Universidade do Minho
Escola de Engenharia

Nuno Alexandre Castro Ferreira

From Information Systems Requirements to Service-Oriented Logical Architectures

Tese de Doutoramento
Programa de Doutoramento em Informática
das Universidades do Minho, de Aveiro e do Porto

Universidade do Minho

Trabalho efectuado sob a orientação de
Professor Doutor Ricardo J. Machado
Professor Doutor Dragan Gašević

Novembro de 2012

Universidade do Minho, **30** / **11** / **2012**

Assinatura: _____

Agradecimentos

"If I have seen farther, it is by standing on the shoulder of giants."

Isaac Newton

Letter to Robert Hooke, 1676

Em primeiro lugar, gostaria de agradecer à minha família, por me terem dado todas as condições e apoio necessários à execução deste trabalho ao longo dos anos. Susana, Margarida, Pedro e Manel, Pais, a todos obrigado.

Aos meus orientadores e amigos, Prof. Doutor Ricardo J. Machado e Prof. Doutor Dragan Gašević, dois gigantes incansáveis que sempre estiveram por perto quando foi preciso.

À i2S, empregadores, motivadores e amigos, que criaram o desafio do projeto de doutoramento, que criaram o contexto que deu origem ao ISOFIN, que suportaram tudo e na qual foi executado o trabalho apresentado.

Um obrigado em especial ao Nuno Santos e à Paula Monteiro, que sempre me acompanharam, motivaram e ajudaram.

Apoio Financeiro

O projeto ISOFIN foi realizado ao abrigo do Quadro de Referência Estratégico Nacional, com a referência QREN 2010/013837.

A Fundação para a Ciência e Tecnologia, financiou o presente trabalho, ao abrigo da Bolsa de Douramento em Empresas com a referência SFRH/BDE/33297/2008.


i2S
INSURANCE KNOWLEDGE


Universidade do Minho


isofin


Centro de Computação Gráfica


FCT
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA


inov
inesc • inovação


i.zone
knowledge•systems
people processes technology


maisis
information systems


kc
Knowledgebiz
consulting


SEMAG
software engineering and management group

Abstract

In a world where there are no certain and doubt prevails, it is not possible to take a grounded decision about the construction of an artifact able to deliver a set of functionalities of which only the intended output is known.

Based on the previous premise, this work details a method, and a process, for creating an architectural representation of a software solution. This representation starts in a very high-level of abstraction and end with a visual representation of the architecture of the desired system. This system is able to implement the initially desired functionalities. The entire process is continuously assessed and monitored.

Common approaches for dealing with the lack of information to create an architectural representation of a system do not regard real world activities nor support successive model derivation. Our approach is based on a V+V-Model approach that starts in the real world activities (process-level) and ends in the logical architecture of the intended system (product-level) and thus taking the alignment of real world with intended system into consideration.

We use the basic V-Model approach for creating an information system representation. The approach starts with an initial V-Model that evolves into a second V-Model. The V-Model transition is sustained in a set of rules and transformations. In the entire process are produced a set of modeling artifacts that are delivered to the development teams with the purpose of helping them in the development efforts.

The V+V-Model approach and the adopted models alongside the entire approach are validated using the architecture validation method ARID and applied in a real industrial case study, the ISOFIN Project, framed within a business model conditioned by cloud-related and service-oriented usage scenarios. This project has the purpose of delivering a set of services that allow the interoperability between financial domain related entities (bank and insurance companies).

Keywords: Logical Architectures; Information Systems Architecture; Architecture Assessment Methods; Requirements Elicitation.

Resumo

Num mundo onde não há certezas e as dúvidas imperam, não é possível efetuar decisões fundamentadas e baseadas na construção de artefactos que, por si, sejam capazes de disponibilizar funcionalidades, das quais só se conhece o resultado esperado.

Baseado na premissa anterior, este trabalho detalha um método e um processo para, construir uma representação arquitetural de uma solução de software. Esta representação inicia-se num nível de abstração muito elevado e termina com uma representação visual da arquitetura do sistema desejado. Este sistema é capaz de implementar as funcionalidades inicialmente desejadas e é passível de ser, durante a sua construção, avaliado e simultaneamente monitorizado no que diz respeito à correção da sua execução.

As abordagens existentes para lidar com a falta de informação para criar uma representação arquitetural de um sistema não consideram as atividades do mundo real nem suportam derivações sucessivas de modelos de forma a enriquecer, em cada interação, o modelo resultante. A nossa abordagem é baseada no modelo V+V que se inicia nas atividades do mundo real (a nível de processo) e termina na arquitetura lógica do sistema desejado (a nível do produto), conseguindo-se desta forma alinhar o mundo real com o sistema desejado. Usamos na base do modelo V+V, uma abordagem baseada no modelo em V, de forma a criar a representação do sistema de informação. Esta abordagem inicia-se com o modelo em V inicial que transita para o segundo modelo em V. Esta transição é suportada num conjunto de regras e transformações. Durante o processo são construídos um conjunto de artefactos de modelação que são entregues às equipas de desenvolvimento com o propósito de os ajudar nos seus esforços de desenvolvimento.

A abordagem do modelo V+V assim como os modelos usados para a sua construção são validados usando o método ARID de validação de arquiteturas e a sua aplicação é realizada num caso industrial real, o projeto ISOFIN. Este projeto é condicionado por um modelo de negócio sustentado em cloud e tem como base a construção de serviços que possibilitem cenários de interoperabilidade no domínio financeiro (banca e seguros).

Palavras-chave: Arquiteturas Lógicas, Arquiteturas de Sistemas de Informação, Métodos de Avaliação de Arquiteturas, Levantamento de Requisitos.

Table of Contents

Acronyms

4SRS	Four-Step-Rule-Set
ARID	Active Reviews for Intermediate Design
IBS	ISOFIN Business Service
ISOFIN	Interoperability in Financial Software
MOF	Meta-Object Facility
NBS	Native Business Service
OMG	Object Management Group
SBS	Supplier Business Service
SPEM	Software and Systems Process Engineering Meta-Model
UML	Unified Modelling Language

Table of Figures

List of Tables

Chapter 1

Introduction

Chapter Contents

1

Introduction

"Ignorance more frequently begets confidence than does knowledge: it is those who know little, and not those who know much, who so positively assert that this or that problem will never be solved by science."

Charles Darwin

This chapter starts by introducing the motivation to the writing of this thesis. We will present an overview of the problem; the research goals and the research strategy used. We will then introduce the contributions present in this work made to the universal body of knowledge. We conclude with the structure of the thesis.

1.1 Motivation

The development of service clouds emerges with the promise to facilitate collaboration between independent parties, affecting thereby how organizations manage their business processes, namely with respect to inter-organizational interactions. In a business ecosystem there are interactions that rely on the availability of services to assure its existence, business continuity and evolution. The

interactions between these services gave origin to a new level of service providers for the community – service clouds, which allow companies to easily scale their demand for applications, resources and services. Services clouds are increasingly gaining importance in the current IT paradigm. As an example, according to IDC, the "*revenue from public IT cloud services exceeded $21.5 billion in 2010 and will reach $72.9 billion in 2015, representing a compound annual growth rate (CAGR) of 27.6%.*" (IDC).

It is nowadays a commonly acknowledged fact that we live in an era of services that reside *somewhere* (in the cloud) and when put together create cloud-enabled solutions. This new found paradigm derives into the trend of having software solution (henceforward referred to as "product") in the cloud as a business advantage, despite all the known problems and disadvantages, becoming a true cloud computing business scenario. These scenarios are changing the business world, triggering companies to move towards the cloud, allured by the economic advantages promised by the model. The cloud paradigm has underneath a financial model, indexed to the real resource usage per user, sustaining the rationale of investment decisions and at the same time, minimizing the risk associated to new projects. At the same time, it is continuously promoting the business world, by the onset of new partnerships, new players and new business offers (Roberto, 2010).

The cloud paradigm is not restricted to finance or trends. It also contributes with scalability, and also with effective cost reductions, increase of processing power and storage. Every time more organizations seek the announced benefits of flexibility, ease and speed of access, elasticity and competitiveness made available by the cloud. In the reverse and at the same time, organizations also demand from the cloud paradigm, more security, integration, quality and return of the investment of the product, creating each time more opportunities for improving the cloud computing model.

Cloud computing was firstly introduced by a group of researchers at MIT in 1996 (Gillett & Kapor, 1996). It is used in apparently distinctive situations (Reese, 2009; Velte, Velte, & Elsenpeter, 2010), but its origin was in the area of software

architectures where the internet is commonly referred to as a cloud (Reese, 2009). Figure 1 exemplifies a typical cloud architecture.

Moving into the cloud

Our motivation for the elaboration of this work was sustained in the need of defining a project – the ISOFIN project – (later detailed), with the purpose of creating a set of services that reside in a cloud infrastructure and contribute for the interoperability of services in the financial domain, insurance companies and banks. The need for moving into the cloud was there.

The first step for moving to a cloud paradigm is by understanding and sharing its definition with all the stakeholders. Despite being unknown to some, a cloud definition that begins to gain consensus by IT professionals is the one stated by the National Institute of Standards and Technology (NIST). According to NIST, cloud computing "*is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.*" (NIST, 2009).

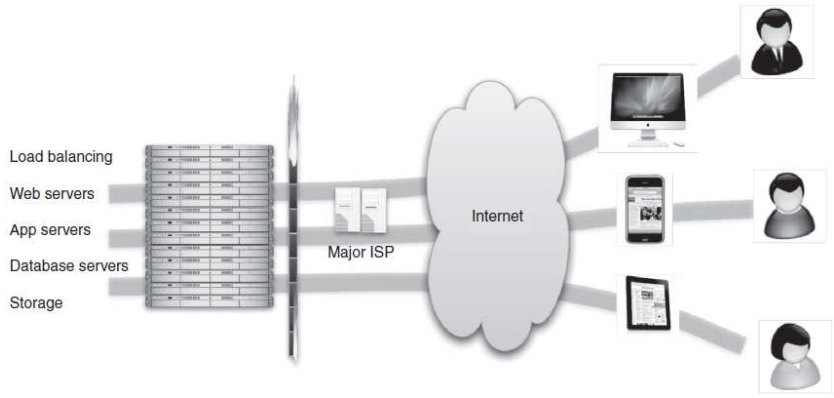

Figure 1: Example of a cloud architecture

The second step of moving towards the cloud, after the business opportunity was understood by the stakeholders, is to clearly state the intended business model of the final product.

The problem that created it all – what is the business model?

In this second step problems began. The first problem arose in the project team with the definition of the business model for the intended project outcome, based on a cloud. Despite all the team knowledge on traditional (non-cloud) applications, the definition of the business model derived from moving the applications (software) towards the cloud was unable to reach consensus in what concerns the requirements definition for the final purpose of the transition. This was a second problem – what are the requirements for the cloud application? Then, a third problem emerged for the project team from the previous: what are the processes that support the final solution (the software product)?

There is a clear misunderstanding of the final solution and traditional (product-level) requirement elicitation techniques are unable to clarify the problem. In this work we propose the use of a process-level perspective for the requirements definition and design of the logical model of the information system architecture. This is built upon the premise that such an approach contributes to a more accurate definition of product requirements and understanding of the project scope (Ferreira, Santos, Machado, & Gasevic, 2012a).

1.2 Research Goals and Strategy

This thesis aims to contribute for the increase of the universal body of knowledge by defining a process able to deliver an artifact that was built with the intention of solving a particular problem and according to the initial specifications of stakeholders' requirements.

The specific goals of this thesis are:

- To define and detail a software development process able to create a context for product design especially focusing on situations where the requirements are not properly defined nor clear for all the involved stakeholders;

- Detail a method for deriving an information systems architecture from a common understanding of the requirements based on business needs;
- Define transition rules from the information system architecture to product requirements to establish traceability between the information system architecture and the intended service-oriented (product) final logical architecture and, at the same time, provide a requirements model suitable for deriving that architecture;
- Create an assessment method for the entire process based on well-established assessment methods, guaranteeing that the created process produces the desired valid and verified outputs.

This work considers a real industrial case study, the ISOFIN Project (ISOFIN Project Consortium, 2010) which is introduced in detail in Chapter 2.

The thesis author role in the ISOFIN project was as project leader. The project leading company is i2S (www.i2s.pt), a Portuguese software company dedicated to software development for the insurance industry, founded in 1984, and with clients in Portugal, Spain, France, Poland, Angola, and Mozambique, among others.

In the thesis scope there is a need to conduct research with the dual purpose of (one) properly following the procedures and (two) increase the effectiveness of the work to be done and increasing the output in term of benefits of this thesis to the ISOFIN project.

The research process

According to Yin (Yin, 2003), five components of a research are important:

(i) the research questions;

(ii) the study propositions;

(iii) the unit of analysis;

(iv) the logic linking of data to the propositions;

(v) the criteria for interpreting the findings.

The following paragraphs give more detail on these points.

Research questions

According to Yin "*Defining the research questions is probably the most important step to be taken in a research study,(…)*"(Yin, 2003). The research questions are fundamental to defining the roadmap for the research efforts and provide an end for which the researcher has to provide the means.

From the initial analysis made on the ISOFIN project, there were some questions that arose from the particular context from where the project was executed:

- How can we achieve a proper product specification if there is no agreed upon or defined context from the requirements?
- How to create proper requirements if there is no intended business scenario for the product execution?
- How to translate the known requirements (intended process to be executed) into the final product architecture?

Proposition

Once again, according to Yin "*(…) each proposition directs attention to something that should be examined within the scope of study*" (Yin, 2003).

The scope of our study regards logical architectures. More precisely, logical architectures that allows defining the process (process-level) architectural perspective and also the product (product-level) perspective of the intended systems. The attention of this study should be directed to the derivation of such logical architectures, regardless of perspective, and how to transform one into the other.

To do so, it is important to analyze the proper artifacts (models) that make up the process representation in each of the intended perspectives (process and product/information systems and service-oriented).

As a proposition, we consider that it is crucial to our project a proper understanding of the models that (first) make up the process of defining the logical architecture of the intended system and (second) the models that assure the transition between perspectives.

Unit of analysis

As a unit of analysis, we have defined the ISOFIN project. Yin states "*the fundamental problem of defining what the "case" is (…)*" (Yin, 2003). The ISOFIN project is suitable to act as a case study in our analysis and provide the necessary problems and at the same time, the context for creating the solutions.

Linking data to propositions and criteria for interpreting results

In what concerns linking data to propositions and defining the criteria for interpreting the results, and according to Yin, "*The fourth and fifth components have been the least well developed in case studies*" (Yin, 2003).

In our presented case study, the problems began when the consortium responsible for the project execution could not agree on the business model that would support the applications (product) that was intended to be developed. The only agreed information was the major activities that should be supported by the product.

For the purpose of aligning "*artifacts (models) that make up the process representation*" (Yin, 2003) with the ISOFIN project data, we are required to establish a proper path from activities known to the stakeholders to the intended and final product logical architecture. To do so, we defined Organizational Configurations, stereotyped sequence diagrams, use case models, a new process-oriented 4SRS

method, logical architecture diagrams, *mashed* use case models, and a whole assessment process based on ARID.

These models and conceptions, linked to the project data and real problems, can be interpreted as design artifacts, uncommon in case studies, but easily framed within Design Science Research (Hevner & Chatterjee, 2010; Vaishnavi & Jr., 2008).

The Design Science Research is, according to Hevner and Chatterjee (2010), *"a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem"*. Having this statement in mind, we establish the link with the case study and the analysis of the results. By looking at the applicability of the designed artifacts that make up our approach in the real industrial case study, it is possible to make a proper assessment of our research contributions. These contributions concern the development of a process to derive a service-oriented logical architecture (product level) from information system requirements (process-level).

In Figure 2, (Vaishnavi & Jr., 2008) details the general design lifecycle that provides a way of explicate the knowledge that is generated in the context of design and its connection with the design science research.

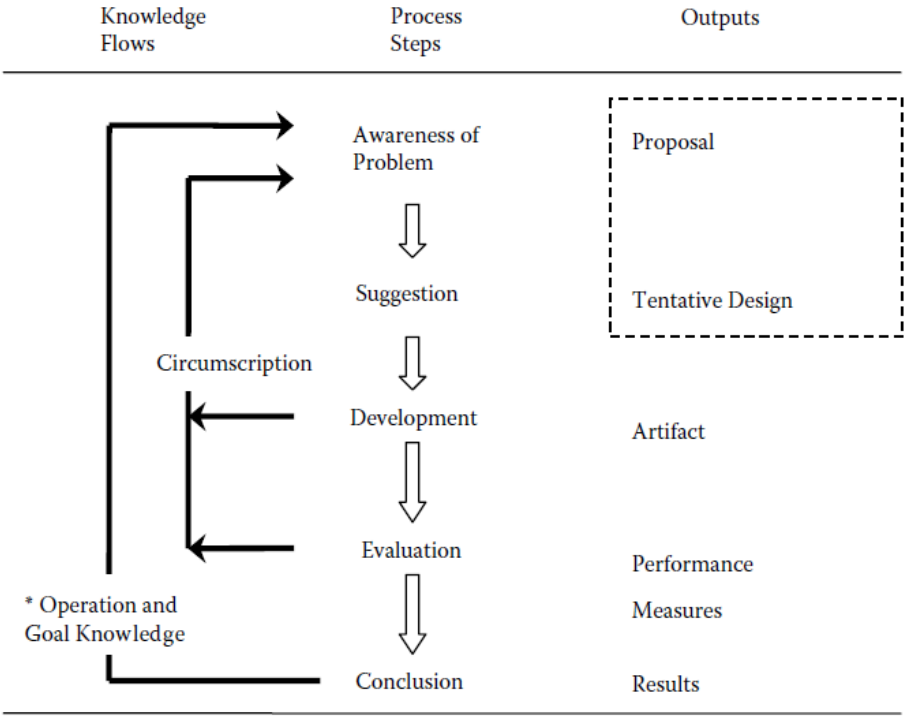

Figure 2:Methodology of Design Science Research (Vaishnavi & Jr., 2008).

The awareness of the problem is given in the initial ISOFIN project consortium meetings, where the first difficulties for establishing the intended business model were observed. Next, the preliminary suggestions for the problem solutions are provided. These are driven by previous knowledge, theories related to the problem scope, or even developed using a research methodology. In the ISOFIN project, all of the previous were used: the previous knowledge from the 4SRS method (chosen method for deriving logical architecture – later detailed), theories addressing eliciting requirements from undefined contexts, and the case study itself.

After the tentative design had been agreed on, the creative development started. The design was refined and the actual artifact (the V+V Model) was produced after a series of interactions, constructing one model after another, deriving information from the previous to build the latest.

The development stage led to an artifact prototype that was validated using empirical methods (an evaluation approach in design science research (Hevner & Chatterjee, 2010)) to establish how well the produced artifact works.

The process followed through and, at the end, the conclusions where driven and used in the ISOFIN project and others projects ever since. The ISOFIN project was developed in cooperation with the Department of Information Systems at Universidade do Minho. This PhD research was made in both the Universidade do Minho and i2S Informática Sistemas e Serviços S.A..

In recent years, the information systems research community has seen an upsurge of interest in design science research. When Hevner et al. (Hevner & Chatterjee, 2010; Vaishnavi & Jr., 2008) described a conceptual framework and guidelines for performing design science research in information systems, another question arose regarding what the Hevner refers to as performing design science research in information systems calls for "communication of research." In academic environments, the primary means of communicating one's research is to publish the work. In the next section, we present our contributions and also the publications that were made related to each of the described contributions.

1.3 Contributions

This thesis aims to contribute to the increase of the universal body of knowledge in the knowledge area of software engineering by defining a new process able to deliver a derived product architecture based on the initial definition of the requirements. Aligned with the generic research goals, the research questions and as a result of the research method, we were able to create the following main contributions:

Contribution 1: Process-level 4SRS method

This method allows for deriving a process-level logical architecture from the initial activities that make up the requirements for a given information system. The method relies on tabular transformations to create a representation (model) of the intended system, taking as input the requirements in form of use cases. This contribution is detailed in (Nuno Ferreira, et al., 2012a).

Contribution 2: The V-Model approach

The V-Model approach is used to create the initial representation of the context for the activities to be executed, the activities representations, and the processes as a set of activities to create the requirements, the 4SRS method to derive a logical architecture from those requirements and the assessment of the architecture. This contribution is detailed in (Ferreira, Santos, Soares, Machado, & Gasevic, 2012).

Contribution 3: The V+V-Model approach

This V+V-Model regards creating context for product design, by executing a first V-Model (process-level) and then, by executing a set of transition rules, apply a second V-Model (product-level) to derive the logical architecture of the intended system to be developed. The entire process is assessed using ARID to assure that the models and the methods are aligned, properly used and produce the desired output. This contribution can be seen in (Ferreira, Santos, Machado, & Gaševic, 2013).

Other contributions, like the (supposed) initial problem definition that gave origin to this work can be seen on (Ferreira, Machado, & Gašević, September, 2009). More details on the contributions can be read on (Ferreira, Santos, Machado, Fernandes, & Gasevic, 2013).

1.4 Structure of this Thesis

This document is structured in six chapters. Each chapter is preceded by a cover page with an index, with the purpose of assisting the perception of the entire chapter and a brief summary of the chapter. After that, the main sections of the chapter are systematically an introduction at the beginning, a conclusion at the end, and the relevant sections for the chapter's theme in the middle.

The six chapters of this document and their main content are:

Chapter 1: Introduction

This chapter introduces the motivation for the research, the areas of research, the research goals and research strategy, major contributions, and the thesis structure.

Chapter 2: Current State of Eliciting Requirements for Information Systems

In this chapter we introduce the ISOFIN project as the real industrial case study in which the design research validates the designed artifacts. The subject of this chapter is also the overview of requirements elicitation for information systems and the cloud-specific context that frames the creation of the activities that will result in the processes to be computationally supported.

