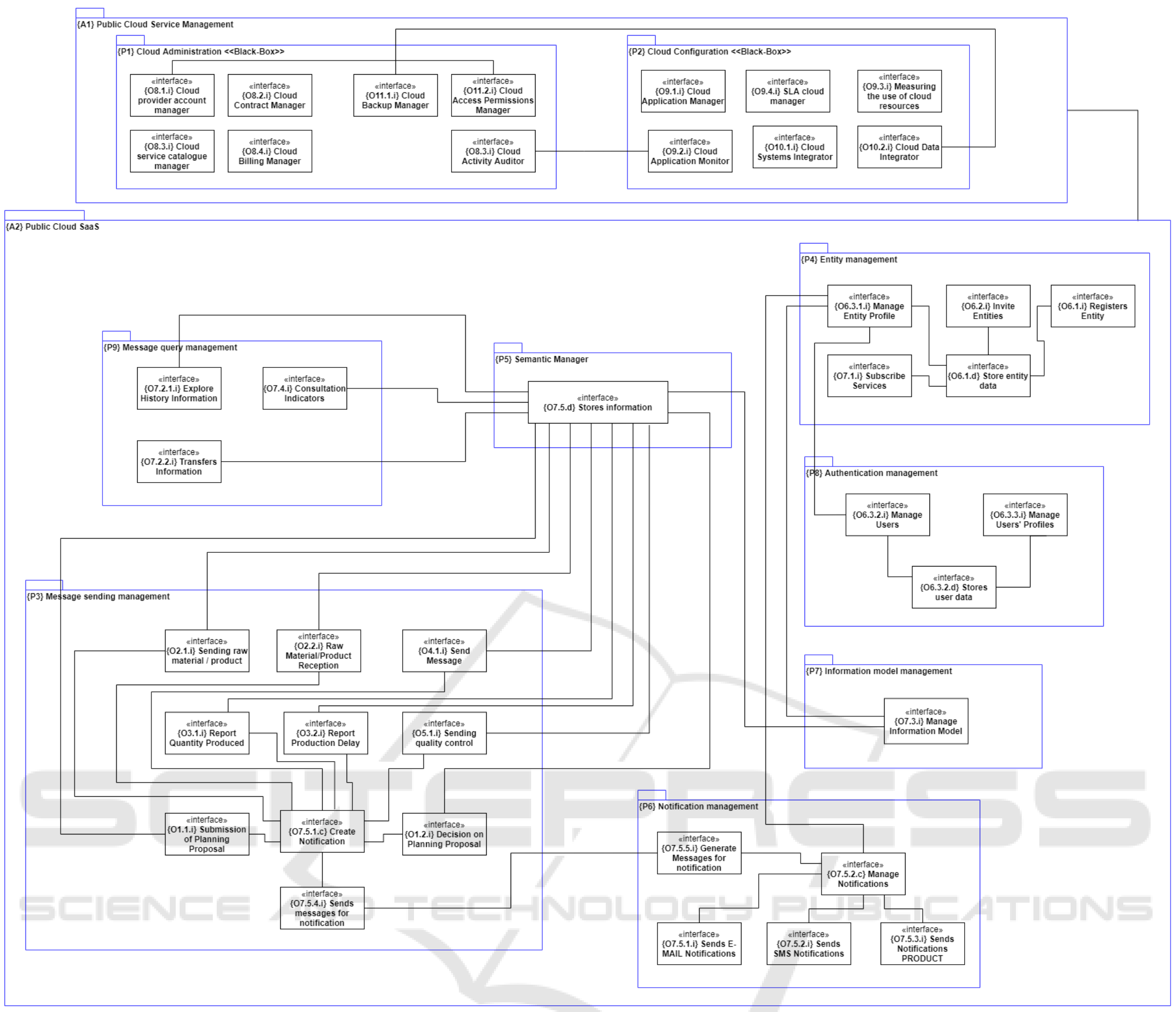

{A1} Public Cloud Service Management
{P1} Cloud Administration <<Black-Box>>
«interface» {O8.1.i} Cloud provider account manager
«interface» {O8.2.i} Cloud Contract Manager
«interface» {O11.1.i} Cloud Backup Manager
«interface» {O11.2.i} Cloud Access Permissions Manager
«interface» {O8.3.i} Cloud service catalogue manager
«interface» {O8.4.i} Cloud Billing Manager
«interface» {O8.3.i} Cloud Activity Auditor
{P2} Cloud Configuration <<Black-Box>>
«interface» {O9.1.i} Cloud Application Manager
«interface» {O9.4.i} SLA cloud manager
«interface» {O9.3.i} Measuring the use of cloud resources
«interface» {O9.2.i} Cloud Application Monitor
«interface» {O10.1.i} Cloud Systems Integrator
«interface» {O10.2.i} Cloud Data Integrator
{A2} Public Cloud SaaS
{P4} Entity management
«interface» {O6.3.1.i} Manage Entity Profile
«interface» {O6.2.i} Invite Entities
«interface» {O6.1.i} Registers Entity
«interface» {O7.1.i} Subscribe Services
«interface» {O6.1.d} Store entity data
{P9} Message query management
«interface» {O7.2.1.i} Explore History Information
«interface» {O7.4.i} Consultation Indicators
«interface» {O7.2.2.i} Transfers Information
{P5} Semantic Manager
«interface» {O7.5.d} Stores information
{P8} Authentication management
«interface» {O6.3.2.i} Manage Users
«interface» {O6.3.3.i} Manage Users' Profiles
«interface» {O6.3.2.d} Stores user data
{P3} Message sending management
«interface» {O2.1.i} Sending raw material / product
«interface» {O2.2.i} Raw Material/Product Reception
«interface» {O4.1.i} Send Message
«interface» {O3.1.i} Report Quantity Produced
«interface» {O3.2.i} Report Production Delay
«interface» {O5.1.i} Sending quality control
«interface» {O1.1.i} Submission of Planning Proposal
«interface» {O7.5.1.c} Create Notification
«interface» {O1.2.i} Decision on Planning Proposal
«interface» {O7.5.4.i} Sends messages for notification
{P7} Information model management
«interface» {O7.3.i} Manage Information Model
{P6} Notification management