Chapter 3: Modeling Information and Software Systems

This chapter presents the V-Model approach for creating context for product design by derivation of information systems' architectures. It starts by framing the need for creating the high level interactions between the domain entities to assist in the creation of the intentional interaction manifestations that enact the domain representation.

Chapter 4: Yet another 4SRS

This chapter introduces and details the process-level 4SRS and presents an overview of the traditional product-level 4SRS. The process-level 4SRS method is able to capture the major activities that compose information system requirements and derive the information system logical architecture representation.

Chapter 5: Process- and Product-level Logical Architectures

This chapter presents the V+V-Model approach, based on the V-Model approach and with a set of defined transition rules from one model to the other. This composed approach is able to derive a service-oriented logical architecture (product-level) from an information system logical architecture (process-level). To assess the entire approach, the chapter introduces an adaptation of the architectural assessment method, ARID, adapted to the V+V-Model approach.

Chapter 6: Conclusion

This chapter presents the conclusions about the present work. It also promotes a discussion the future word and details some research questions that the authors' would like to address. This chapter also promotes a discussion on the results of the applicability of the V+V-Model approach to the ISOFIN project.

Chapter 2

Current State of Eliciting Requirements for Information Systems

Chapter Contents

2

Current State of Eliciting Requirements for Information Systems

It's more important to understand the fundamental truth behind a given problem than to methodically describe its conclusions (consequences)

Odracir

This chapter introduces the ISOFIN project and the problem that arises in contexts with insufficient information to create a logical representation of information systems architecture. Specifying functional requirements brings many difficulties namely when regarding the cloud services. During the analysis phase, the definition of the process level requirements (information systems) may not be fully accomplished if there is no context for starting uncovering those requirements. The process-level requirements must be aligned with the product level requirements (service-oriented software).

2.1 Introduction

One challenging activity in the application development lifecycle in a development project is the transformation of a given requirement specification into a view of the system represented by a logical architecture (Yue, Briand, & Labiche, 2011). Later development stages are also complex, but there is a better understanding, more methodologies and tools supporting them. Poorly defined requirements are one of causes of project failure (Cerpa & Verner, 2009). This chapter proposes a new method of deriving activities that lead to requirements and use cases of the system, thus proving a way of eliciting requirements based on common domain activities. The common domain activities will be based on cloud-computing (NIST, 2009) related usage scenarios.

Architecture design of a system encompasses dealing with several design objectives at the same time due to the nature of requirements specifications. Since new requirements emerge anytime, before, during and even after the development stage, we can say that designing software architectures is an activity performed several times during the development (Bosch, 2000). Dealing with change and open requirements is a problem that many are trying to minimize at some degree, as seen on Agile Methods, Open Unified Process or Rational Unified Process, amongst others.

There are several ways of describing what users want from a given system, like use cases, user stories or textual descriptions (Ambler & Lines, 2012; Dijkman & Joosten, 2002b). In the case where none of the before mentioned methods fulfill the intended purpose for describing a system's intended functionalities or lack information for providing the necessary detail as input, we propose looking at the activities that a given system is supposed to execute and frame them with the system under study specific context to derive the necessary information for building its initial requirements and later achieve an architectural representation of it.

This section presents PL.AC.E (Process-Level ACtivities Elicitation), a lightweight method for eliciting requirements, in a process-level perspective, through the discovery and classification of canonic domain activities. The method is domain-

specific because it starts by focusing the analysis on a given ICT-related domain and framing the scope of analysis to that domain, in an attempt to understand it. A multi-level process perspective is adopted since we are focusing our analysis on activities in order to enable definition of high-level business processes. It is called multi-level because the method's core encompass' the creation of an n-level matrix of ICT-related canonic activities. The method also promotes a classification of the activities. Each step of the method is described in this chapter and examples are given for better understanding.

Alongside with the PL.AC.E method, we present an overview of the technological target for the activities implementation: a cloud terminology overview. This overview is useful in the context of placing the necessary mindset for the PL.AC.E execution and determine the high-level interactions that will be part of the business model of the intended final software solution.

As far as definitions concern, in the scope of this section, an activity is a set of human and/or machine actions that take place in a given context for the fulfillment of a specific objective. A process, according to, and in a business context, is a set of activities executed to achieve a given business goal and where business process, human resources, raw material, and internal procedures are combined and synchronized towards a common objective. The definition of "level" goes back to Hammer (Hammer, 1997). He uses the term to describe qualitatively different entities, with a hierarchical relationship. Our usage relates to the different entities that are combined together to give origin to the elicited activities. Multi-level refers to the interactions we make between the different levels (entities). Another useful definition to have in mind while reading this document is of "model". For us, a model is a formal specification (machine readable) and explicit of the mental representation that results from the study of a given domain.

2.2 Problem Overview and Related Approaches

Before any software development is carried, there are three major phases that must occur: domain engineering, requirements engineering and software design. The domain engineering phase concerns all the necessary work for understanding and characterizing the domain under study, resulting in a model of the domain. Domain Analysis is part of Domain Engineering. The requirements engineering phase results in a requirements model that specifies how we expect the resulting artifact should be. This model is in direct relation with the domain model. Finally, the software design encompass' the system architecture, the code organization, components, modules and code design, all aligned with the requirements model.

The problem begins when there is not enough understanding of the domain under analysis. If there is no documentation, the problem-space relates to a new situation, the stakeholders do not want or do not know how to describe the intended system, it is not feasible to properly create a requirements model aligned with a non-existing domain model. An analysis of the system or of the domain needs to be carried out for accomplishing a full understanding of the requirements we want to fulfill.

The PL.AC.E method focuses on delivering a set canonic ICT-related candidate activities for building a domain model from a process-level perspective of the system under study. The method is intended to work in specific environments where standard domain analysis methods cannot gather enough information to deliver coherent domain models.

There are some methods, methodologies and frameworks that relate directly or indirectly to activity elicitation. The next paragraphs briefly present some of them that influence this work proposed approach. It is not our intention to present an exhaustive survey of the system analysis or domain analysis literature, but instead focus on the ones that influence the PL.AC.E method.

The Work System Method (Alter, 2002) allows for a better understanding and analysis of IT and non-IT systems present in organizations. This method presents a combined static view of the current (or proposed) system and a dynamic view of the system evolution over time. It takes in account planned and unplanned changes of the system over time. A work system's goal is to produce products and/or services and make them available to customers.

The Work System Framework is based on the Work System Method and was designed to help to understand IT dependent systems. Its elements are graphically represented in Figure 3: Processes and Activities, Participants, Information, Technologies, Products and Services, Customers, Infrastructure, Environment and Strategies. All of them should be included in the analysis conducing to the understanding of a specific system.

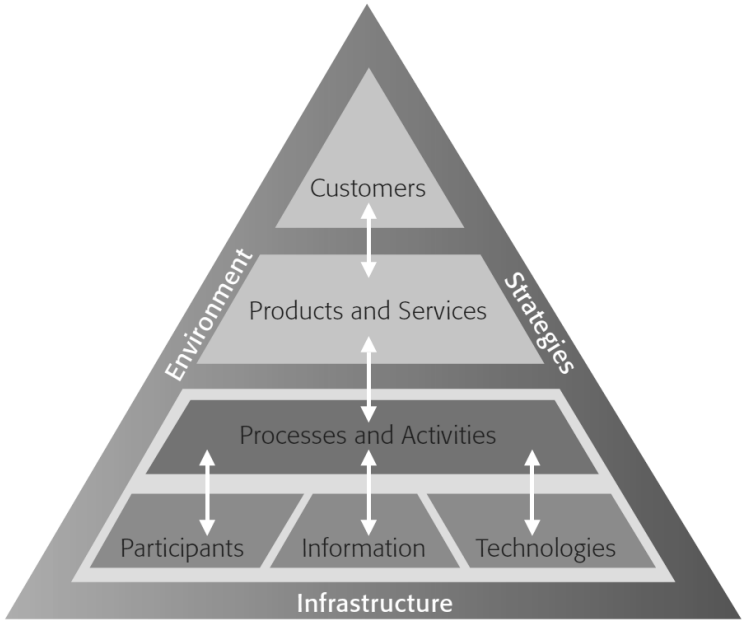

Figure 3: The Work System Framework

Complementing the Work System Framework, the Service Value Chain Framework (Alter, 2008) adds activities and responsibilities of service providers and customers, associated with services. In this context, services are defined as *"the application of specialized competences (knowledge and skills) through deeds, processes, and performances for the benefit of another entity or the entity itself"* (Vargo & Lusch, 2004). In this line of though, the Work

System Lifecycle Model appeared. This adds to the static views presented in both the Work System Framework and in the Service Value Chain Framework the necessary dynamics to represent how work systems change over time.

The Soft Systems Methodology (SSM) (Checkland, 1981, 1985, 2000) is a domain-independent analysis methodology designed for tackling problematic situations where there is neither a clear problem definition nor a solution. SSM can widely be described by its “seven-stage model” as seen on (Checkland, 1985, 2000).

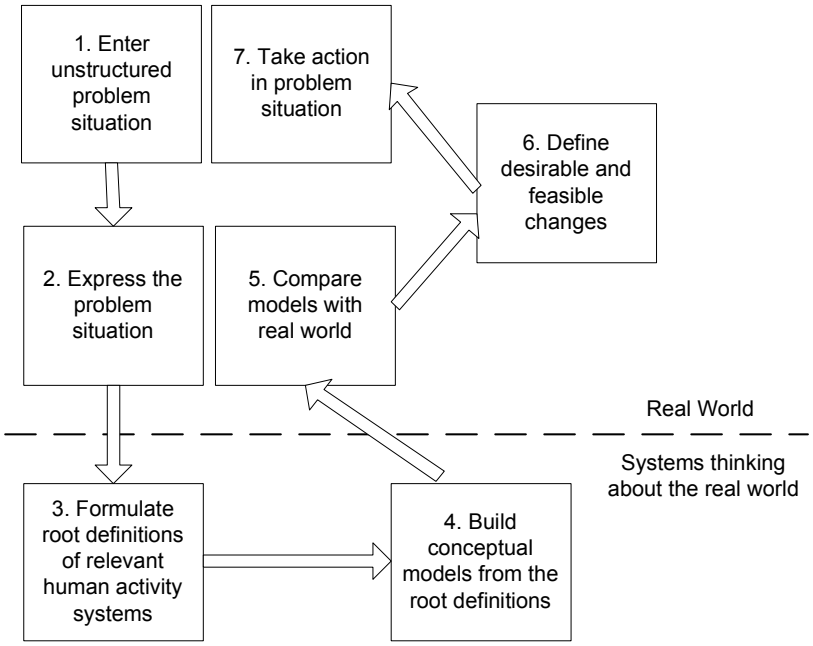

Figure 4: Soft Systems Methodology (adapted from (Checkland, 2000))

The SSM model can be seen in Figure 4 and the stages are:

(i) Identify the problematic situation where the intervention is going to happen;

(ii) Build an interpretative representation of the situation;

(iii) Build “root definitions” or define the key processes that need to occur in the desired solution;

(iv) Build a conceptual model of the change system from the key processes;

(v) Establish a comparison between the model and the real world;

(vi) Define a set of changes to be implemented;

(vii) Take the corresponding actions in the problem.

The methodology uses system models to help improve and make more visible changes that must occur in a system for particular actors in a given period of time for a specific condition by focusing on cultural process that lead to the proposed change actions.

The Yourdon Systems Method (Yourdon Inc., 1993) points out three main orthogonal and independent viewpoints of a system: Function (what the system does), Time (what happens and when), and Information (what information is used by the system). The method designs a system by constructing models with the purpose of capturing all the relevant information about the enterprise where the system will be implemented and about the system itself. The three viewpoints are used to identify that information. The requirements for the system are represented through a System Essential Model. This model disregards any technological issue and only concerns on the real-world subjects of the system under construction.

In relation to the domain engineering, and in particular to domain analysis, we can briefly reference the following:

DRACO, presented by Neighbors in his thesis (Neighbors, 1980), promotes transformation between domain descriptions and an executable program. The method encompass' three main phases: determine domains of interest, research the domain and construct a software system. The first phase, determine domains of interest, takes as input the organizational goals, and an analysis of demand for similar systems. The output is a problem domain where the organization is interested in producing software. This output and information about different problem domains are inputs for the second phase, research the domain. This phase outputs, at least, a domain analysis reports that can be inputted to second phase through successive refinements until there is enough detail to construct the domain. The second phase encompasses the construction and test of a domain and the further inclusion of that

domain to a library of domains. The third and last phase takes as input the analysis report and produces an executable language.

The DRAMA framework (Kim, Park, & Sugumaran, 2008) allows for establishing traceability between domain requirements and domain architecture, based on a quantitative analysis. The traceability is supported by a semi-automated tool. DRAMA supports domain requirements analysis and domain architecture modeling. Our interest in the framework resides in the techniques used to elicit the domain requirements through goal and scenario based analysis. Scenarios describe real situations and as such, they capture real requirements.

The Feature-Oriented Domain Analysis (FODA) method (Kang, Cohen, Hess, Novak, & Peterson, 1990) promotes better understanding of the domain under analysis and presents some guidelines for the desired domain architecture. The focus of the method, as its name implies, is on domain analysis, that is, the analysis of the problem space. FODA essentially encompasses three phase, as depicted in Figure 5, Context Analysis, Domain Modeling and Architecture Modeling.

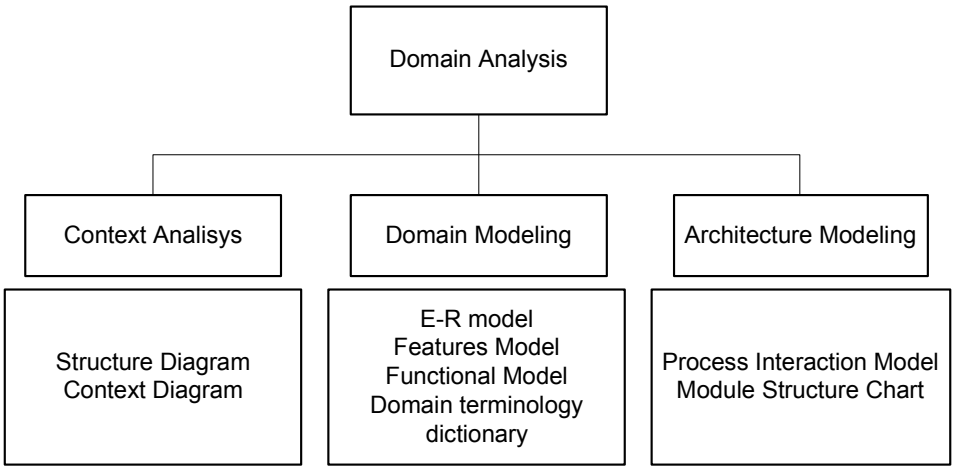

Figure 5: Phases and Products of Domain Analysis (based on (Kang, et al., 1990))

The Context Analysis phase contextualize the domain, the Domain Modeling phase express the software problems to tackle and the Architecture Modeling phase defines the structure for the software implementation, that is, the software architecture. Each phase encompasses several activities and each produce documents that

describe the domain. As such, we can say that this method contributes for a better understanding of the domain under study. Another point of interest concerning this method is the fact that it establishes a relation between itself the possible roles participating in the domain analysis process. That is to say that, for example, domain user and domain experts are sources for the method and the domain user is also a consumer. Later on, Kang et al. presented an evolved version of FODA called Feature-Oriented Reuse Method (FORM) (Kang et al., 1998), as an extension to support software design and implementation. This extension gives more attention to the implementation details that were not addressed in the previous version.

The primary objective of the Domain Analysis and Reuse Environment (DARE) CASE tool and method (Frakes, Prieto-Diaz, & Fox, 1998) is to create a generic architecture that describes architectural elements and their relationships for a family of systems.

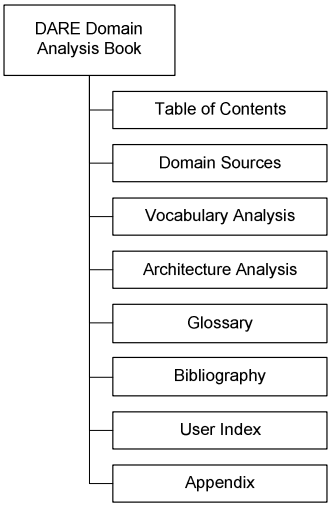

Figure 6: Domain Analysis Book Composition (based on (Frakes, et al., 1998))

The DARE method focus is on the extraction of high-level domain information from experts. To create a domain model, in DARE called Domain Book, it is necessary to extract information from the domain experts and documents and from existing code present in the various systems under analysis. The DARE Domain Book contains the sections described in Figure 6. Each section has several chapters (book metaphor), while each chapter has entries and the information of how it was created. The method has two main steps: bottom-up analysis, with the validation of the generic

architecture and features through the analysis of text and code of the domain (documents); top-down analysis with the postulation of generic architecture and features based on domain expert knowledge and experience (people).

Organization Domain Modeling (ODM) (M. A. Simos, 1995) is a multi-organization and multi-domain domain analysis systematic approach able to produce domain analysis process models able to be implemented in various technologies. The ODM method focus, as its name implies, in the organizational aspects of an organization rather than on technological issues. It defines a domain modeling lifecycle, distinct and orthogonal to the system development lifecycle. The method is separated into three main phases (M. Simos, Creps, Klinger, Levine, & Allemang, 14 June 1996): Plan Domain, Model Domain, and Engineer Asset Base. The Plan Domain phase mainly consists in setting the overall project objectives and in the specification of a domain for the project, aligned with organizational needs. This phase also encompasses the designation of the stakeholders relevant for the domain. The Model Domain phase produces a domain model for the selected domain based on three fundamental constructors: lexicon, concepts and features. The domain model describes commonality and variability within the selected domain and is constructed by acquiring domain information, describing the domain and by refining the domain model. The Engineer Asset phase purpose is to implement the variability defined in the domain model. This is done by defining a scope for the asset base, architect the asset base and then implement it.

Other domain analysis methods, like Stability-Oriented Domain Analysis (SODA) (Haitham, 2011) or Family-oriented Abstraction, Specification, and Translation (FAST) (D. Weiss, 1998) focus on the product perspective and thus are excluded from this process-level perspective.

2.3 Process-level Activities Elicitation

Here we introduce the PL.AC.E lightweight method. PL.AC.E was defined after we had experienced, in the scope of the ISOFIN project (related activities) a need to define a set of processes that are based on canonic activities in the Information and Communication Technologies (ICT) domain related to engineering projects (engineering projects are projects where professional engineering skills are required. See (R. J. Machado & Amaral, Fevereiro, 2011)). These processes later act as input for the use cases descriptions required for describing the project result, the proposed system.

The method has six main phases, as shown in Figure 7:

(i) Definition of the project scope;

(ii) Creation of the cross product between canonic regulated and non-regulated ICT activity kinds;

(iii) Definition of the relevant constraints in the Mission's scope and concerning the overall project;

(iv) As a result of the instantiation of the canonic ICT activities in (ii) and applying the constraints defined in (iii), define specific activities for the domain under analysis;

(v) Specific activities resulting from (iv) are validated against a set of questions and the domain stakeholders are discovered.

(vi) The approved activities from (v) can be classified by its nature;

In the remainder of this section, we will detail each phase of the PL.AC.E method, depicting the rationale for each and relating them with the previously exposed methods.

The definition of a project scope includes the definition of the project purpose, objectives, and operational activities of the enterprise, expressed thru its vision, mission, goals or strategies (OMG, 2010) depending on the granularity of the project under analysis.

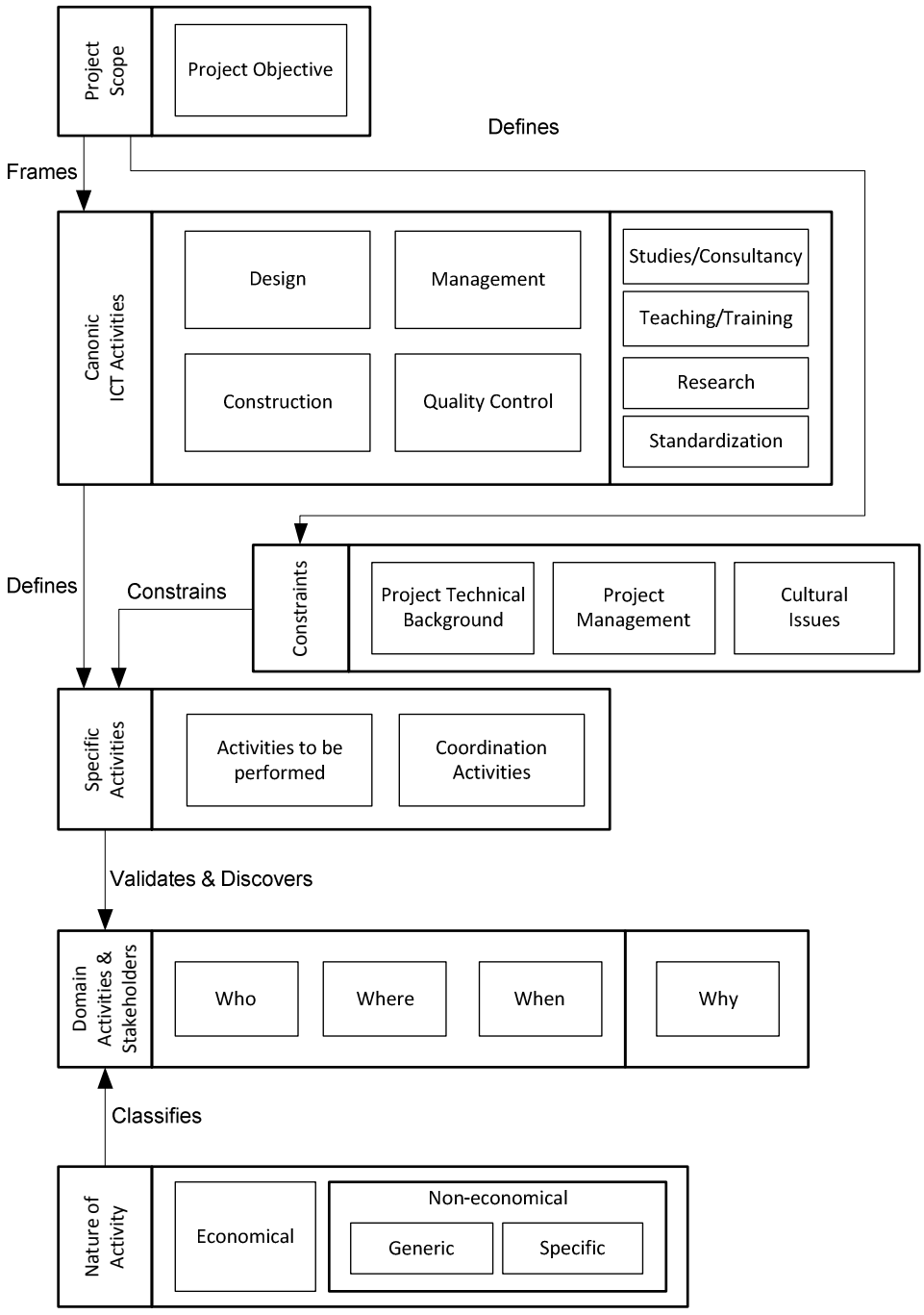

Figure 7: Phases of PL.AC.E

We analyze activities in the domain of ICT engineering, and since an Engineer is considered a *"professional that deals with the application of sciences and techniques relating to various branches of Engineering in research, design, study, design, manufacture, construction, production, inspection and quality control, including coordination and management of these and other activities with them related"* (R. J. Machado & Amaral, Fevereiro, 2011), such activity kinds should also be considered.

We have chosen to divide the activities in two main groups: regulated activities, encompassing Design, Construction, Quality Control and Management; and an un-regulated group, encompassing the remaining. By regulated, we mean that the activity kinds are subject to formal, legal or defined regulations in some sort of meaning due to the consequences of the execution of such activities carries to the entities related to them, in the context of an ICT project. For each of these activity kinds it is necessary to elicit matching activities.

In "Design", we encompass all business and organizational analysis and also requirements engineering related activities. These activities include (BABOK): planning and monitoring of business analysis activities throughout the requirements process, managing requirements (when requirements change, that change must be managed, including communication with the interested stakeholders), business situation/enterprise analysis, requirements elicitation and analysis, solution assessment and validation and also individual competencies identification.

In "Construction", we are grouping all the activities related to building descriptions of the software internal structure and related support infra-structure. This category of activities takes as input all the requirements elicited in the previous set of activities and guarantee, at least, information type organization, description of the relationships between business process, data, IT mission systems, among others (EABOK). Activities performed in this set (EABOK; G2SEBoK) can be included, for example, the ones present in the Architecture Development Method of TOGAF (The Open Group, 2009) and in the Zachman Framework (Zachman, 1987).

The "Quality Control" group refers to activities concerning verification and validation of software solutions, as referred by the Capability Maturity Model Integration

(CMMI) (CMMI Product Team, 2006). These activities encompass testing and regard it as one of the most important in the software development lifecycle, with average costs and duration above any other activity (Monteiro, Machado, & Kazman, 2009). Common activities in this group relate to system evaluation for determining if it satisfies given requirements, during and after the development process and evaluation, in a given development moment, if the imposed starting conditions were met.

In “Management”, all project management activities are performed from the ones depicted in PMBOK (Project Management Institute, 2008) to the Agile methods, like SCRUM (Schwaber & Beedle, 2001). This group relates heavily to the previous since its activities focus are on the previous groups’ activities. Here, we include planning, coordination, measuring, monitoring, controlling and documenting all the intangible artifacts developed in the context of software.

In relation to the non-regulated activities, Studies/Consultancy refers to the studies and consultancy efforts that can be made to formalize new Design, Management, Construction or Quality Control methodologies and provide support to establish new criteria on the regulated activity kinds.

Teaching/Training refers to the activities that occur with the objective of disseminating information or develop new skills concerning the regulated activities in audiences that did not have those skills at a desired level.

In this context, Research refers to the activities that engineers execute to construct new methodologies and theories enacting future usages and exploration of the regulated activity kinds.

Standardization refers to the set of regulations that are imposed on the regulated activity kinds. Even these standards evolve in conjunction with research or experience. Standardization applied to the regulated activity kinds originates a validated and coherent set of activities that are accepted by a given group or within a given context. When applying research and standards, there are some regulations that can be used within specific contexts, like for instance, The Guide for Software

Design ("IEEE Guide to Software Design Descriptions," 1993) when dealing with design-related processes or Business Analysis (BABOK). Other standards can be generically applied, like the Portuguese standards for Research, Development and Innovation (Instituto Português da Qualidade (IPQ), 2007) that defines requirements and processes enabling an organization to coherently promote innovation. Other general-purpose standard that can be referred in this section is the ISO 15504 (International Organization for Standardization, 2008-11-25), that provides a framework for the assessment of processes and influence organizations to make decisions. These kinds of standards help framing the processes that can be executed within the context of the defined project.

Constraints related to engineering projects must also be considered. Engineering projects environments are defined by (R. J. Machado & Fernandes, 2002) and project management are defined in (Project Management Institute, 2008):

(i) the project technical background, consisting of a scientific framework that supports a methodology and a technological framework that defines the techniques and tools used by engineers in the context of project activities;

(ii) taking into consideration the scope of the project and its deliverables in the project management section and;

(iii) Every cultural issue of interest in the scope of project must be accounted, like in SSM (Checkland, 1985) or in DARE (Frakes, et al., 1998), cultural aspects that surround the project (by itself) or the product resulting from the project (Hanisch & Corbitt, 2007). As an example, the case where usability requirements based on cultural aspects of a given region constrains the activities that must be executed in that region.

A baseline for the definition of activities in the domain of a given ICT project is based on Mintzberg's theories. He states that *"Every organized human activity – from the making of pottery to the placing of a man on the moon – gives rise to two fundamental and opposing activities: the division of labor into various tasks to be performed and the coordination of those tasks to accomplish the activity."* (Mintzberg,

1989). These translate into activities to be performed and coordination activities. Specific activities are only used for tagging purposes.

Having in mind the previously exposed, we reach a phase in the method where canonic activities for the domain under analysis can be derived. A list with the activities must be produced and the next step allows the validation of such list, by imposing a set of questions. These questions, encompass the discovery of who the actors (domain stakeholders) are that interact with each activity and where and when the activity is executed and when. To each of these questions another validation question must be posed: why. This allows for the validation of the results with the project objective, defined in the project scope and thus producing only contextualized information.

Finally, the nature of the economic activities is also a factor that must be accounted for. There is the need to classify the elicited activities in Economical (R. J. Machado & Fernandes, 2002) or Non-Economical (Bensaou & Venkatraman, 1993). The Nature of the Activity is a set of tags that are added, allowing the classification of the elicited activities (Bensaou & Venkatraman, 1993).

The PL.AC.E method reuses some of the previously exposed methods features. Like in DRACO, we need to first analyze the domain, gather information about it and catalog it, in order to obtain iteratively detailed domain information. As in DRAMA, a similar situation to scenarios is used, where we intent to capture real present situations or desirable future actions. Like in FODA, PL.AC.E allows for establishing relations between the participating roles in the process (domain user, domain expert and all the canonic ICT activities). At the same time, both methods share the contextualization concept, both wanting to obtain information about the domain of study. Like in FORM, PL.AC.E also regards implementation and execution details that must be accounted in the regulated activity kinds or in the cultural issues (this is also common to SSM and DARE). From ODM, PL.AC.E also benefits from the domain analysis and scope perspective impact on the project. This domain must be aligned

with the organization objectives, depicted in the project scope, defined in its early phase.

2.4 Context for the Activities on the Cloud

One of the inputs for PL.AC.E regards knowing the intended target for deployment. The chosen scenario regards a cloud-related target. It is of the utmost relevance the definition of what the cloud-related subject is in order to define precisely the actuation context of the activities to be defined.

According to NIST's cloud definition (NIST, 2009), the cloud computing model has five essential characteristics: On-demand Self-service; Broad Network access; Resource Pooling; Rapid Elasticity; and Measured Service. For the purpose of understanding the next chapters design decisions, each is these characteristics is detailed in the next paragraphs.

On-demand Self-service

A consumer can unilaterally have access to computational resources like processing time, storage and network servers, automatically and according to their specific needs and without human interaction (Krutz & Vines, 2010; Mell & Grance, 2009).

Graphical interfaces to be effective and acceptable for an end user must be easy to use and make available features to manage the service catalogue (Krutz & Vines, 2010). By making the interfaces easy to use and eliminating user interaction, there are productivity gains and cost reduction, for the end-users and for the service suppliers.

Broad Network access

Resources are available through the network and are accessible by means of standard interfaces that promote the usage by heterogeneous platforms: mobile phones,

laptops, and tablets, among others. To make the cloud paradigm a real alternative to in-house data centers there must exist broadband connections that connect end-users to cloud services (Krutz & Vines, 2010; Mell & Grance, 2009).

Resource Pooling

The resources are organized in a way to support multiple end-users by means of multi-tenant models. There are different physical and virtual resources, managed according to the customer demand (Mell & Grance, 2009). The end-user does not have control or knowledge about the server's exact location. Nevertheless, it is possible to define, at a very high level of configuration, their location (due to legal constraints that affect where databases should be, for example).

This computational model has a broad range of resources in order to answer customer needs, reach economy of scale and respect the agreed quality of service. Applications need resources for executing and those should be efficiently assigned to achieve an optimal performance, even if they are geographically dispersed (Krutz & Vines, 2010).

Rapid Elasticity

The resources should be supplied in a quick and elastic way, in some cases, automatically to assure the system scalability. For the end-used, the resources should appear like if they are unlimited, bought at any time and number and the cost calculated accordingly to the usage time and amount of resources spent. This characteristic regards the ability of increasing or diminishing allocated resources to comply with self-service requirements (Krutz & Vines, 2010; Mell & Grance, 2009).

Measured Service

These computational systems control and optimize resource usage, automatically and accordingly with the appropriate service level and type (example: storage, processing, bandwidth, among others). Resource usage can be monitored, controlled and

communicated making the service transparent for the end-user as for the supplier. The number of used resources can be monitored and defined dynamically. This way, only the resources used for a given session will be charged (Krutz & Vines, 2010; Mell & Grance, 2009).

Regarding the distribution model for cloud, there are three typical cloud classifications in the perspective of who owns and manages the cloud infrastructure.

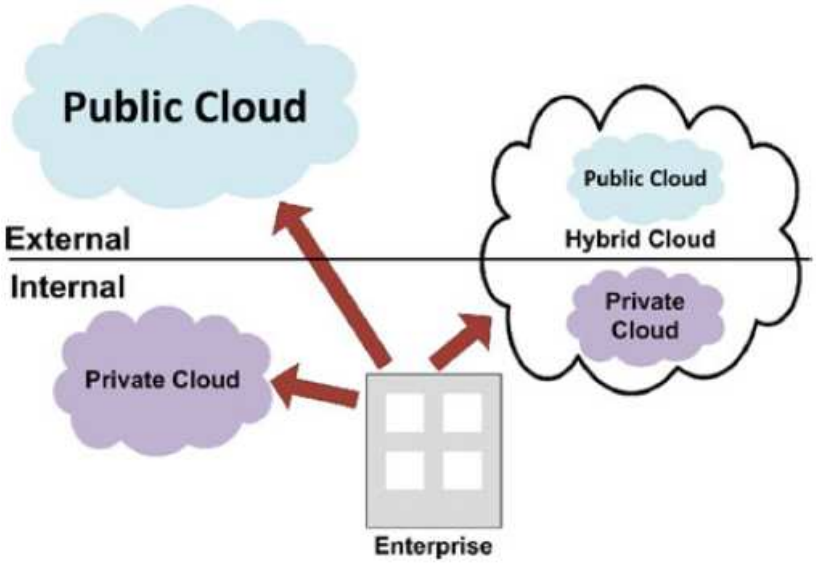

Figure 8: Distribution Models

In Figure 8, typical distribution models are depicted: public cloud, private cloud and hybrid cloud. For the sake of understandability the next paragraphs will make a brief overview of each.

Public Cloud

A cloud infrastructure is publicly available, or to a large industrial group. The infrastructure is proprietary of an organization that sells cloud services. This is the most common type of cloud and its services are made available through a service supplier and the resources are shared among other customers. Security and data governance are the greatest concerns of this approach (Furht & Escalante, 2010).

Private Cloud

The cloud infrastructure is owned by a single organization. It may be deployed inside or outside the organization's facilities, being managed by the organization itself or by another one. Many of the infrastructures are managed by big organizations or governmental groups that prefer to maintain their data in a controlled and more secure environment (Furht & Escalante, 2010).

Table 1 summarizes a comparison between the public and private cloud models.

Table 1: Public vs Private Cloud Models (adapted from(Furht & Escalante, 2010))

	Public Cloud	**Private Cloud**
Infrastructure owner	Third party (cloud provider)	Company
Scalability	Unlimited and On-Demand	Limited to the installed infrastructure
Control and Management	Only manipulate the virtual machines, resulting in less management costs	High level of control over the resources, and need more expertise to manage them.
Cost	Lower cost	High cost (space, cooling, energy consumption, hardware, etc.)
Performance	Unpredicted multi-tenant environment, making performance goals hard to achieve	Guaranteed performance
Security	Concerns regarding data privacy	Highly secure

Community Cloud

A cloud infrastructure is shared by several organizations, supporting a community with identical concerns (e.g., mission, security requirements, and policies), able to co-exist inside or outside the organization facilities and being managed either by the organizations that own it or by an external entity (Mell & Grance, 2009).

Hybrid Cloud

A cloud infrastructure is a composition of two or more cloud (private, public, or community) that remain single entities and, being connected through standard or proprietary technology, allow data and application portability (Mell & Grance, 2009). This way, an organization can sustain its data and critical applications, inside its facilities, deploying the less critical in a public cloud (Furht & Escalante, 2010).

Regarding the service models, the NIST definition regards the following three: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In this definitions are left outside Business Process Management as a Service (and related) for being considered a specializations of the SaaS model.

Infrastructure as a Service

In this model, the user has available processing time, storage, networks and other computational resources. The user can deploy and execute any software, including operative systems and applications. The basic cloud infrastructure is not controlled by the end-user, but they have control over the operative system, storage, and application deployment, having the possibility to control a limited range of network components, like firewalls. In this category are included Computation as a Service (CaaS) and Data as a Service (DaaS).

Platform as a Service

In this model, the end-user has access to deploy in the cloud applications bought or created by themselves, but only if specific programming languages/tools are supported by the model infrastructure. The end-user does not control or manage the infrastructure of the cloud (network, servers, operative system, or storage), but controls the deployed applications and their configurations.

Software as a Service

This model allows to the end-user to use applications that are available by the cloud supplier in the cloud system. Those applications can be accessed through a graphical interface, like a web browser. The end-user does not have control on anything on the cloud, with the exception of the application user-specific configurations.

2.5 The ISOFIN Project in the Cloud

Every time a new technology emerges, its adoption always faces risks and drawbacks from the adopters. In the insurance industry, whose focus is on the calculation and mitigation of risks, this situation is more noticeable. Deploying a new technology or process in a business with an existing infrastructure presents some risk. If we also add some uncertainty in the offer-side features, it can lead to loss of innovation and business opportunities.

The combination of SaaS and cloud is of great value to the insurance industry but its adoption is slow, namely in Policy Administration Systems and Claims Management. This is mainly due to insurance companies not having a well-defined cloud computing strategy and a visible benefit that counteracts the potential security problems (J. Weiss, 2010). By having a combination of private and public cloud in a SaaS model, it is possible to coordinate different channels, like distribution, claims and marketing. The cloud assures to end-users continuous availability of most data, files and relevant information.

The ISOFIN project (ISOFIN Project Consortium, 2010) aims to deliver a set of coordinating services in a centralized infrastructure (public cloud), enacting the coordination of independent services relying on separate infrastructures (private clouds). The resulting ISOFIN platform, allows for the semantic and application interoperability between enrolled financial institutions (Banks, Insurance Companies and others), as depicted in Figure 9.

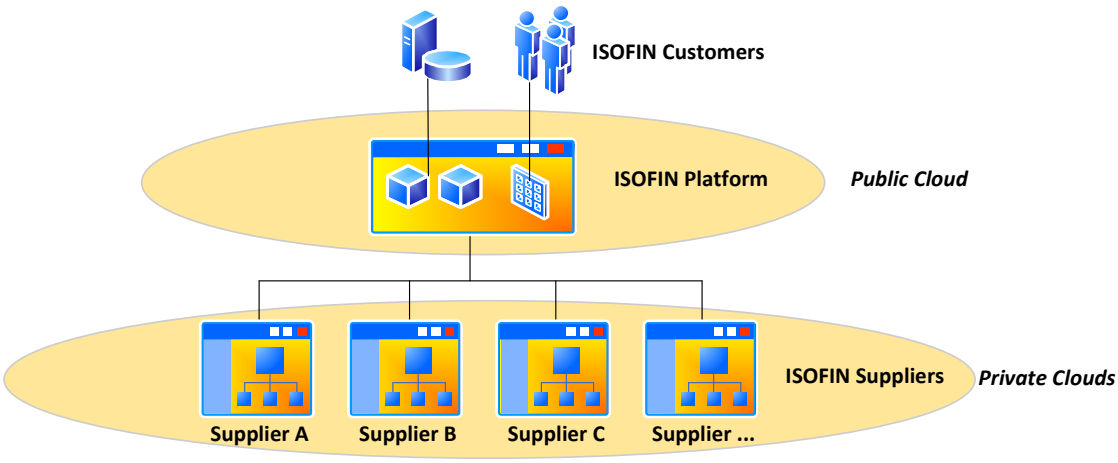

Figure 9: Placing the ISOFIN on the Cloud

This project is used as the main case study that supported this thesis work. The project started officially in 2010 but the preparatory works began in 2009 with the application submission for funding from the Portuguese Government (project funded with reference QREN/2010/013837) (ISOFIN Project Consortium, 2010).

The ISOFIN project encompasses eight institutions, ranging from universities, research centers and private software development companies for the bank and insurance domains. The stakeholders of this group had different backgrounds and expectations regarding the project outcome. These differences resulted in the lack of definitions for the requirements that the project's applications would support and even to a proper definition of a business model that the organizations that participate in the project would pursue.

The expected benefits of the ISOFIN project are to deliver a platform able to guarantee the coordination of different application residing in the supplier's private clouds in a unified view, made available in the ISOFIN Platform public cloud.

The ISOFIN project main constructors and terminology are described in Section 4.2 of this thesis.

2.6 Conclusions

The PL.AC.E. method is used to create the activities that will make up the initial context for the project process definitions, based on the cloud premises detailed in the previous sections. This detail is essential to contextualize the problem, possible solutions and design decisions that must be taken during the analysis and design phases. The deployment configurations, the various usages of the cloud infrastructure will frame and at the same time provide a hints and guidelines for the intended business model.

A possible problem that can be addressed to the method concern the fact that its execution can sometimes be constrained by organizational or project structures, leading to activities that represent the actual situation and not the expected one. The method is a green-field approach able to deliver a set of canonic ICT activities suitable for a scoped project.

Having in mind that the purpose of domain analysis is the reuse, and since the method is commonly used in a process-level perspective, eliciting requirements in this perspective is a task that needs to be carefully executed in order not to exclude key requirements from the analysis that should be arisen when transiting to the product-level perspective. Hence, the domain analysis perspective that we use regards the domain of analysis perception with the purpose of creating a requirement model (activities and processes) that will support traditional (product-level) engineering processes with reuse purposes.

The ISOFIN project previously presented emerges in two distinct edges: the first one, with the need of the financial domain to have an infrastructure able to enact the coordination of distinct services and, the second one, if there is no knowledge on the real business model of the project, there is no guarantee on the development roadmap for the development of the final solution. These edges, connected, are combined into schema that is the roadmap of this thesis: Starting from (1) a reality where there is no proper definition of the required inputs for a business model, (2) learning about the target environment and the scenarios that must be supported and then, moving on to the (3) high-level definitions of the activities that must be supported. Step (1) relates to PL.AC.E.; Step (2) relates to the cloud environment; and Step (3) relates to the organization configurations that will be detailed in the next chapter.

Chapter 3

Modeling Information and Software Systems

Chapter Contents

3

Modeling Information and Software Systems

"Depending on where you're looking, one person's system is another's subsystem."

Grady Booch

This chapter introduces the basic structure that makes our approach: the V-Model process representation. This representation was built upon a set of models that are intertwined together to derive a logical architecture representation of an information system. The input for the V-Model process consists of a set of requirements that are not agreed upon, and are intended to be the business model of the desired information system.

3.1 Introduction

One of the top concerns of information technologies (IT) managers for almost thirty years relates to software and the business domain alignment (Luftman & Ben-Zvi, 2010). The importance of aligning software with domain specific needs for the purpose of attaining synergies and visible success is a long-running problem with no visible or deterministic solution. There are many questions concerning this subject,

going from how to align several strategic components of an organization with the necessary maturity or how specific domain needs and software that supports the domain are aligned with each other.

The perspective on domain specific needs with software alignment has changed along the years. Initially, alignment meant relating specific domain needs with supporting software plans. Later, the concept evolved to include business and software strategies, business needs and information system priorities. This created the need for aligning business models (as a rationale for how the organizations create, deliver and capture value for a given business) with the underlying information system (people and software solutions) that is designed to support part or whole of the business model.