«interface» {O7.5.5.i} Generate Messages for notification
«interface» {O7.5.2.c} Manage Notifications
«interface» {O7.5.1.i} Sends E-MAIL Notifications
«interface» {O7.5.2.i} Sends SMS Notifications
«interface» {O7.5.3.i} Sends Notifications PRODUCT

Figure 6: IDT4CTI Raw Object UML Diagram.

mers, testers or deployers.

In the following phase, the method classifies the domain application and the cloud management resources and actors based on the delivery model to validate roles, raise more functional requirements, and model judgments about cloud provider technology dependency. In our case study, the application was classified as a SaaS delivery model, and the cloud management was classified as a PaaS delivery model. This allowed identifying the actors and correspondent functionality which have later given origin, in the logical architecture, to the identification of design decisions concerning the data objects, packaged as services and picked from the provider's service catalog (database services).

This approach simplifies the procedure for obtaining cloud requirements while also enriching the context of use case diagrams, containing descriptions of the deployment model and service delivery model, which can then be utilized as input to an MDD technique to build a cloud architecture that is consistent with the product owner's decisions and comprehensibility of requirements for cloud service consumers. The used MDD technique (the 4SRS method) ensures that models of user requirements are transformed into models of system requirements while altering the relevant specifications. The adoption of the SOA paradigm enables the managed system to respond flexibly to changes in the business process through reconfiguration and the addition of new functionalities. It also organizes the various objects into services, which can lead to differing perspectives on the final product. The logical architecture's object diagram thus ensures the specification of the system platform, which is based on a service-oriented platform (SOA) and cloud computing.