One of the possible representations of a software solution is its logical architecture, resulting from a process of transforming business-level and technological-level (of any given domain) decisions and requirements into a representation (model). A model can be seen as a simplified view of reality, and possesses five key characteristics: abstraction, understandability, accuracy, predictiveness, and inexpensiveness (Selic, 2003). This representation is fundamental and mandatory to analyze and validate a system but is not enough for achieving a full transformation of the requirements into a model able to implement stakeholders' decisions. It is necessary to promote an alignment between the logical architecture and other supporting models, like organizational configurations, products, processes, or behaviors.

An organization is about people. Stakeholders are responsible for the decision-making processes that influence the organization's strategy at any given level under analysis (Campbell, Kay, & Avison, 2005). At the same time, the stakeholders also influence the organization's software architecture and systems. Aligning domain specific needs with the way that software solutions are organized is a task that must be accounted for and whose results are not easily, or at all, measurable.

Our approach is based on the premise that there is no clearly defined context for eliciting product requirements within a given specific domain. As an example for a situation where there is no clearly defined context, we present the ISOFIN project (ISOFIN Consortium, 2010). This project is used along the present work as a real industrial case study of the applicability of the presented approach.

The ISOFIN project is executed in a consortium comprising eight entities (private companies, public research centers and universities), making the requirements elicitation and the definition of a development roadmap difficult to agree. The initial request for the project requirements resulted in mixed and confusing sets of misaligned information. Even when a requirement found a consensus in the consortium, all the stakeholders did not easily understand the intended behavior or its definition.

Our proposal of adopting a process-level perspective was agreed on and, based on the knowledge that each consortium member had of the intended project results, the major processes were elicited and a first approach to a logical (process-level) architecture was made. After execution of the process-level perspective, it was possible to gather a set of information that the consortium is sustainably used to evolve to the traditional (product-level) development scenario.

Elicited requirements in a process-level perspective describe the processes in a higher level of abstraction, making them understandable by the consortium key decision-taking members (business stakeholders). At the same time, by defining the major activities, their relations and flows, the definitions and intended behavior of the system, expressed in the architecture that results from the process-level 4SRS method, describe the system to the consortium key technological developers (technological stakeholders).

Our approach results in a “Vee” Model-based adaptation (V Model) (Haskins & Forsberg, 2011), which suggests a roadmap for product design based on domain specific needs. The model requires the identification of those domain specific needs and then, by successive models derivation, it is possible to transit from a domain level

perspective to a software (IT) level perspective and at the same time, aligns the requirements with the derived models, reducing the gap between business and technological stakeholders.

This chapter starts by framing our approach with others work. Then we detail our V-Model representation to promote the creation of an information system logical architecture. The ISOFIN project, as a case study is presented, giving details about the pertinence of using the chosen presented models for creating context to product design based on the information system logical architecture. We also explain how to proceed from one model to another and includes discussions, comparison with the related work and an assessment overview of the presented approach and its validation through ARID.

3.2 Related Work

A typical software development project is coordinated so that the resulting product properly aligns with the domain-specific (business) model intended by the leading stakeholders. As an economical plan for the organization or for a given project, the business model contributes for eliciting the requirements by providing the product's required needs in terms of definitions and objectives.

By "product", we mean applications that must be computationally supported. They may be in the form of independent application modules or interconnected business services.

In situations where organizations focused on software development are not capable of properly eliciting requirements for the software product, due to insufficient stakeholder inputs or some uncertainty in defining a proper business model, a process-level requirements elicitation is an alternative approach.

The process-level requirements assure that organization's business needs are fulfilled. However, it is absolutely necessary to assure that product-level (software-

related) requirements are perfectly aligned with process-level requirements (derived from the business requirements), and hence, are aligned with the organization's domain-specific requirements. In this section, we chose to refer to other author's work related to ours in the diverse topics that integrate our approach: business and IT alignment, governance, alignment of requirements with system specifications, the process-level perspective, process architectures and the models that can be used to describe requirements and help build the context for product elicitation.

An approach that enacts the alignment between domain-specific needs and software solutions, is the goal oriented approach GQM+Strategies (Goal/Question/Metric + Strategies) (Basili et al., 2010). The GQM+Strategies approach uses measurement to link goals and strategies on all organizational levels. This approach explicitly links goals at different levels, from business objectives to project operations, which is critical to strategic measurement. Applying GQM+Strategies makes easier to identify goal relationships and conflicts and facilitates communication for organizational segments. Another goal-oriented approach is the Balanced Scorecard (BSC) (Kaplan & Norton, 1992). BSC links strategic objectives and measures through a scorecard in four perspectives: financial, customer, internal business processes, and learning and growth. It is a tool for defining strategic goals from multiple perspectives beyond a purely financial focus.

Another approach, COBIT (Information Technology Governance Institute (ITGI), 2012), is a framework for governing and managing enterprise IT. It provides a comprehensive framework that assists enterprises in achieving their objectives for the governance and management of enterprise IT. It is based on five key principles: (1) meeting stakeholder needs; (2) covering the enterprise end-to-end; (3) applying a single, integrated framework; (4) enabling a holistic approach; and (5) separating governance from management. These five principles enable the enterprise to build an effective governance and management framework that optimizes information and technology investment and use for the benefit of stakeholders.

In order to represent the intended aligned system specification we use models. It is recognized in software engineering that a complete system architecture cannot be represented using a single perspective or model (Kruchten, 1995; Sungwon & Yoonseok, 2005). Using multiple viewpoints, like logical diagrams, sequence diagrams or other artifacts, contributes to a better representation of the system and, as a consequence, to a better understanding of the system. Some architecture views can be seen in the works of Clements *et al.* (P. Clements, Garlan, Little, Nord, & Stafford, 2003), Hofmeister *et al.* (Hofmeister, Nord, & Soni, 2000) and Krutchen (Kruchten, 1995). Krutchen's work refers that the description of the architecture can be represented into four views: logical, development, process and physical. The fifth view is represented by selected use cases or scenarios. Zou and Pavlovski (Zou & Pavlovski, 2006) add another extra view, the control case view, that complements the use case view to complete requirements across the collective system lifecycle views.

Since the term *process* has different meanings depending on the context, in our process-level approach we acknowledge that (1) real-world activities of a software production process are the context for the problem under analysis and, (2) in relation to a software model context (Conradi & Jaccheri, 1999), a software process is composed of a set of activities related to software development, maintenance, project management and quality assurance.

For scope definition of our work, and according to the previously exposed acknowledgments, we characterize the process-level perspective by (1) being related to real world activities, including business, and when related to software (2) those activities encompass the typical software development lifecycle. Typically, product-level approaches promote the functional decomposition of systems models. Our approach is characterized by using refinement (as one kind of functional decomposition) and integration of system models. Activities and their interface in a process can be structured or arranged in a process architecture (Browning & Eppinger, 2002).

The process architecture represents a fundamental organization of service development, service creation, and service distribution in the relevant enterprise context (Winter & Fischer, 2006b). Designing a software architecture provides a more accurate definition of the requirements. There are several approaches to supporting the proper design of software architectures, like FAST (D. M. Weiss & Lai, 1999), FORM (Kang, et al., 1998) or KobrA (Bayer, Muthig, & Göpfert, 2001). These all relate to the product-level perspective. In a process-level perspective, Tropos (Castro, Kolp, & Mylopoulos, 2002) is a methodology that uses notions of actor, goal and (actor) dependency as a foundation to model early and late requirements, architectural and detailed design. Machado *et al.* present the 4SRS (Four-Step-Rule-Set) method for architecture design based on requirements. 4SRS is usually used in a product-level perspective (R. J. Machado, Fernandes, Monteiro, & Rodrigues, 2006b), but it also supports a process-level perspective (N. Ferreira, et al., 2012; R. J. Machado & Fernandes, 2002). The result of the application of the 4SRS method is a logical architecture. Logical architectures can be faced as a view of a system composed by a set of problem-specific abstractions supporting functional requirements (Kruchten, 1995).

The defined and derived models suggested by our approach, used alone and unaligned with each other, are of a lesser use to organizations and stakeholders. Our approach begins in a domain-specific perspective, by defining the organizational configurations that represent major interactions, at a very high-level, in the chosen domain, and ends with a technological view of the system. From one perspective to the other, alignment must be assured. The alignment we refer to relates to domain-specific and software alignment (Campbell, 2005), and in our case, where the domain-specific needs must be instantiated into the creation of context for proper product design.

A possible point of failure in achieving the intended alignment relates to the lack of representativeness of the necessary requirements for expressing domain-specific needs. According to Campbell *et al.* (Campbell, et al., 2005), the activities that support the necessary information for creating context for requirements elicitation

are not explicitly defined or even promoted. Also, existing approaches to designing software architecture do not support any specific technique for requirements elicitation in a process-level perspective; rather, they use the information delivered by an adopted elicitation technique.

Typical (product-oriented) elicitation techniques may not be able to properly identify the necessary requirements within a given context creating an opportunity for our approach to define the process that support the derivation of models with the purpose of creating context for product design. With the case study described in this work we demonstrate that firstly adopting a process-level perspective allows for better understanding of the project scope and then support the creation of context for the elicitation of requirements of the product to be developed.

3.3 An Approach to Domain and Software Models Alignment

In this section, we present our approach, based on successive and specific models generation. As models, we use Organizational Configurations (OC) (Evan, 1965), *A-Type* and *B-Type* Sequence Diagrams (R. Machado, Lassen, Oliveira, Couto, & Pinto, 2007), use cases and process-level logical architecture diagrams. All these models are briefly described in this section and properly exemplified in the case study that follows, where more detail is given on how to derive a model from the previous models.

Traditional development processes can be referenced using the Royce's waterfall model (Ruparelia, 2010) that includes five typical phases in its lifetime:

(i) Analysis;

(ii) Design;

(iii) Implementation;

(iv) Test;

(v) Deployment.

Defining a simplified macro-process for supporting the requirement elicitation in a process-level approach must take into account the waterfall model lifecycle for a project. An adaption of the macro-process for the ISOFIN project will be later detailed in Figure 31 in Section 4.2 - The ISOFIN Project.

We frame our proposed V-Model approach in the Analysis phase of the lifecycle model, as depicted in Figure 10. This simplified development macro-process based on the waterfall model uses the V-Model generated artifacts for eliciting requirements that, in a process-level approach, are used as input for the traditional 4SRS usage (product level) (R. J. Machado, et al., 2006b). The product-level 4SRS promotes the transition from the Analysis to the Design phase.

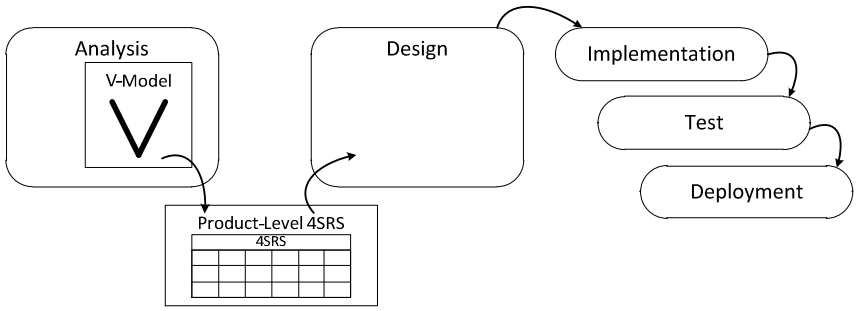

Figure 10: Framing the V-Model representation in the development macro-process

The OC model is a high-level representation of the activities (interactions) that exist between the business-level entities of a given domain. Figure 11 shows an example of the aspect of an OC, with two activity types, each with a role and two interactions.

The set of interactions are based on domain-specific requirements (such as business) and, in conjunction with the entities and the stakeholders, are represented with the intention of describing a feasible scenario that fulfills a domain-specific business vision. In what concerns OCs characterization for the purpose of our work, each configuration must contain information on the performed activities (economical (R. J. Machado & Fernandes, 2002) or non-economical (Bensaou & Venkatraman, 1993)),

the several professional profiles (actors and skills) that participate in the activity execution and also the exchange of information or artifacts. There must be defined as many OCs as the ones required to express all the major interactions defined by the business stakeholders that relate to the intended system.

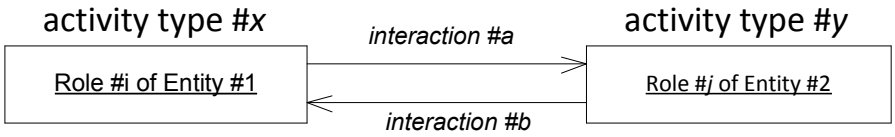

Figure 11: Organizational Configuration

Our approach uses a UML stereotyped sequence diagram representation to describe interactions in early analysis phase of system development. These diagrams are presented in this work as *A-Type* Sequence Diagrams. Another stereotyped sequence diagram, called *B-Type* Sequence Diagrams, allows for deriving process sequences represented by the sequence flows between the logical parts depicted in the logical architecture.

One must assure that a process' sequences modeled in *B-Type* Sequence Diagrams depict the same flows as the ones modeled in *A-Type* Sequence Diagrams, as well as being in conformity with the interactions between architectural elements (AEs) depicted in the logical architecture associations. An AE is a representation of the pieces from which the final logical architecture can be built. This term is used to distinguish those artifacts from the components, objects or modules used in other contexts, like in the UML structure diagrams. An example of *A-Type* and *B-Type* Sequence Diagrams can be found in Figure 12.

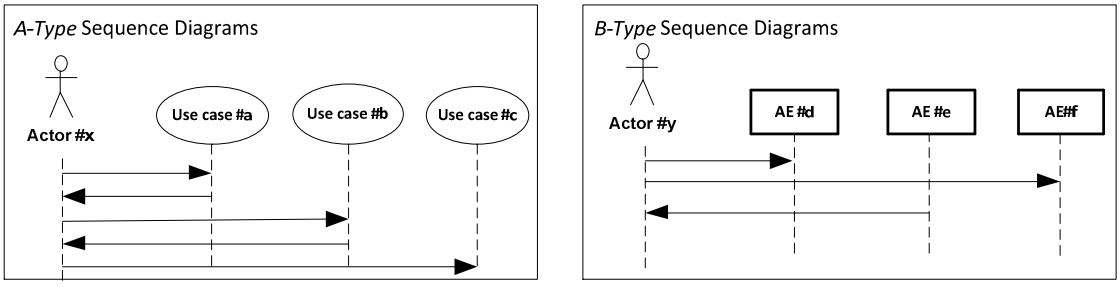

Figure 12: A- and B-Type Sequence Diagrams

The generated models and the alignment between the domain specific needs and the context for product design can be represented by a V-Model as seen in Figure 14. The V-Model representation (Haskins & Forsberg, 2011) provides a balanced process representation and, simultaneously, ensures that each step is verified before moving to the next. In this V-Model, the models that assemble it are generated based on the rationale and in the information existing in previously defined models, i.e., *A-Type* diagrams are based on OCs, use cases are based on *A-Type* diagrams, the logical architecture is based on the use case model, and *B-Type* diagrams comply with the logical architecture.

A-Type Sequence Diagrams can be gathered and afterwards used as an elicitation technique for modeling the use cases. It can be counterintuitive to consider that use case diagrams can be refinements of sequence diagrams. It is possible if we take into consideration that the scenarios expressed in the *A-Type* Sequence Diagrams are built using the use-case candidates in the form of activities that will be executed and must be computationally supported by the system to be implemented. These activities in form of use cases are placed in the *A-Type* Sequence Diagram and associated with the corresponding actors and other used cases. These use cases are later arranged in use case diagrams after redundancy is eliminated and proper naming is given. The flow expressed by the sequences creates the rationale for discovering the necessary use cases to complete the process.

Use cases are modeled and textually described and used as input for the 4SRS. The execution of the 4SRS (N. Ferreira, et al., 2012) results in a logical architecture with a direct relation between the process-level use cases assured by the method's execution. Due to that, the logical architecture is derived, in a process- or in a product-level perspective, using the use case information to create AEs and their associations, in a properly aligned approach. The product level perspective is described in (R. J. Machado, et al., 2006b) and the process-level perspective in (N. Ferreira, et al., 2012; R. J. Machado & Fernandes, 2002). The process-level perspective imposes a different rationale to the method's execution. It is not our intention to describe the 4SRS method application. That is thoroughly done in the

literature (N. Ferreira, et al., 2012; R. J. Machado & Fernandes, 2002; R. J. Machado, et al., 2006b) and we use it as described in those works. For the sake of understandability, we only present a brief paragraph of the method's structure and application.

Step 1 -architectural element creation		Step 2 - architectural element elimination									Step 3 - packaging & aggregation	Step 4 - architectural element association	
		2i - use case classification	2ii - local elimination	2iii - architectural element naming	2iv - architectural element description	2v - architectural element: represented by	2v - architectural element: represent	2vi - global elimination	2vii - architectural element renaming	2viii - architectural element specification		4i - Direct Associations	4ii - UC Associations
{U2.1.}		cd											
{AE2.1.c}	Generated AE		T	IBS Analysis Pre-Start Decision	Browse the IBS and SBS Catalogs searching already existing IBS and SBS information with the intent of analyzing if the current business need isn't already fullfilled and if the ISOFIN Platform infrastructure supports the new implementation. ...	{AE2.1.c}	{AE1.11.i} {AE2.2.c} {AE2.5.c} {AE2.5.i}	T	Access Remote Catalogs	Allows browsing the available catalogs in the ISOFIN Platform (ISOFIN Application, IBS, and SBS). The user (Business User or the IBS Business Analyst) is allowed to search for information regarding the desired artifact and to select artifacts to use on his purposes. ...	{P2.2} IBS Analysis Decisions	{AE1.11.d1} {AE1.11.d2} {AE2.1.d}	{AE2.3.1.i} {AE3.2.i} {AE2.10.i} {AE2.11.i} {AE3.3.i} {AE3.7.1.i}
{AE2.1.d}	Generated AE		T	ISOFIN Functionalities Requirements List	Set of functional and non-functional requirements needed to fulfill identified business needs, intended system functionalities and all the constraints that may restrict design and implementation.	{AE2.1.d}		T	ISOFIN Functionalities Requirements List		{P2.1} IBS Requirements	{AE2.1.c}	
{AE2.1.i}			F										

Figure 13: Tabular Transformation of the 4SRS Method

The 4SRS method is organized in four steps to transform use cases into architecture elements:

Step 1 (architectural element creation) creates automatically three kinds of AEs for each use case: an *i-type* (interface), *c-type* (control) and *d-type* (data);

Step 2 (architectural element elimination) removes redundancy and automatically creates architectural elements, redundancy in the requirements passed by the use cases, and promotes the discovery of hidden requirements;

Step 3 (architectural element packaging & aggregation) semantically groups architectural elements in packages and also allows for representing aggregations (of, for instance, existing legacy systems);

Step 4 (architectural element association) aims to represent associations between the remaining architectural elements.

According to the previously described, the 4SRS method takes use cases representations (and corresponding textual descriptions) as input and (by recurring to tabular transformations) creates a logical architectural representation of the system. We present a subset of the tabular transformations in Figure 13. These tabular transformations are supported by a spreadsheet and each column has its own meaning and rules. Some of the steps have micro-steps; some micro-steps can be completely automatized. Tabular transformations assure traceability between the derived logical architecture diagram and the initial use case representations. At the same time tabular transformations makes it possible to adjust the results of the transformation to changing requirements. Tabular transformations are thoroughly described in (Nuno Ferreira, et al., 2012a; R. J. Machado, Fernandes, Monteiro, & Rodrigues, 2005) and in chapter 4 of this thesis.

As suggested by the V-Model represented in Figure 14, the models placed on the left hand side of the path representation are properly aligned with the models placed on the right side, i.e., *B-Type* Sequence Diagrams are aligned with *A-Type* Sequence Diagrams, and the logical architecture is aligned with the use case model.

Alignment between the use case model and the logical architecture is assured by the correct application of the 4SRS method. The resulting sets of transformations along our V-Model path provide artifacts properly aligned with the organization's business needs (which are formalized through Organization Configurations).

The V-Model representation promotes the alignment between the models of the problem domain and the models of the solution domain. The presented models are created in succession, by manipulating the information that results from one model to make decisions on how to create the other. In the descending side of the V-Model (left side of the V), models created in succession represent the refinement of requirements and the creation of system specifications. In the ascending side (right side of the V), models represent the integration of the discovered logical parts and their involvement in a cross-side oriented validating effort.

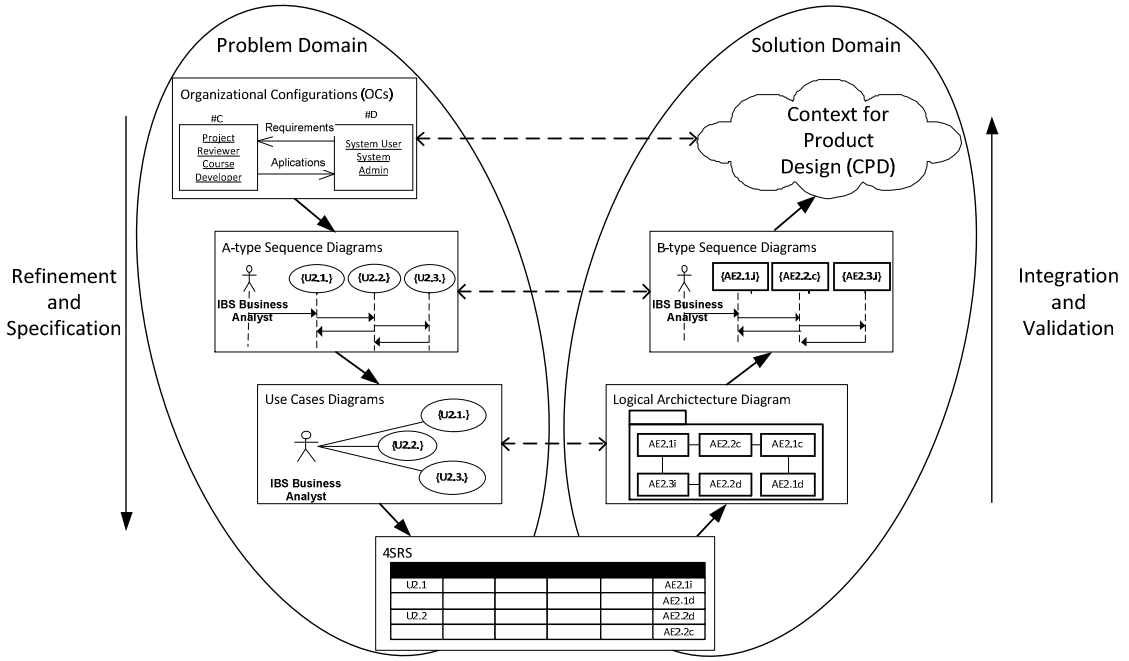

Figure 14: V-Model Adaption for Domain and Software Alignment

To assess the V-Model approach, we present a process of our case study, the ISOFIN project, as an example. The process under analysis, called “Create IBS”, deals with the creation of a new Interconnected Business Service (IBS). The inter-organizational relations required to create a new IBS are described under a new OC.

The definition of activities and actors required to create a new IBS are described in an *A-Type* Sequence Diagram. This diagram provides details of functionalities required to create an IBS, formally modeled in use cases. Use cases are used as input for a transformation method and the process-level logical architecture is derived. A *B-Type* Sequence Diagram allows for validation of the logical architecture required to create an IBS and also validates the requirement expressed in the corresponding *A-Type* Sequence Diagram. After the generation of these models, we assure that the “Create IBS” process is aligned with the stakeholder’s needs.

A V-Model SPEM representation

The development of software systems encompasses the application of several good practices and diversified knowledge as well as, eventually, the introduction of new

ideas or strategies. This results on the possibility of existence of several distinct approaches or ways for the development of a software system. In order to be able to express, establish, or organize the structure of activities inherent to a software development approach, it is convenient a standard way for expressing the process structure. In this context, Software and Systems Process Engineering Meta-Model 2.0 (SPEM 2.0), standardized by the Object Management Group (OMG), is a process engineering meta-model that provides process engineers with a conceptual framework for "*modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes*" (OMG, 2003a). In its current version (version 2.0), SPEM is defined as a meta-model as well as a UML 2 Profile (concepts are defined as meta-model classes as well as UML stereotypes); the UML 2 Profile is an alternative representation to the SPEM 2.0 meta-model. Attending to the usefulness of the SPEM specification, we use it to describe our approach. As such, attending to the work performed and products produced, Figure 15 presents a SPEM perspective of the V-Model based process used to derive the product-level requirements elicitation context. For this purpose, we use the typical SPEM representations for presenting the approach, *i.e.*, activities (*e.g.*, Use Case Modeling), artifacts (*e.g.*, Use Case Model), deliverables (Product-level Requirements Elicitation Context) and associations («input», «output», «predecessor» and «composition»).

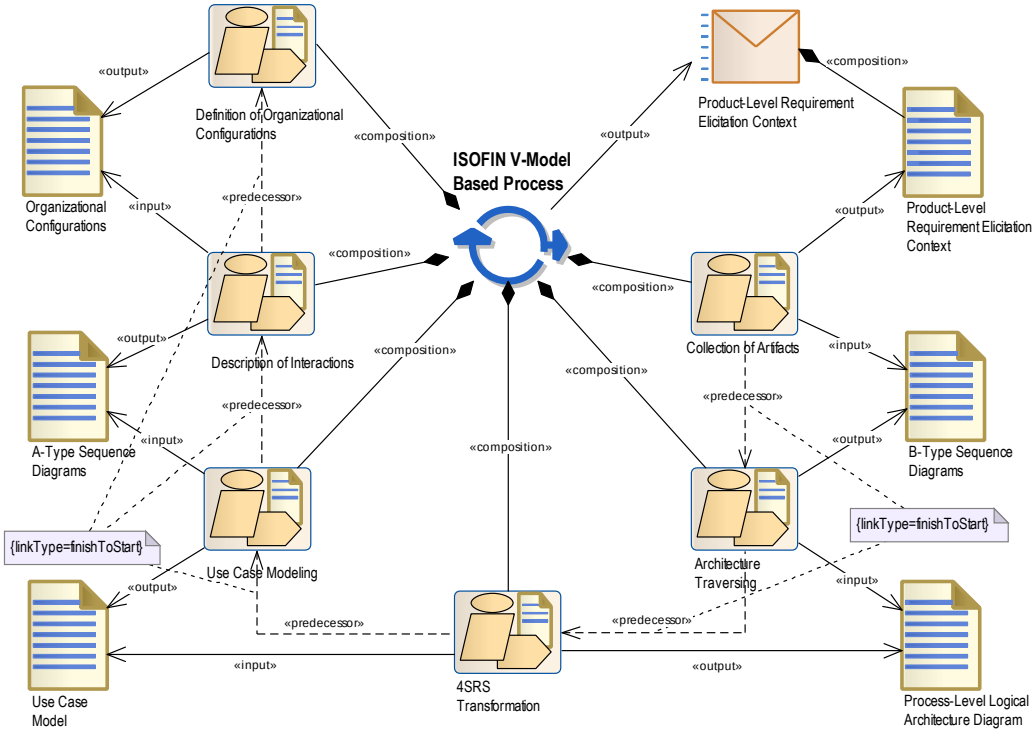

Figure 15: SPEM diagram of ISOFIN V-Model based process.

As depicted by Figure 15, the V-Model representation has the purpose of providing the deliverable *Product-Level Requirement Elicitation Context*. The main activities that make up the process are *Definition of Organizational Configurations*, *Description of interactions*, *Use Case Modeling*, *4SRS Transformation*, *Architecture Traversing*, and *Collection of Artifacts* (as indicated by the «composition» associations). These activities are sequentially performed in a way that an activity starts only when its predecessor activity has finished (as indicated by the «predecessor» dependencies). The activities use and produce (as indicated by «input» and «output» associations) artifacts, namely *Organizational Configurations*, *A-Type Sequence Diagrams*, *Use Case Model*, *Process-Level Logical Architecture Diagram*, *B-type Sequence Diagrams*, and *Product-Level Requirement Elicitation Context*.

3.4 The V-Model in the ISOFIN Project

We assess the applicability of the proposed approach with a case study that resulted from the process-level requirements elicitation in a real industrial case: the ISOFIN project (Interoperability in Financial Software) (ISOFIN Consortium, 2010).

The ISOFIN project encompasses eight institutions, ranging from universities, research centers and private software development companies for the bank and insurance domains (later introduced in 4.2 The ISOFIN Project). The stakeholders of this group had different backgrounds and expectations regarding the project outcome. These differences resulted in the lack of definitions for the requirements that the project's applications would support and even to a proper definition of a business model that the organizations that participate in the project would pursue.

Without an agreed or even a defined business model, it is not possible to define the context for the requirements elicitation of the products (applications) to be developed. There is, however, commonality in the speech of the stakeholders. They all contain hints on the kind of activities that the intended products would have to support – that is, they got beforehand an idea of the processes that the ISOFIN platform applications were required to computationally support.

The author of this work proposes a process-level approach to tackling the problem of not having a defined context for product design and researched on the models that the stakeholders agreed on to support the knowledge they had of the process-level requirements – Organizational Configurations, *A-Type* Sequence Diagrams and Use Cases. After executing the 4SRS method, properly adjusted to handle the process-level perspective, we were able to deliver a process-level logical architecture representation of the processes that are intended to be computationally supported by the applications to be developed. This approach created the context for product design, since the authors were able to identify the primary constructors that would support the processes. *B-Type* Sequence Diagrams appeared seamlessly in the

process. They represented the scenarios depicted in the *A-Type* Sequence Diagrams and also contributed to the validation of the process-level logical architecture diagram. These two aspects will be detailed later in this section.

The primary constructors that were identified correspond to the two main service types that the global ISOFIN architecture relies on: Interconnected Business Service (IBS) and Supplier Business Service (SBS). IBSs concern a set of functionalities that are exposed from the ISOFIN core platform to ISOFIN Customers. An IBS interconnects one or more SBSs and/or IBSs exposing functionalities that relate directly to business needs. SBSs are a set of functionalities that are exposed from the ISOFIN Suppliers production infrastructure. Figure 16 encompasses the primary constructors related to the execution of the platform (IBS, SBS and the ISOFIN Platform) available in the logical representations of the system: in the bottom layer there are SBSs that connect to IBSs in the ISOFIN Platform layer and the later are connected to ISOFIN Customers.

There are other constructors that were identified by using the V-Model approach and that support the operations for the execution of the ISOFIN Platform. These other constructors are, for instance, Editors, Code Generators, Subscriptions Management Systems, and Security Management Systems. These constructors support the creation and the operation of the primary constructors (IBS, SBS and ISOFIN Platform). The process-level architecture, later presented, depicts their interactions, major elements and organization.

By adopting the process-level perspective, we were able to create a system's representation that supports the elicitation of the process-level requirements from the stakeholders. This approach also allowed for creating the context for product design by representing the processes that must be supported by the applications to be developed. The next sections detail the V-Model process and exemplify the construction of the adopted models in real case study situations.

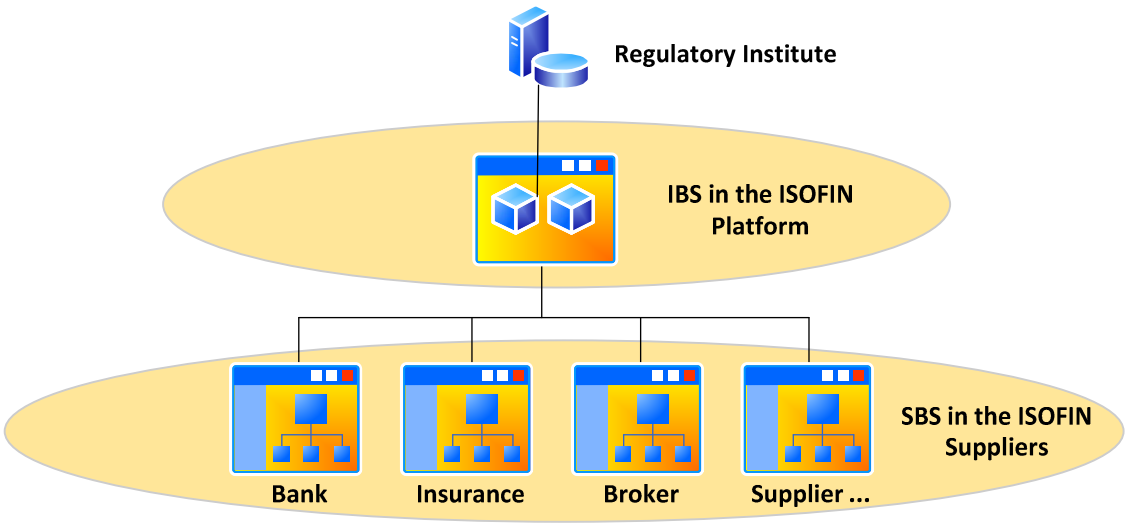

Figure 16: Desirable Interoperability in ISOFIN

Alignment between Organizational Configurations and Interactions

In a process-level approach, in opposition to the product-level approach, the characterization of the intended system gives a different perspective on the organizational relations and interactions. When defining a specific domain context, we consider that interactions between actors and processes constitute an important issue to be dealt. This section focuses on characterizing those interactions by using three different levels of abstraction, as depicted in Figure 17: OCs represent the first level; different types of Stereotyped UML Sequence Diagrams, presented as *A-Type* and *B-Type* Diagrams (later described in this section), represent the other two.

Today's business is based on inter-organizational relations (Evan, 1965), having an impact on an organization's business and software strategy (Barrett & Konsynski, Dec., 1982). We model a set of OCs to describe inter-organizational relations as a starting point to the definition of the domain-specific context. An OC models a possible inter-organizational relation, at a very high-level of abstraction and not considering lower-level processes and/or actors involved in the relation. For better deriving the domain-specific context, it is advisable to model as many OCs as required to describe, at least, the main relations as depicted by the stakeholders' domain-specific needs.

We present an example of an OC, for the purpose of assessing our approach, which has been characterized and applied in our case study (the ISOFIN project). Firstly, it is necessary to define the types of activities performed in the domain-specific context. By analyzing the types of activities, the execution of an IBS within a domain activity regards #A activities, while the creation of a new IBS regards #B activities:

(i) *#A Activities – Financial Domain Business Activities:* these are the delivered domain business activities regarding the financial institutions.

(ii) *#B Activities – ISOFIN Platform Services Integration*: these are the activities that relate to the integration of supplier services.

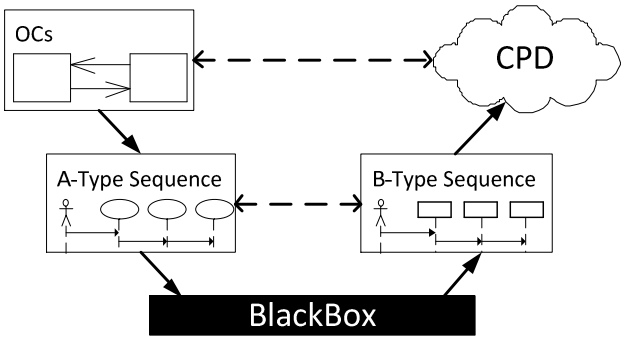

Figure 17: Organizational Configurations and Interactions Alignment

In order to characterize an organization, it is required to relate a set of roles to the performed activity type. Finally, the interactions between organizations are specified. In Figure 18, it is possible to depict the required relations between organizations in order to create an IBS and providing it to ISOFIN Customers. The professional profiles and the exchange of information between organizations are not relevant in this work, so only brief and simple examples are presented and only the types of activities are described.

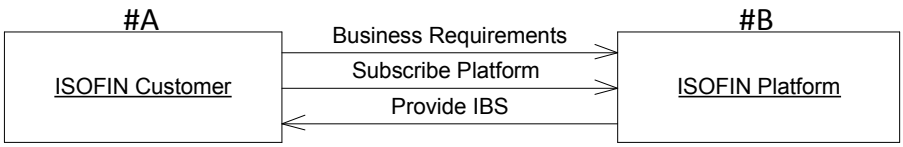

Figure 18: Organizational Configuration Example

In an early analysis phase, we need to define the relations between activities and actors, defined through interactions in our approach. Interactions are used during the more detailed design phase where the precise inter-process communication must be set up according to formal protocols (OMG, 2011b). An interaction can be displayed in a UML sequence diagram.

Traditional sequence diagrams involve system objects in the interaction. Since modeling structural elements of the system is beyond the scope of the user requirements, Machado *et al.* propose the usage of a stereotyped version of UML sequence diagrams that only includes actors and use cases to validate the elicited requirements at the analysis phase of system development (R. Machado, et al., 2007). We create *A-Type* Sequence Diagrams, as shown in Figure 19. In the example, we present some of the activities and actors required to create a new IBS. *A-Type* Sequence Diagrams also models the message exchange among the external actors and use cases (later depicted in Figure 23).

In Figure 19, we depict sequential flows of process-level use cases that refer to the required activities for creating an IBS. These activities are executed within #B activities, after receiving domain-specific requirements from ISOFIN Customers and before delivering IBS (interactions depicted in the OC of Figure 18).

The usage of *A-Type* Sequence Diagrams is required to gather and formalize the main stakeholder's intentions, which provide an orchestration and a sequence of some proposed activities. *A-Type* sequence diagrams realize the roles presented within an OC and instantiates them into activities. *A-Type* Sequence diagrams allow a pure functional representation of behavioral interaction with the environment and are appropriate to illustrate workflow user requirements (R. Machado, et al., 2007). They also provide information for defining and modeling use cases at a process-level perspective and frame the activities execution in time. Modeled diagrams must encompass all processes and actors.

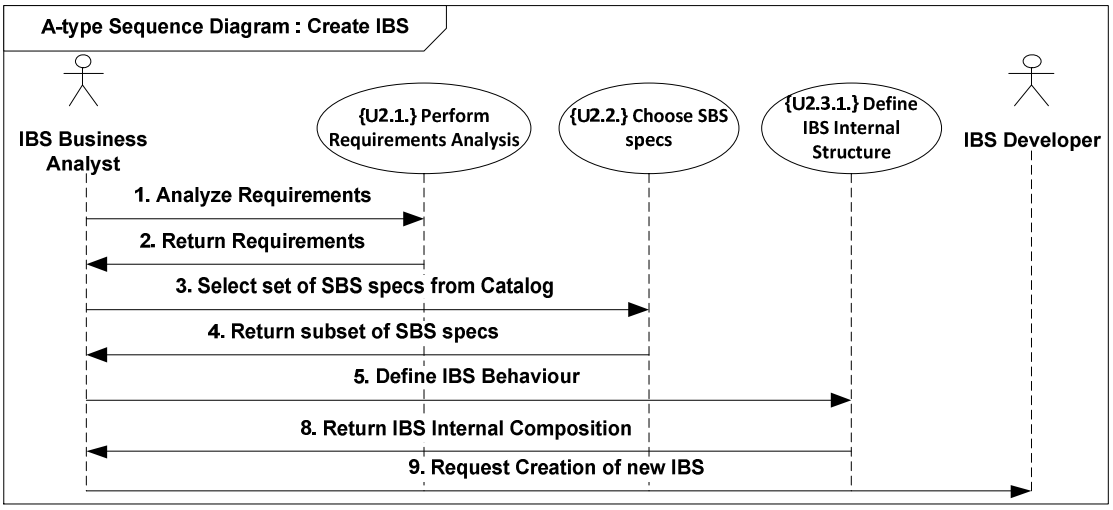

Figure 19: A-Type Sequence Diagram

One of the purposes of creating a software logical architecture is to support the system's functional requirements (Kruchten, 1995). It must be assured that the derived logical architecture is aligned with the domain-specific needs. On the one hand, the execution of a software architecture design method (e.g., 4SRS) provides an alignment of the logical architecture with user requirements. On the other hand, it is necessary to validate if the behavior of the logical architecture is as expected. So, in a later stage, after deriving a logical architecture, to analyze the sequential process flow of AEs (as shown in Figure 20), we adopt different stereotype of UML sequence diagrams, where AEs (presented in the logical architecture), actors and packages (if justifiable) interactions are modeled. In Figure 20, we present the same activities concerning creating an IBS but in a lower level of abstraction, closer to product design. *B-Type* Sequence Diagrams differ from the traditional ones, since they model the exchange of information between actors and logical AEs, thus they are still modeled at the system level.

Sequence flows between AEs are only possible if such a path is allowed within the logical architecture. *B-Type* Sequence Diagrams are used to validate the derived logical architecture, through the detection of missing architecture elements and/or associations to execute a given process within the derived logical architecture.

B-Type Sequence Diagrams can also be used to validate sequences in the previously modeled *A-Type* Sequence Diagrams, since the sequence flows between use cases must comply with the related sequence flows between AEs in *B-Type* diagrams. This validation is considered essential in our V-Model process. There must be modeled as many *A-Type* sequence diagrams as necessary to fully represent the business context detail. *B-Type* sequence diagrams must be modeled to match corresponding business requirements given in *A-Type* sequence diagrams and there must be enough *B-Type* sequence diagrams to ensure that all AEs of the logical architecture are used.

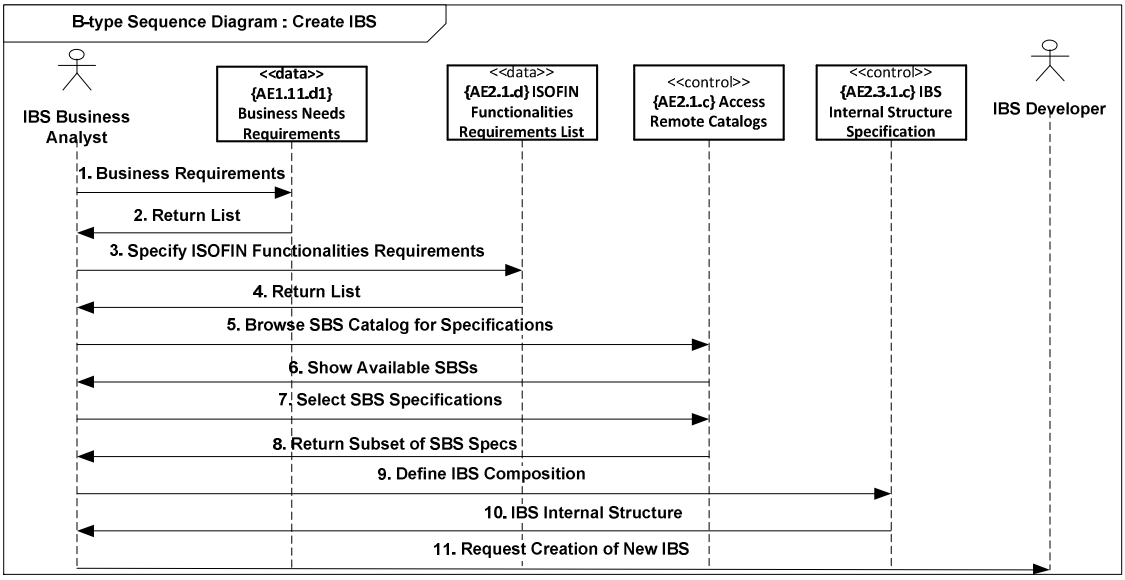

Figure 20: B-Type Sequence Diagram

An UML Metamodel Extension for *A-Type* and *B-Type* Sequence Diagrams

The usage of *A-Type* and *B-Type* sequence diagrams in our approach is perfectly harmonized with UML sequence diagram's original semantics, as described in the UML Superstructure (OMG, 2011b). On the left hand side of Figure 21, we present some of the classes of the UML metamodel regarding sequence diagrams (in the *Interactions* context of the UML Superstructure). As *A-Type* and *B-Type* sequence diagrams differ from typical sequence diagrams in the participants of the interactions, the usage of these diagrams regards the *Lifeline* class. A lifeline represents an

individual participant in the *Interaction*. The *Lifeline* notation description presented in the UML Superstructure details that the lifeline is described by its <*connectable-element-name*> and <*class_name*>, where <*class_name*> is the type referenced by the represented *ConnectableElement*, and its symbol consists in a "head" followed by a vertical line (straight or dashed). A *ConnectableElement* (from *InternalStructures*) is an abstract metaclass representing a set of instances that play roles of a classifier. The *Lifeline* "head" has a shape that is based on the classifier for the part that this lifeline represents.