6 CONCLUSIONS

In this paper, we use NIST Cloud Computing Reference Architecture in eliciting requirements by employing a new approach (the 2P2S technique), which enables the use of an existing Model-Driven Design method (the 4SRS method) to derive logic architectures for cloud-based solutions.

The result is to define the requirements in such a way that they are understood by the cloud service consumers (the owner of the product) because it uncovers more information as if using only the use case model of the application domain. The discussion of the deployment model uncovers functional requirements for cloud management and helps distribute the domain application functional requirements up to a multi-cloud concept. Moreover, the service model discussion uncovers more role actors for the provider and the consumers and enriches use cases with more descriptions that may lead to new design decisions or non-functional requirements. The requirements elicitation involved a detailed analysis of the NIST CCRA, where a set of key use cases that capture the functional requirements of cloud services were presented. The use of this reference architecture was useful in the requirements elicitation phase, since it allows proper functional refinement of the domain application and the cloud services, being a conceptual model that constitutes an effective tool to discuss the requirements, structure, and operation of the cloud. The 4SRS had never been used for this particular context. However, it has proved very useful since it formalized and oriented the development of the architecture.

Some subjectivity may result from the application of the 4SRS, so to minimize it, the model is iterated and revised until a sufficient model is obtained to serve as the basis for the technical modeling of the architecture, allowing a linkage between the semantic component, the logical architecture and the cloud architecture. The correct derivation of system requirements from user requirements is an important topic in requirements engineering research, where our approach contributes to the adoption of emerging cloud paradigms in this early requirements phase, assuring that the system requirements are based on effective client needs.

REFERENCES

Iankoulova, I., D. M. (2012). Cloud computing security requirements: A systematic review. *In 2012 Sixth International Conference on Research Challenges in Information Science (RCIS) (pp. 1–7). IEEE.*

Klems, M., Nimis, J., and Tai, S. (2009). Do clouds compute? a framework for estimating the value of cloud computing. volume 22, pages 110–123.

Koitz, I. T., Seyff, N., and Glinz, M. (2013). How cloud providers elicit consumer requirements: An exploratory study of nineteen companies. *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 105–114.

Liu, F., Tong, J.; Mao, J.; Bohn, R.; Messina, J.; Badger, M.; Leaf, D. (2011). *NIST Cloud Computing Reference Architecture*. NIST Special Publication 500-292.

Machado, R.-J., Fernandes, J., Monteiro, P., and Rodrigues, H. (2005). Transformation of uml models for service-oriented software architectures. In *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS 05)*, pages 173–182.

Pichan, A., Lazarescu, M., and Soh, S. T. (2015). Cloud forensics: Technical challenges, solutions and comparative analysis. *Digital Investigation*, 13:38–57.

Reed, P. (2002). Reference architecture: The best of best practices. *The Rational Edge*.

Reese, G. (2009). *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud*. Theory in Practice (O'Reilly). O'Reilly Media.

Repschlaeger, J., Wind, S., Zarnekow, R., and Turowski, K. (2012). A reference guide to cloud computing dimensions: Infrastructure as a service classification framework. In *2012 45th Hawaii International Conference on System Sciences*, pages 2178–2188.

Rimal, B. P., J. A. K. D. G. Y. (2011). Architectural requirements for cloud computing systems: An enterprise cloud approach. volume 9, pages 3–26.

Robertson, J. and Robertson, S. (2000). Volere requirements specification template.

Schrodl, H. and Wind, S. (2011). Requirements engineering for cloud computing. *Journal of Computer and Communication*, 8:707–715.

Souza, E., Moreira, A., and Goulão, M. (2019). Deriving architectural models from requirements specifications: a systematic mapping study. *Information and Software Technology*, 109.

Zalazar A.S., Ballejos L., R. S. (2017). *Analyzing Requirements Engineering for Cloud Computing*. Ramachandran M., Mahmood Z. (eds) Requirements Engineering for Service and Cloud Computing. Springer.

Zardari, S. and Bahsoon, R. (2011). Cloud adoption: A goal-oriented requirements engineering approach. *Proceedings - International Conference on Software Engineering*.