The participants in the interactions in *A-Type* sequence diagrams are use cases and in *B-Type* sequence diagrams are architectural elements. Regarding *A-Type* sequence diagrams, the UML Superstructure clearly defines a class for use cases. However, regarding *B-Type* sequence diagrams, architectural elements are not considered in any class of the UML metamodel and, despite some similarities in semantics, are different from UML components. Such situation leads to the necessity of defining a stereotype *«Architectural Element»* for the *NamedElement* class (depicted in the right side of Figure 21). AEs refer to the pieces from which the final logical architecture can be built and currently relate to generated artifacts and not to their connections or containers. The nature of architectural elements varies according to the type of system under study and the context where it is applied.

Like the *ConnectableElement* class, *UseCase* class is also generalized by *NamedElement* class. The information regarding abstract syntax, concrete syntax, well-formedness and semantics (Atkinson & Kuhne, 2003) of *UseCase* class and the context in which we defined the stereotype *«Architecture Element»* does not express any condition that restricts them of being able to act as a *ConnectableElement*.

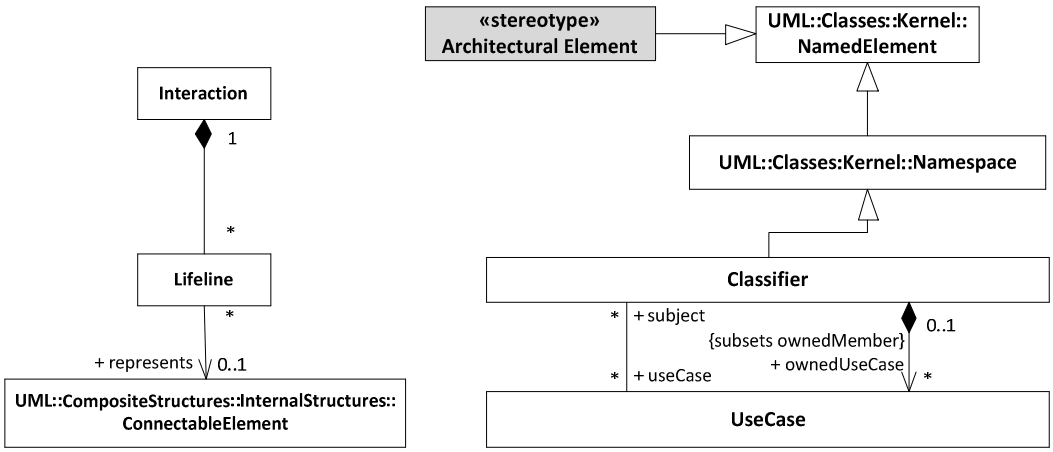

Figure 21: The Proposed Extension to the UML Metamodel for Representing A-Type and B-Type Sequence Diagrams

Derivation of Process-Oriented Logical Architectures

In this section, we present the process-level logical architecture derived using the 4SRS method. The process-level application of the 4SRS method used in this example is detailed in (Nuno Ferreira, et al., 2012a; R. J. Machado, et al., 2005) and in the next chapter, and so we treat the 4SRS like a black box in the V-Model description as represented in Figure 22. The method takes use cases as input, since they reflect elicited requirements and functionalities. Use cases are derived from *A-Type* Sequence Diagrams and from the OCs.

Gathering *A-Type* Sequence Diagrams can be used as an elicitation technique for modeling use cases, after eliminating redundancy and give a proper name to the use cases used in the sequences. All use cases defined in the *A-Type* Sequence Diagrams must be modeled and textually described in the use case model in order to be used in the 4SRS method.

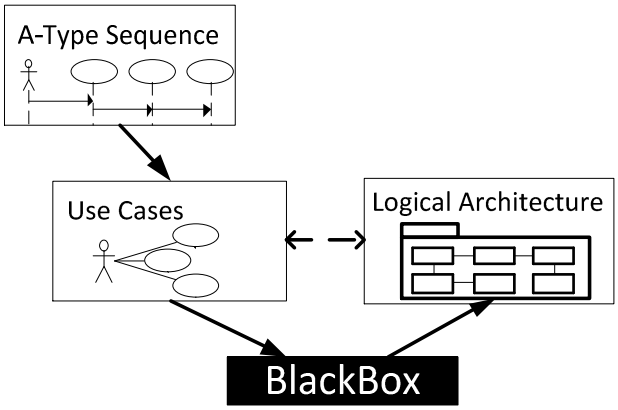

Figure 22: Derivation of Process-Oriented Logical Architectures

The use case model specifies the required usages of the ISOFIN Platform. In Figure 23, we present a subset of such usages, regarding the development of functionalities to be accessed by ISOFIN Customers. These use cases intent to capture the requirements of the system that where initially expressed through OCs in the business perspective and later represented using A-Type sequence diagrams.

Use cases, in the process-level perspective, portray the activities (processes) executed by persons or machines in the scope of the system, instead of the characteristics (requirements) of the intended products to be developed. It is essential for use case modeling to include textual descriptions that contain information regarding the process execution, preconditions and actions, as well as their relations and dependencies.

The 4SRS method execution results in a logical architecture diagram, presented in Figure 24. This logical architecture diagram represents the architectural elements, from which the constructors can be retrieved, their associations and packaging. The architectural elements derive from the use case model by the execution of the 4SRS method. In this representation, there are packages that represent, for example, subscription activities in *ISOFIN Platform Subscriptions Management*, and the SBS and IBS development in *{P1.} SBS Development* and *{P2} IBS Development* respectively. Inside both *{P1}* and *{P2}* it can be found the requirements activities, the analysis decisions and the generators for the major constructors (IBS and SBS). It is also possible to observe that each SBS (in *{P1.4} SBS*) and IBS (in *{P2.4} IBS*) result from

activities able to generate their code. This process-level logical architecture shows how activities are arranged so the major constructors are made available to ISOFIN Customers within the intended IT solution.

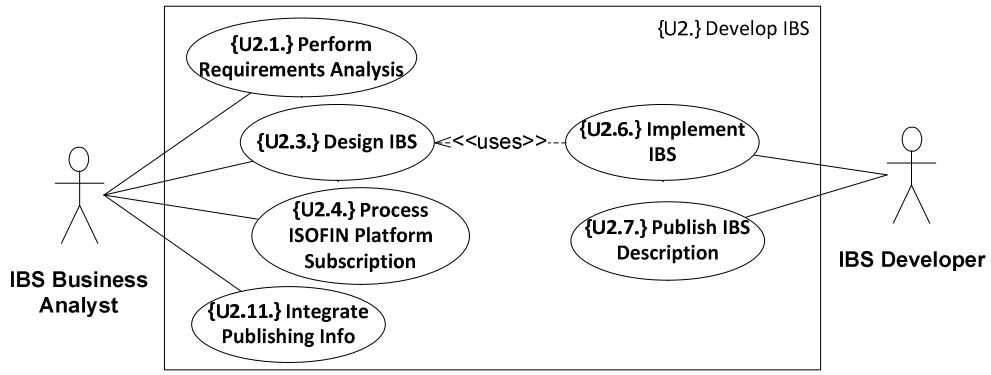

Figure 23: Subset of the Use Case Model from the ISOFIN Project

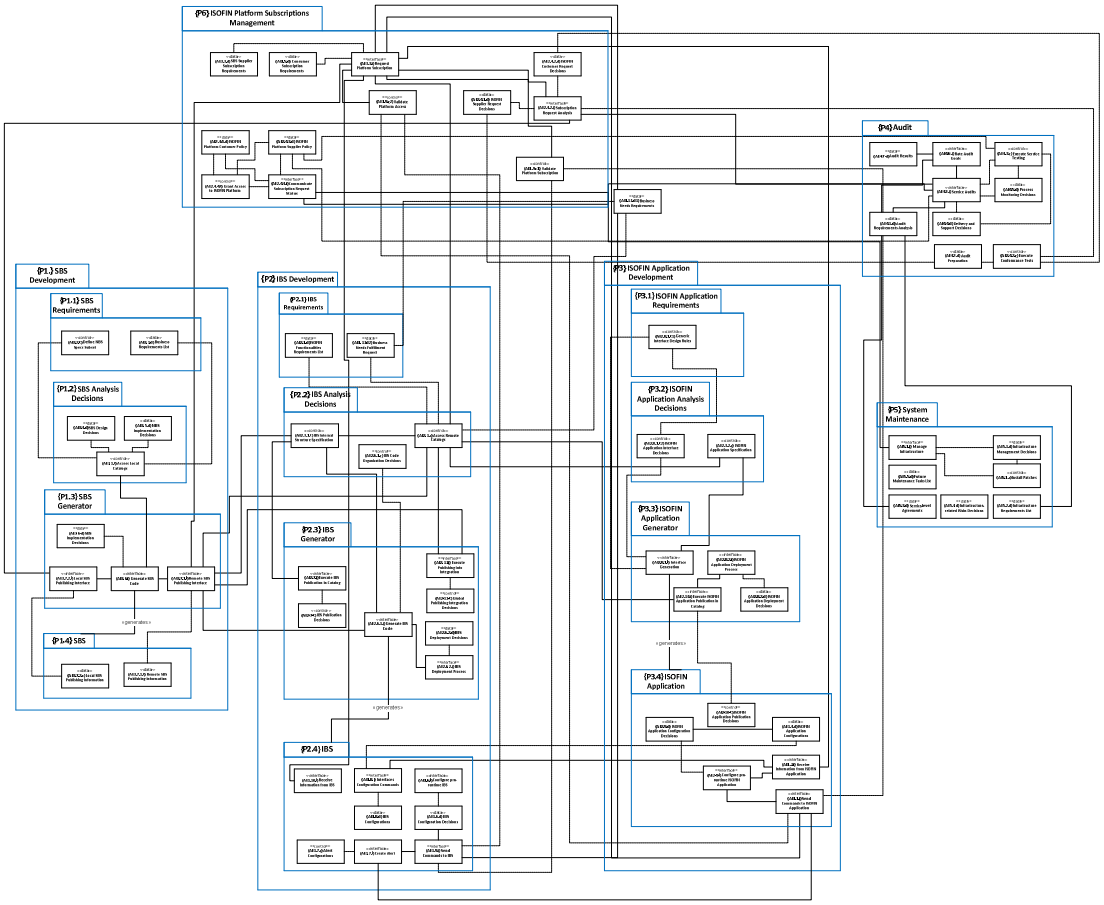

Figure 24: ISOFIN Process-level Logical Architecture

Figure 24 depicts the process-level logical architecture for the ISOFIN project and contains nearly eighty architectural elements. This figure is intentionally not zoomed in (and thus not readable), just to show the complexity of the ISOFIN project that has justified the adoption of process-level techniques to support the elicitation efforts. A proper zoom of the architecture can be found in Figure 25, detailing some of its constructors.

3.5 V-Model Considerations and Comparison with Related Work

For creating a context for IT product design, the V-Model presented in this chapter encompasses a set of artifacts through successive derivation. Our approach is different from existing ones (Bayer, et al., 2001; Kang, et al., 1998; D. M. Weiss & Lai, 1999), since we use a process-level perspective. Not only do we manage to create the context for product design, but we also manage to align it with the elicited domain-specific needs.

Our stereotyped usage of sequence diagrams adds more representativeness value to the specific model than, for instance, the presented in Krutchen's 4+1 perspective (Kruchten, 1995). This kind of representation also enables testing sequences of system actions that are meaningful at the software architecture level (Bertolino, Inverardi, & Muccini, 2001). Additionally, the use of this kind of stereotyped sequence diagrams at the first stage of analysis phase (user requirements modeling and validation) provides a friendlier perspective to most stake-holders, easing them to establish a direct correspondence between what they initially stated as functional requirements and what the model already describes.

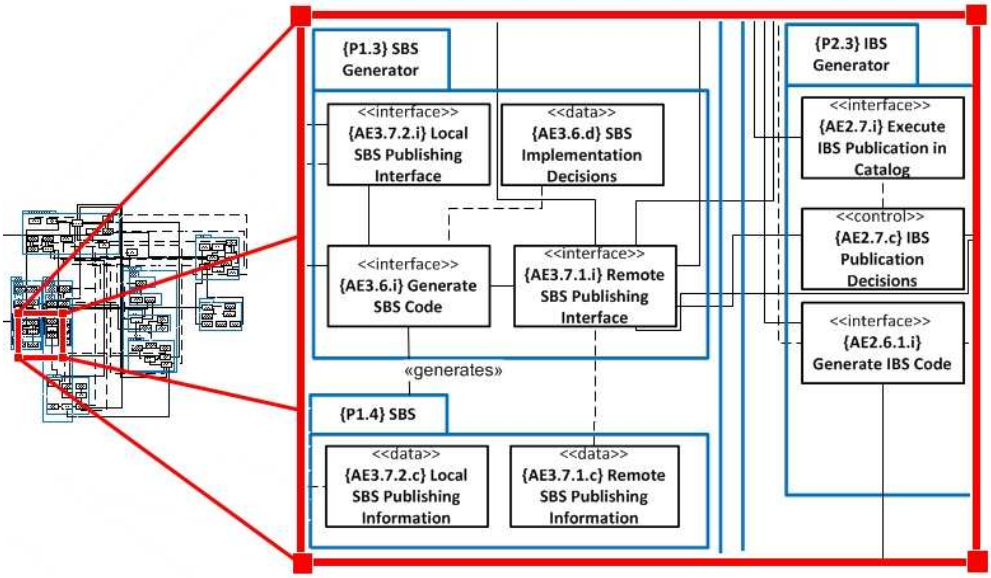

Figure 25: Subset of the ISOFIN Process-level Logical Architecture

In the ISOFIN project the usage of *A-Type* Sequence Diagrams also contributed to creating a standard representation for the scenarios that are intended to be supported. The *B-Type* Sequence Diagrams that derived from the *A-Type* Sequence Diagrams allowed designers to validate the logical architecture against the given scenarios and at the same time represent the process flow depicted in the architectural elements.

Regarding alignment approaches that use a set of models (like GQM+Strategies (Basili, et al., 2010), Balanced Scorecards (Kaplan & Norton, 1992) or COBIT (Information Technology Governance Institute (ITGI), 2012)), all relate to aligning the domain-specific concerns with software solutions. As far as the authors of this work are concerned, none of the previous approaches encompasses processes for deriving a logical representation of the intended system processes with the purpose of creating context for eliciting product-level requirements. Those approaches have a broader specification concerning risk analysis, auditing, measurement, or best practices in the overall alignment strategy.

The *Project Charter* regards information that is necessary for the ongoing project and relates to project management terminology and content (Project Management Institute, 2008). This document encompasses information regarding the project

requirements in terms of human and material resources, skills, training, context for the project, stakeholder identification, amongst others. It explicitly contains principles and policies of the intended practice with people from different perspectives in the project (analysis, design, implementation, etc.). It also allows for having a common agreement to refer to, if necessary, during the project execution.

The *Materials* document contains the necessary information for creating a presentation of the project. It regards collected seed scenarios based on *OCs (or Mashed UCs), A-type sequence diagrams* and (business or software) *Use Case Models*. Parts of the *Logical Architecture* model are also incorporated in the presentation that will be presented to the stakeholders (including software engineers responsible for implementation). The purpose of this presentation is to enlighten the team about the logical architecture and propose the seed scenarios to discussion and create the *B-type sequence diagrams* based on presented information.

The *Issues* document supports information regarding the evaluation of the presented logical architecture. If the logical architecture is positively assessed, we can assume that we reached consensus to proceed into the macro-process. If not, using the *Issues* document it is possible to promote a new iteration (as seen on Figure 26) of the corresponding V-Model execution to adjust the previously resulting logical architecture to make the necessary corrections to comply with the seed scenarios. Main causes for this adjustment are:

(i) suboptimal decisions that were made in the corresponding 4SRS method execution;

(ii) *B-type sequence diagrams* not complying with all the *A-type sequence diagrams*;

(iii) created *B-type sequence diagrams* not comprising the entire *logical architecture*;

(iv) the need to explicitly placing a design decision in the logical architecture diagram, usually done by using a common architectural pattern and

injecting the necessary information in the use case textual descriptions that are input for the 4SRS.

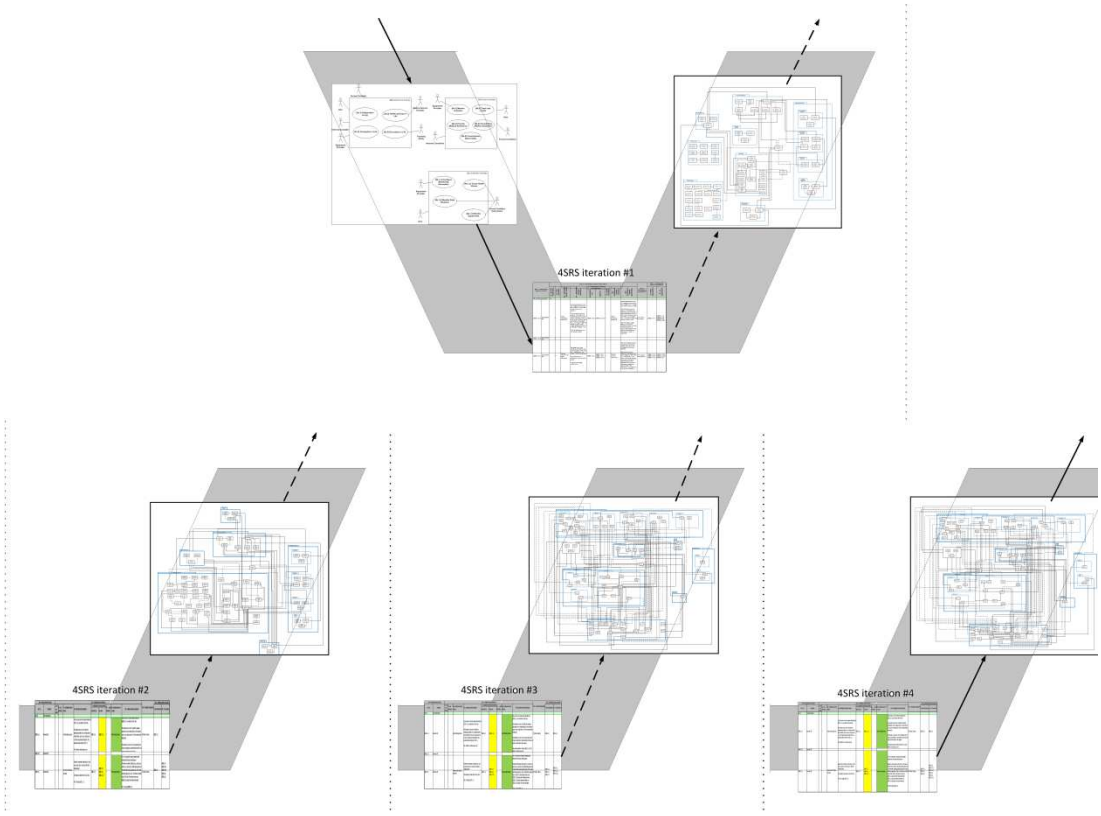

Figure 26: Iterations for producing a logical architecture

The adjustment of the logical architecture diagram (by iterating the same V-Model) suggests the construction of a new use case model or, in the case of a new scenario, the construction of new *A-type sequence diagrams*. The new use case model captures user requirements of the revised system under design. At the same time, through the application of the 4SRS method, it is possible to derive the corresponding logical architecture diagram.

3.6 Assessment of the V-Model

Having a structured method makes the analysis repeatable and at the same time helps ensuring that the same set of validation questions are placed in early development stages. With the purpose of assuring the attained logical architecture representation is tenable, we chose to validate it and the underlying V-Model, using the Active Reviews for Intermediate Designs (ARID) method (P. C. Clements, 2000).

Our concerns relate to discovering errors as soon as possible, inconsistencies in the logical architecture or even inadequacies with the elicited requirements, expressed through the *A-Type* Sequence Diagrams (scenario requirements) and use case models (specific process-level requirements).

The ARID method is a combination of Architecture Tradeoff Analysis Method (ATAM) with Active Design Review (ADR). ATAM is a refined and improved version of Software Architecture Analysis Method (SAAM) that helps reviewing architectural decisions having the focus on the quality attributes requirements and their alignment and satisfaction degree of specific quality goals. The ADR method targets incomplete (under development) architectures, performing evaluations on sections of the global architecture. Those features made ARID our method of choice regarding the evaluation of the in-progress ISOFIN logical architecture.

In Figure 27 we present a simplified diagram that encompasses major ARID representations required to align with our V-Model models.

We now present our adapted ARID specific models like *Project Charter*, *Materials* and *Issues*. ARID requires that a project context is defined, containing information regarding the identification of the design reviewers. We have represented such information using the *Project Charter* box as used in project management (Project Management Institute, 2008) terminology. The *Materials* box represents the supporting documentation, like presentation that needs to be done to stakeholders,

seed scenarios and meeting agenda. *Issues* relates to a checklist that includes but is not limited to notes concerning the presentation, the presented logical architecture, newly created scenarios and validation scenarios. The *issues* representation is used to identify flaws in the logical architecture diagram and therefore promoting a new iteration of the 4SRS method.

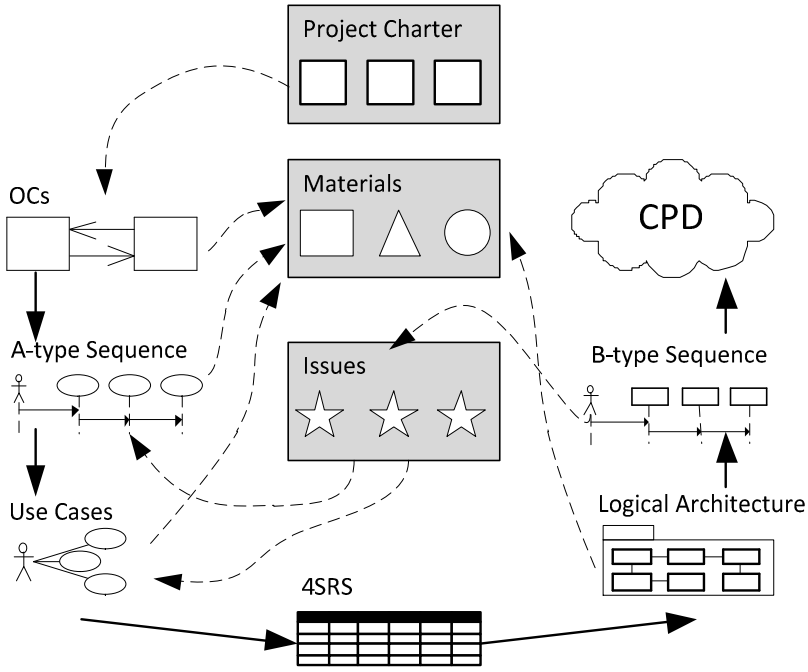

Figure 27: ARID and the V-Model Intertwining

ARID was used in the ISOFIN project to assess the process-level logical diagram as a result of the V-Model approach. The *Project Charter* was created with the initial requirements the project, the stakeholders, the teams, budget, timings, intended context and others, that influence directly or indirectly the project's execution. Having this in mind, it is possible to represent the Organizational Configurations (high-level interactions in the domain of analysis). The intended context described in the *Project Charter* gives hints on the domain interactions and the stakeholders are able to provide more information about the roles and activity types that must be supported.

The *Materials* model stores information regarding the created Organizational Configurations, *A-Type* Sequence Diagrams, Use Case models and the derived Logical

Architecture. This information is useful for presenting the project, the rationale that sustained the creation of the used models and the scenarios that are used as basis for the requirements elicitation.

Using the information of the *Materials* model a presentation is made to the stakeholders with the intention of assuring that all the initial requirements are met, in the form of scenarios. A scenario is represented by an *A-Type* Sequence Diagram and, for each, is discussed and presented the path that must be followed in the Logical Architecture diagram to accomplish that given scenario. This path is represented using *B-Type* Sequence Diagrams. Any problem with the path (architectural elements missing, associations not possible to accomplish, bad routes, etc.) are stored in the *Issues* model and a new iteration of the 4SRS method is executed. This iteration can be promoted by changing the initial scenarios (*A-Type* Sequence Diagrams) or the initial requirements (use cases). The process- and product-level iterations of the 4SRS are found on Appendix A and Appendix B.

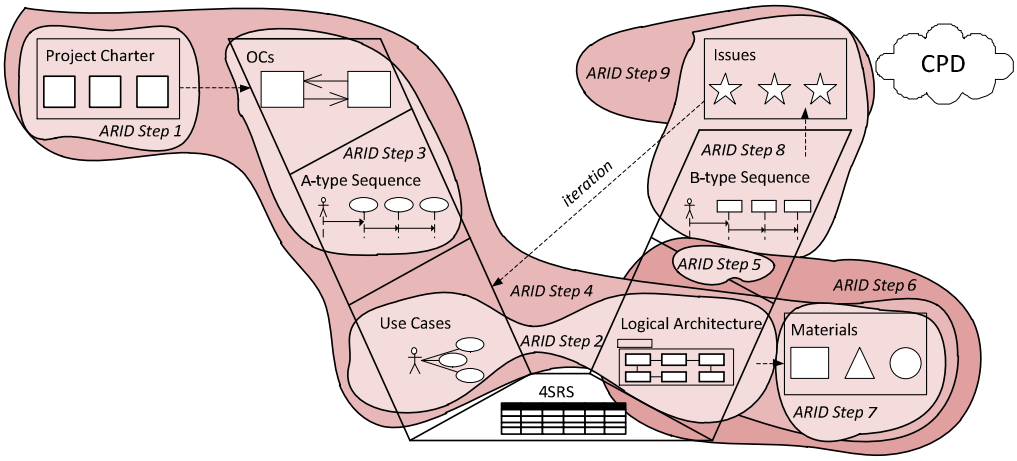

Figure 28: ARID Steps in the V-Model

Figure 28 shows the coverage of each ARID step with respect to the V-Model artifacts. There are also represented ARID specific artifacts like *Project Charter*, *Materials* and *Issues*. ARID requires that a project context is defined, containing information regarding the identification of the design reviewers.

The ARID method is divided in two phases: *Rehearsal* and *Review*. The *Rehearsal* phase was adapted to the ISOFIN project context as follows:

ARID Step 1. *Identify the Reviewers*: We chose 10 reviewers from the ISOFIN project design team. We chose 2 stakeholders from each of the 5 entities that were involved directly or indirectly with the design decisions.

ARID Step 2. *Prepare the design briefing*: For the purpose of demonstrating the design we prepared a presentation showing the logical architecture diagram as a background and the OCs, *A-type sequence diagrams* and use cases that were used to derive each part of the logical architecture.

ARID Step 3. *Prepare the seed scenarios*: Associated with each OC and *A-type sequence diagram* set there was defined a feasible scenario in a total of 10 scenarios, included in the presentation with the purpose of rising questions regarding the presented logical architecture.

ARID Step 4. *Prepare the materials*: We scheduled a meeting with all the stakeholders (reviewers), and distributed to them the presentation and the meeting agenda.

The second ARID phase, *Review*, was adapted to the ISOFIN context as follows:

ARID Step 5. *Present ARID*: We have presented the steps of ARID to the stakeholders in order to create a context for the method execution.

ARID Step 6. *Present the design*: Prepared materials, scenarios and logical architecture were presented. The reviewers followed the rule of not questioning the presentation contents or making any improvement comment. Only clarification questions where allowed for the sake of better understanding the materials. One of the design team members was assigned to take notes of any occurrence of references to deliverables that where not yet available. These notes helped to show potential issues in the logical architecture diagram that needed to be taken care of in a next iteration.

ARID Step 7. *Brainstorm and prioritize scenarios*: Reviewers presented the new scenarios that solved problems they were dealing. Those scenarios where put in the pool with the seed scenarios. We analyzed that pool to exclude duplicates and overlaps. At this moment we had 16 feasible scenarios and formalized the *A-type*

sequence diagrams. Each reviewer was allowed a vote equaling 30% the number of scenarios. That vote could be allocated on any scenario or scenarios they wanted to be discussed.

ARID Step 8. *Apply the scenarios*: Scenarios that won where used to test the logical architecture diagram for suitability. We began with the scenario that gathered the most votes. The reviewers, working as one and having that scenario in mind, designed the *B-type sequence diagrams* that corresponded to the scenario under analysis. These diagrams were used to check if the logical architecture diagram solved the problem arisen by the scenario. The team member allocated to taking notes recorded the *B-type sequence diagrams*. At any time, the design team responsible for the logical architecture intervened to help. We have established a four-hour window to execute this step and in that time, we managed to create just as many *B-type sequence diagrams* as *A-type sequence diagrams*. This is considered the necessary condition for the architecture validation.

ARID Step 9. *Summarize*: As a last step we reviewed the notes and inquired the participants concerning the exercise. All this feedback helped improve the logical architecture diagram and define a check-list of subjects that required attention and needed to be attended before moving on to design or implementation.

Figure 28 shows the issues discovered in *step 8* and summarized in *step 9*. The discovery of issues may promote a new iteration of the 4SRS method, depending on their relevance for the understanding of the intended final result or severe flaws are detected in the logical architecture diagram. A severe flaw is detected when it is not possible to fully create a *B-type sequence diagram* traversing and connecting all the AEs in the logical architecture diagram that comply with the defined *A-type sequence* diagrams. In our study we required the execution of four iterations in the 4SRS method before the logical architecture passed the ARID assessment.

3.7 Conclusions

In this chapter, we have presented a process-level approach to creating context for product design based on successive derivation of models in a V-Model representation. We have used *A-Type* sequence diagrams as a bridge from domain-specific needs to the first system requirements representation, *B-Type* sequence diagrams are used as validation for *A-Type* sequence diagrams and the logical architecture diagram. The used models represent the system in its behavior, structure and expected functionalities.

The approach assures that validation tasks are performed continuously along with the modeling process. It allows for validating: (i) the final software solution according to the initial expressed requirements; (ii) the *B-Type* sequence diagrams according to *A-Type* sequence diagrams; (iii) the logical diagram by traversing it with *B-Type* sequence diagrams.

Due to the use of a process-level perspective instead of the typical product-level perspective, our approach might be considered to delay the delivery of usable results to technological teams. Although, we are formalizing a model called process-level architecture that is the basis for the domain-specific and software alignment, assuring the existence of one effective return on the investment put into action during that so-called delay. This decreases, namely, the probability of project failure or the need for post-deployment product rework. These advantages were well appreciated by the designers and developers that used the process-level logical architecture artifacts in their work. Also, they were presented with the rationale that was made, in terms of processes that must be supported by the applications they developed.

The presented approach compels the designers and developers to provide a set of models that allow the requirements to be sustainably specified. Also, using multiple viewpoints, like logical diagrams, sequence diagrams or other artifacts, contribute to a better representation and understanding of the system. Each created model in the

V-Model takes knowledge from the previously created model as input. Since they are created in succession, the time required to derive a given model, for the same degree of representativeness, is smaller than the previous one. For example, *A-Type* Sequence Diagrams take as input information from the OC model. This means that the context for building *A-Type* Sequence Diagrams is created by the OC model.

On the left-side of the process, the OC model represents processes at a very high-level. The refinement of requirements lowers the abstraction level. In similar context to the one presented in our case study (not having a defined context for product design), this approach is capable of starting with very high-level models and end with low-level information. Also, deriving the models allows uncovering requirements that weren't initially elicited.

As recommended by the ARID method, the V-Model is able to conduct reviews regarding architectural decisions, namely on the quality attributes requirements and their alignment and satisfaction degree of specific quality goals that are imposed to the created scenarios (*A-Type* Sequence Diagrams). These quality attributes reviews were not explicitly done in the ISOFIN project. Instead, those requirements were imbued in design decisions related to the logical architecture.

Unfortunately, our approach could not be compared with other approaches within the same case study. It was also not possible to add a fresh team on the project just to perform other approach for comparison reasons.

It is a common fact that domain-specific needs, namely business needs, are a fast changing concern that must be tackled. Process-level architectures must be in a way that potentially changing domain-specific needs are local in the architecture representation. Our proposed V-Model process encompasses the derivation of a logical architecture representation that is aligned with domain-specific needs and any change made to those domain-specific needs is reflected in the logical architectural model through successive derivation of the supporting models (OCs, *A*- and *B-Type* Sequence Diagrams, and Use cases). In addition, traceability between those models is built-in by construction, and intrinsically integrated in our V-Model process.

Chapter 4

Yet Another 4SRS

Chapter Contents

4

Yet Another 4SRS

"A problem well stated is a problem half solved."

Charles F. Kettering

This chapter presents the process-level perspective of the 4SRS method. This perspective allows the creation of a logical architecture representation of the information system based on the requirements initially expressed by the business stakeholders of the system under developed. These requirements are not always clearly defined and the usage of the process-level 4SRS contributes for their clarification.

4.1 Introduction

A logical architecture provides the conceptual foundation on which other type of architectures (for instance enterprise architectures) can be built upon. This architecture can be represented in a model (diagram) that provides a centralized view of all processes and systems that supports the intended final solution. Such a representation helps the teams responsible for the enterprise architecture to ensure

that they are addressing the relevant all areas necessary for maximum effectiveness and achievement of the initially defined purposes.

The design of software architectures for systems to be executed in any target environment (for instance, cloud computing or service-oriented platforms) brings many difficulties to system architects. Instead of designing an entire enterprise architecture based on user requirements traditionally defined in a product-level perspective, in this chapter, we propose the use of a process-level perspective for the requirements definition and design of the logical model of the intended final architecture. This assumption is built upon the premise that such an approach contributes to a more accurate definition of the desired final product requirements (software architecture) and understanding of the project scope. This is mainly useful when the project stakeholders to not have enough confidence in the project information to decide key issues, like the final logical architecture, the intended business model or even the parts (components, services, connection points or databases) that make the final system.

In our presented approach, we use the term *process*. This term, in a generic context, is hard to define. In the definition given in (Davenport, 1993), a process is a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs. Our process definition aligns with the previous one, giving emphasis to another aspect: a process is executed by *someone* and thus, that also must be accounted for. Therefore, a process is, in our definition, a set of interconnected or interrelated activities, with a beginning and an end, executed by someone, with the purpose of transforming a set of inputs into outputs.

This section describes the extensions introduced into the 4SRS method to be adopted at the process-level perspective in large-scale projects using as case study the ISOFIN project (ISOFIN Project Consortium, 2010) for achieving a representation of the system's logical architecture. The resulting work is presented in (Nuno Ferreira, et al., 2012a) and in (Ferreira, Santos, Machado, Fernandes, et al., 2013). Since the obtainment of a logical architecture based on system requirements is a well-

documented task in the works of (Fernandes, Machado, Monteiro, & Rodrigues, 2006; R. J. Machado & Fernandes, 2002; R. J. Machado, et al., 2005), and due to our knowledge in the 4SRS method presented in those works, we decided to use it in order design the logical architecture of the ISOFIN Platform.

The process-level perspective allows capturing the intentionality's presented in the desired activities that the platform will sustain and at the same time resolve the ambiguity in the product definition that obscure the borderline of actuation of the ISOFIN Project.

The 4SRS method was first defined and detailed in (R. J. Machado, et al., 2006b; R. J. Machado, et al., 2005). The described extensions are focused on a process-level perspective to deliver a logical architectural model. This logical architectural model contributes to the context definition of a proper requirements elicitation. We additionally illustrate the applicability of the proposed approach in the real industrial case, the ISOFIN project, later presented. In the presented real industrial case, the process-level 4SRS is used to create the necessary context to elicit the requirements for designing an architecture capable to be implemented in the three typical cloud-layers: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), as defined in (NIST, 2009). The transformation of such context into product-level requirements is presented in Chapter 5 of this thesis.

This chapter begins with the ISOFIN Project presentation and its framing as the problem context from where the delving of solutions began. In the next sections, we introduce the method of choice for creating a logical architecture representation of the intended system in two perspectives: the product- and process-level perspectives of the 4SRS method. We end this chapter with a presentation of the results of the application of the method in the process-level perspective and introduce the next steps: the transition from process to product to achieve a service-oriented logical architecture representation.

4.2 The ISOFIN Project

The ISOFIN project aims to deliver a set of coordinating services in a centralized infrastructure, enacting the coordination of independent services relying on separate infrastructures. The resulting ISOFIN platform will desirably contribute for the semantic and application interoperability between enrolled financial institutions (Banks, Insurance Companies and others), that is, between the ISOFIN Customers and the ISOFIN Suppliers, as depicted in Figure 29.

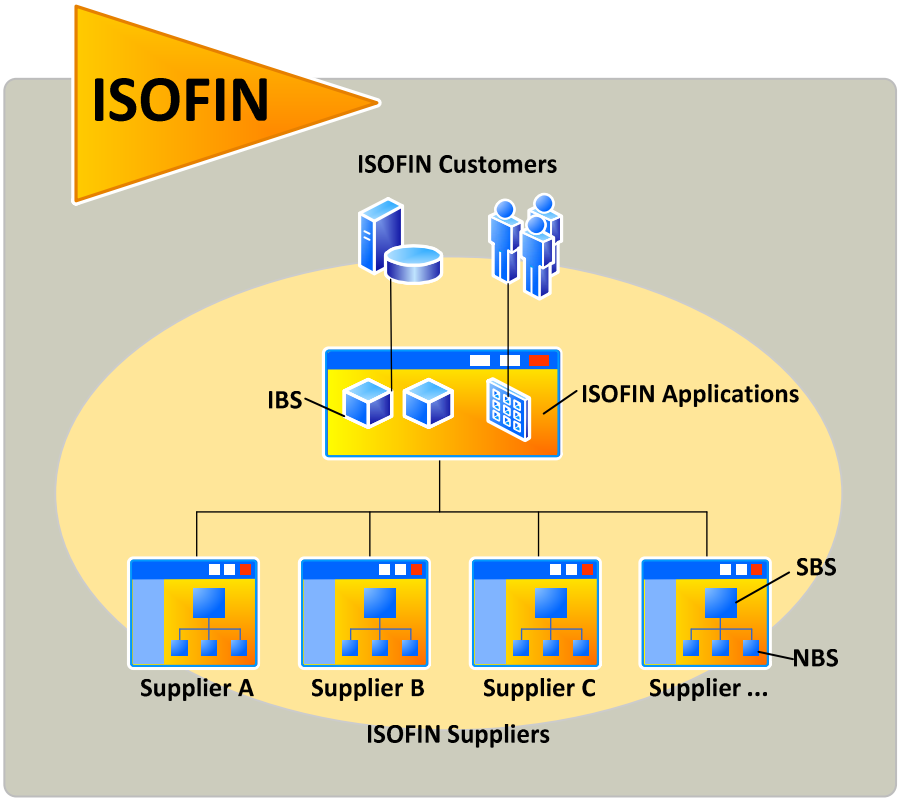

Figure 29: ISOFIN Main Constructors

In this section, we present an overview of the ISOFIN terminology based on Figure 29. This terminology was created as a result of the execution of the process-level 4SRS method described in this section as it is presented to create an adequate context for the method, techniques and approaches that are described in this work and specially in the process-level 4SRS described in this chapter.

The global ISOFIN architecture relies on two main service types, Interconnected Business Service (IBS) and Supplier Business Service (SBS). Alongside those two main service types, the global ISOFIN architecture references the following:

- NBS (Native Business Service): The already existing software installed and exploited within the context of each ISOFIN Supplier. NBS examples (from the insurance core application context) are GetRolesInContract, GetAgreementAccounts, and CreateRecurringTransfer.

- SBS (Supplier Business Service): A set of functionalities exposed from the ISOFIN Supplier infrastructure. A SBS is the result of externalizing of one or more NBSs. The creation of SBSs is dependent upon the ISOFIN Supplier strategy (business needs) and/or legal requirements that it is required to fulfill. If a SBS consists in the externalization of a single NBSs, the analysis and design of that SBS is negligible since the development effort is done mainly in the NBS, leaving to the SBS the externalization effort. In the case where a SBS is the result of interconnecting more than one NBS, the analysis and design efforts must be accounted since there is a need to align the SBS with business needs (elementary NBSs that compose that SBS do not fulfill, by themselves, any business need). In this case, the implementation effort is also higher since it may be necessary to interconnect NBSs that reside in different execution environments and/or that are implemented with different technologies for example. It is not comprised in the scope of the development of the ISOFIN Platform functionalities the development of SBS. The ISOFIN Platform functionalities only regard mechanisms of cataloguing and externalizing SBSs allowing the development of IBSs. An SBS never takes the initiative of beginning interactions with IBSs. The interactions are always initiated by IBSs.

- IBS (Interconnected Business Service): A set of functionalities that are exposed from the ISOFIN Platform to ISOFIN Customers. An IBS interconnects one or more SBSs and/or IBSs exposing functionalities that relate directly to business needs. IBSs are externally available to ISOFIN Customers applications and

internally available for interconnections with other IBSs or ISOFIN Applications. IBSs execute in the ISOFIN Platform and each IBS has an interface defined according to an interface formal definition or contract. An IBS interface is an application program interface (api) that allows access to the IBS functionalities. IBSs can be regarded as an incomplete final software building block that will be presented to customers. Incomplete because it is not standalone (its execution, by itself does not translates into any business need, it is required another part to perform the necessary transformations to deliver the expected functionalities) and final because is a self-contained software artifact able to be used, as it is, in conjunction with other artifacts (IBSs, ISOFIN Applications, Remote Business Programs).

- ISOFIN Application: A software application that is built as a result of joining an interface to a single IBS. Interfaces in this context are graphical user interfaces, executed in the scope of the ISOFIN Platform and that are externally exposed to Business Users. Access to ISOFIN Applications is usually done using a secure session in a web browser. ISOFIN Applications are developed to fulfill the need (of the ISOFIN Customer) of accessing, using a simple interface, functionalities exposed by IBSs. That need derives from business needs or legal requirements. By adding graphical user interfaces to IBSs, the functionalities of the ISOFIN Platform can be accessed by a larger group without requiring specific implementation of programs (Remote Business Programs).

- ISOFIN Platform: Software system developed whose main purpose is to respond to ISOFIN Customer's requests through orchestration of a set of integrated services (IBSs and ISOFIN Applications) concerning the financial domain. The ISOFIN Platform encompasses all the tools, services and catalogs required to externalize SBSs (and related information) from the ISOFIN Customers infrastructure and operationalize (assuring) execution of interconnected functionalities in IBSs and ISOFIN Applications. The platform functionalities also include self-management, security and auditing.

An ISOFIN Supplier encompasses all the entities that supply the ISOFIN Platform with SBSs. An entity can be, for example, a Bank, an Insurer, or a Broker. Within the context of the ISOFIN project, these companies can become ISOFIN Suppliers after successful subscription of the ISOFIN Platform.

ISOFIN Customers are those entities whose domain of interactions resides in the scope of consuming the functionalities exposed by the IBSs or ISOFIN Applications.

The ISOFIN project execution was in the context of a consortium. The ISOFIN Consortium is an association of business and academic entities with the objective of promoting the interoperability of applications in the financial domain. The goal of the consortium is to develop the ISOFIN Platform by creating the conditions for future commercial use. The consortium encompasses the following entities: I2S Informática Software e Serviços (designated as the consortium leader), Universidade do Minho, Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa, CCG Centro de Computação Gráfica, INOV INESC Inovação, iZone Knowlege Systems, Maisis Information Systems and KnowledgeBiz Consulting.

The ISOFIN Platform enables information systems integration across multiple domains of interest. In this context, we define “information” as data that is processed and used to make decisions, take actions, and provide better understanding on a subject allowing uncertainty to be reduced. A “system” is a group of multiple components or subsystems that act together in order to accomplish a common purpose. A system is called a subsystem when it is understood as a component of a larger system. On the other hand, a subsystem is considered a system when it is the focus of attention. An “information system” is the set of procedures by which data are collected and processed into information, and then distributed to end users. Due to the previously exposed, the ISOFIN Platform can be regarded as a software-based system that allows information subsystems to interact together.

The typical high-level interactions between all the entities addressed in the ISOFIN project are presented in Figure 30. This representation is one of the results of the interpretation of the final output of the process described in this thesis.

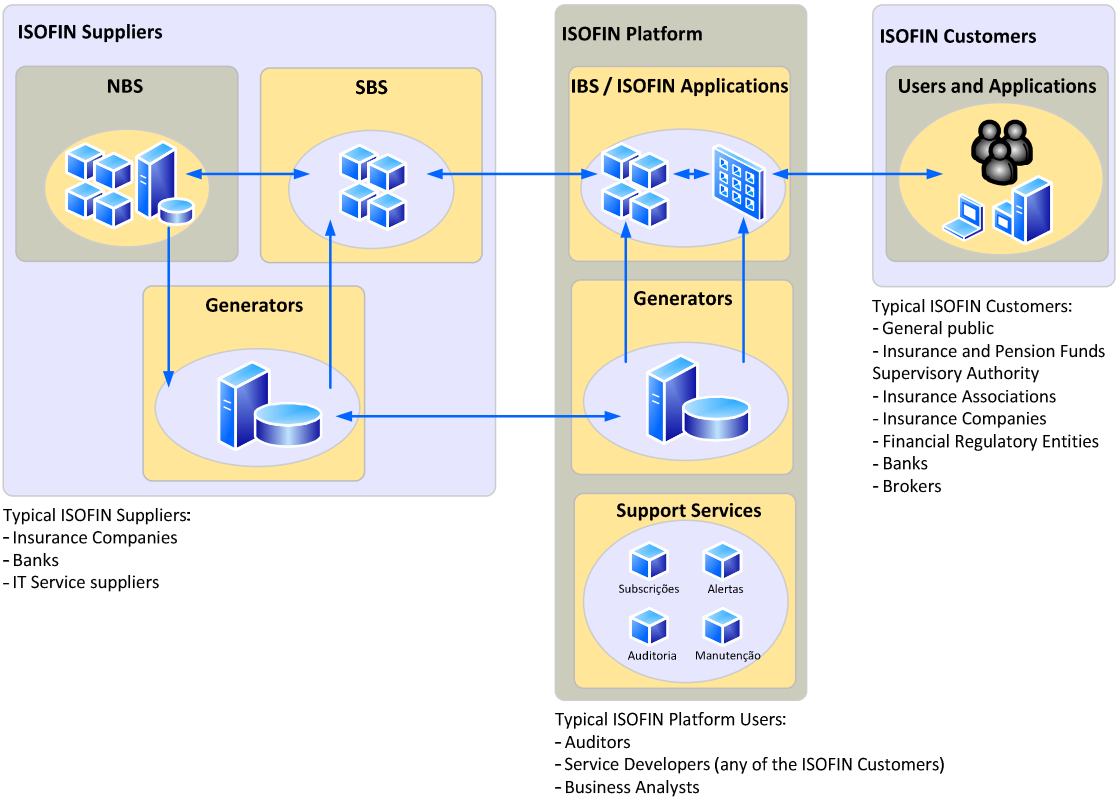

Figure 30: ISOFIN High-level Interactions

In what concerns the scientific research conducted, we frame each of the ISOFIN constructors in a development macro-process as depicted in Figure 31. Based on the business needs, it is made the analysis of the SBS that drives the design and then the implementation. From the SBS emerges a set of IBS arranged as an orchestration of one of more SBS. IBS also derive from the business needs and by its turn, give origin to ISOFIN Applications. Each of the major constructors, SBS, IBS and ISOFIN Applications analysis, design and implementation are made in succession, evolving over time and being functionally dependent from the previous.

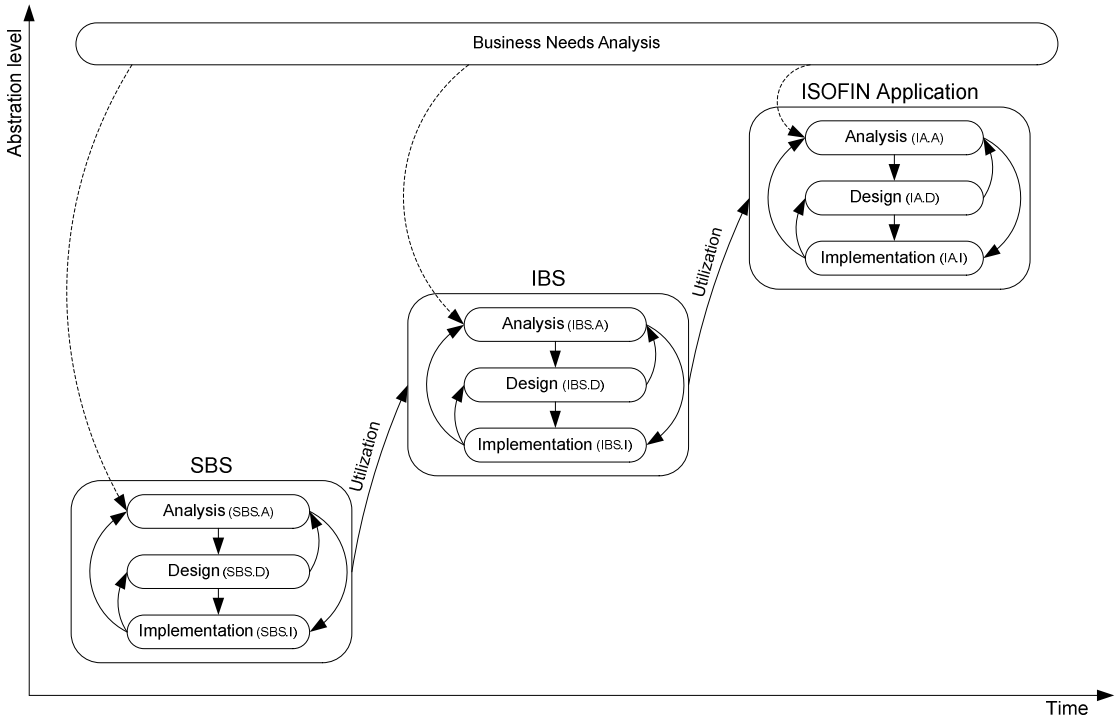

Figure 31: ISOFIN Macro-process

4.3 The design of software architectures

The presented approach is based on a premise that the process-level 4SRS method can be used when there is no agreed on or defined context for requirements elicitation. Requirements Elicitation is concerned with where software requirements come from and how they are collected (Abran, Moore, Dupuis, Dupuis, & Tripp, 2001) within the Requirements Engineering area. The objective of a requirements elicitation task is to communicate the needs of users and project sponsors to system developers (Zowghi & Coulin, 2005). A proper requirements elicitation task must encompass an understanding of the organizational environment, through their business processes (Cardoso, Almeida, & Guizzardi, 2009).

An accurate requirements elicitation can be assured through the use of requirements elicitation methodologies, methods or techniques. The Work System Method (Alter, 2002) presents a combined static view of the current (or proposed) system and a

dynamic view of the system evolution over time. The Soft Systems Methodology (SSM) (Checkland, 2000) is a domain-independent analysis methodology designed for tackling problematic situations where there is neither clear problem definition nor solution.

Our approach suggests the derivation of a process-level logical architecture for creating context for cloud design. Software architecture deals with the design and implementation of the high-level structure of the software (Kruchten, 1995). There are several approaches to support the design of software architectures, in a product-level perspective, like RSEB (Jacobson, Griss, & Jonsson, 1997), FAST (D. M. Weiss & Lai, 1999), FORM (Kang, et al., 1998), KobrA (Bayer, et al., 2001) and QADA (Matinlassi, Niemelä, & Dobrica, 2002). The product-level perspective of the 4SRS (R. J. Machado, et al., 2006b) method also promotes functional decomposition of software systems.

Tropos (Castro, et al., 2002) and 4SRS (in (R. J. Machado & Fernandes, 2002)) are process-level requirement modeling methods. Tropos uses notions of actor, goal and (actor) dependency as a foundation to model early and late requirements, architectural and detailed design. The 4SRS method is usually applied in a product-level perspective. Our presented approach formalizes the process-level perspective that was firstly used in (R. J. Machado & Fernandes, 2002). Use cases act as input for the 4SRS method and, in the 4SRS process-level perspective, portray the activities (processes) executed by persons or machines in the scope of the system, instead of the characteristics (requirements) of the intended products to be developed. According to (Hammer, 1997), and in a business context, a process is executed to achieve a given business goal and where business processes, human resources, raw material, and internal procedures are combined and synchronized towards a common objective. Our processes represent the real-world activities of a software production process, like in (Conradi & Jaccheri, 1999). A software process is composed of a set of activities related to the software development lifecycle. Designing a process comprises the development of a process architecture that continually aggregates

process elements to support tailoring and enhancements of processes. Implementing a process encompasses the specification of the requirements for process execution.

The requirements for process execution can be represented in a logical architecture. A logical architecture can be considered a view of a system composed of a set of problem-specific abstractions supporting functional requirements (Sofia Azevedo, Machado, Muthig, & Ribeiro, 2009). The process architecture represents the fundamental organization of service development, service creation, and service distribution in the relevant enterprise context (Winter & Fischer, 2006b). A process architecture can also be defined as an arrangement of the activities and their interfaces in a process (Browning & Eppinger, 2002), takes into account some non-functional requirements, such as performance and availability (Kruchten, 1995), and can be represented with components, connectors, systems/configurations of components and connectors, ports, roles, representations and rep-maps (Medvidovic & Taylor, 2000), as well as by architectural elements' static and temporal features (Kazman, 1996). The result of the application of the 4SRS method is a logical architecture.

Existing approaches for designing software architecture do not support any specific technique for requirements elicitation; rather, they use the information delivered by an adopted elicitation technique. One problem arises when typical (product-oriented) elicitation techniques cannot properly identify the necessary requirements. With the real industrial case described in this work we demonstrate that firstly adopting process-level techniques allows for better understanding of the project scope since it allows for the elicitation of the activities that will be supported by the product to be developed.

The product-level 4SRS method

A product-level perspective to design can translate system requirements into software architectures and design elements. This is the case where the product-level 4SRS perspective of the method is proven useful (Bragança & Machado, 2009). The

4SRS method in its essence maps UML use case diagrams into UML object diagrams (specialized as architectural elements) resulting in a model of the logical architecture representation of the intended system to be developed (R. J. Machado, et al., 2006b).

The 4SRS method comprises four steps: architectural element creation, architectural element elimination, architectural element packaging & aggregation, and architectural element association. The method takes as input a set of use cases describing the intended system (product) requirements and transforms them into a logical architecture representation of the intended system to be developed. Such is represented in Figure 32. The logical architecture is made of interconnected architectural elements.

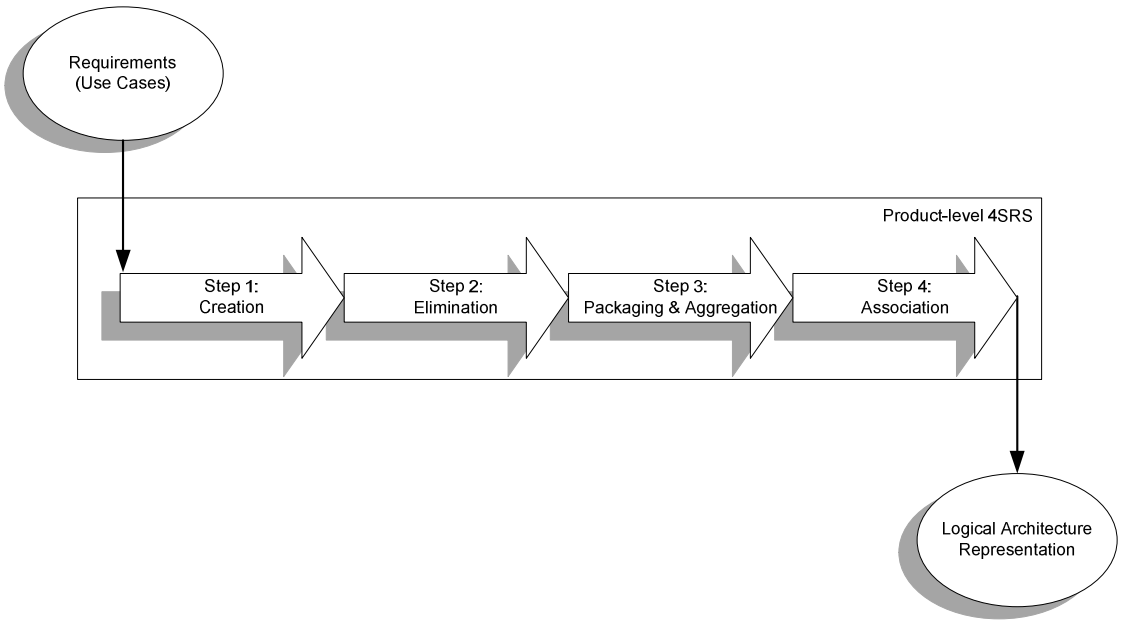

Figure 32: High-level representation of the 4SRS method

Step 1: Architectural Element Creation

The transformation of the use cases into architectural elements is the first of the four steps in 4SRS, the architectural element creation. It is a fully automated step since the transformation is made by creating three architectural element types for each use case. There are three kind of architectural elements: interface (i), data (d) and control (c). Each represents a relation with a design space dimension, namely presentation, information and behaviour respectively.

Step 2: Architectural Element Elimination

The second step, architectural elements elimination, is the most critical of the method application. It comprises seven micro-steps. A full description of the micro-steps can be found in (R. J. Machado, et al., 2005). In order to better understand the method applicability, this critical step, and the changes latter introduced to create the process-level 4SRS method, we present in short, step 2 micro-steps:

- *2i: Use Case Classification*: a classification of each architectural element is made. Each use case is imbued with one or more analysis space dimensions and, when deriving the three "blind" architectural elements in the first step of the 4SRS, in this micro-step there must be made an explicit reference to the architectural element types that are present in the use case description. We must first perform an analysis of the dimensions where the use case currently being processed exists and then, establish a mapping to one or more of the 4SRS architectural element types. The mapping guidance can be seen in Figure 33. An use case may belong to one of the subsets {Ø, i, c, d, ic, di, cd, icd}.

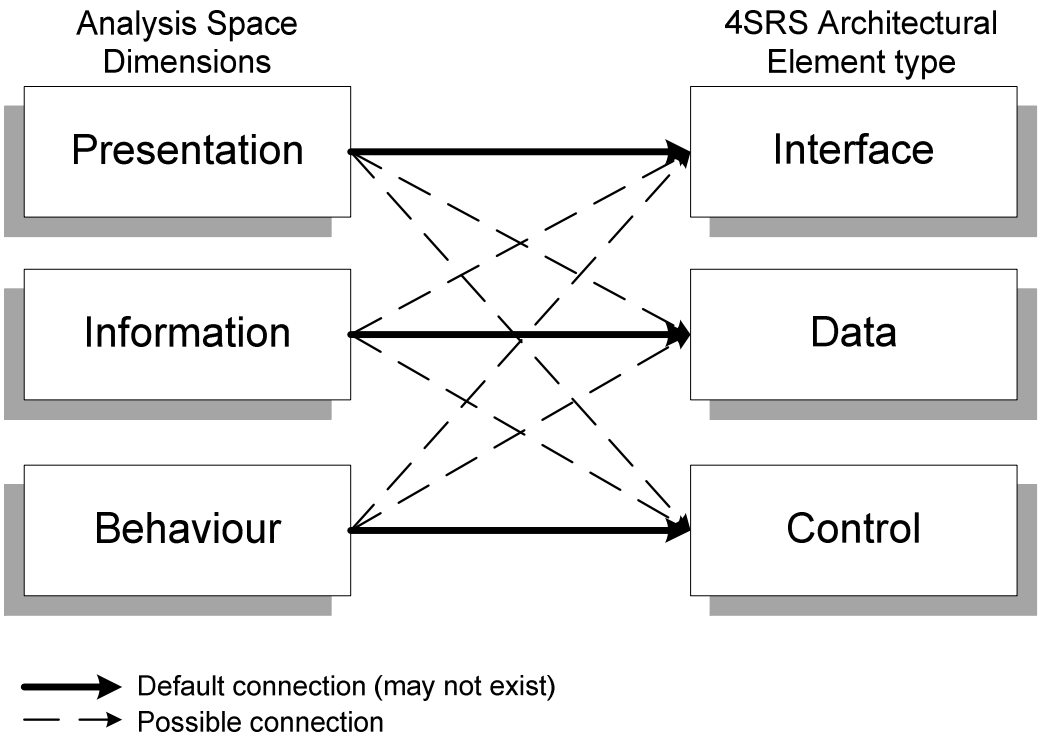

Figure 33: 4SRS Architectural Element and Analysis Space Dimensions mapping

- *2ii: Local Elimination*: elimination of the architectural elements blindly created in step 1 that does not make sense preserving in the use case context.

- *2iii: Architectural Element Naming*: create a name for the architectural elements that remained from elimination in micro-step 2ii.

- *2iv: Architectural Element Description*: provide an accurate description of the architectural element under analysis in order to achieve the most comprehensive detail on the rationale that presided to its creation and prevented it from elimination in the previous micro-steps.

- *2v: Architectural Element Representation*: This micro-step is the most critical in the product-level perspective of the 4SRS method since it encompasses the elimination of redundancy in the architectural element representation while concerning the entire set of elements and not only the derived from a single use-case. This micro-step takes into account if a given architectural element under analysis fully represents or is represented functionally by another architectural element. This micro-step also works as a discovery of hidden requirements.

- *2vi: Global Elimination*: in this micro-step all the architectural elements that are fully represented by others (that is, the system requirements are no longer represented by them) must be eliminated.

- *2vii: Architectural Element Renaming*: since the architectural elements in the previous micro-steps have gained more representativeness it is necessary to rename them to represent the entire requirements of the architectural elements they represent.

Step 3: Architectural Element Packaging & Aggregation

The third step uses the architectural elements that remained from the second step and must be packaged or aggregated in the cases where there is a reason for them to

be treated in a single unit. This unit works as a coherent logical block that allows to group functionalities.

Step 4: Architectural Element Association

The last 4SRS step regards creating associations between the remaining architecture elements. These associations are created by interpreting the initial use cases descriptions and the tasks performed in micro-step 2i.

After the execution of this last step, we achieve a logical architecture representation of the system to be developed. The product-level usage of the 4SRS method can be used recursively (R. J. Machado, et al., 2006b) to allow the refinement of a subset of the logical architectural model. By using the refinement technique, it is possible, by using the method applied to that particular subset, to create a refined and detailed logical architecture model of that particular part of the global system.

When there is not enough information to gather the necessary use cases to act as input for the product-level 4SRS method, it is not possible to assure its proper execution. Since there is not enough information to create a coherent representation of the entire system requirements, micro-steps (like 2v – architectural element representation) do not have necessary information to i) discover if there is any missing requirement and ii) eliminate redundancy.

Looking at some examples of the historical usage of the 4SRS method (the Virtual Automation project (R. J. Machado & Fernandes, 2002), the USE-ME.gov project (R. J. Machado, et al., 2006b), and the ISOFIN project (N. Ferreira, et al., 2012)), it is possible to acknowledge that for creating a proper representation of the system, it is necessary to have an initial representation of, at least, thirty use cases. Lesser representativeness generated a flawed logical architecture model that demanded a new requirements elicitation phase.

If, even though, it is not possible to gather the required use cases with the proper textual descriptions, the product-level perspective is not useful. It is necessary to execute first a process-level 4SRS as described in the following sections to achieve a

proper system requirements representation and, only then, execute the product-level 4SRS method to derive the logical architecture representation of the intended system.

The process-level 4SRS method in the ISOFIN Project

The logical process-level architecture of the ISOFIN solution (ISOFIN Project Consortium, 2010) has embedded design decisions that are initially injected in the processes descriptions. The design decisions concern the deployment of the system in a public cloud environment and its interoperability with several other private clouds as defined in the project objectives.

The resulting logical model of the system architecture, based on the processes that are intended to be executed, shows a software solution able to be deployed in an IaaS layer. That layer will support the execution of a set of services that will allow suppliers to specify the behaviour of the services they intend on supplying, in a PaaS layer. This will allow customers, or third-parties, to use the platform's services, in a SaaS layer and be billed accordingly. This chapter only presents a subset of the proposed process-level architecture related to the customer perspective, as seen in Figure 34. Further details are found in annex A where we present the evolution of the process-level 4SRS iterations and some additional diagrams related to the process-level perspective. Processes regarding the provider perspective (e.g., infrastructure management) are not considered. We present subsets of two use case models concerning two distinctive functionalities provided by the platform.

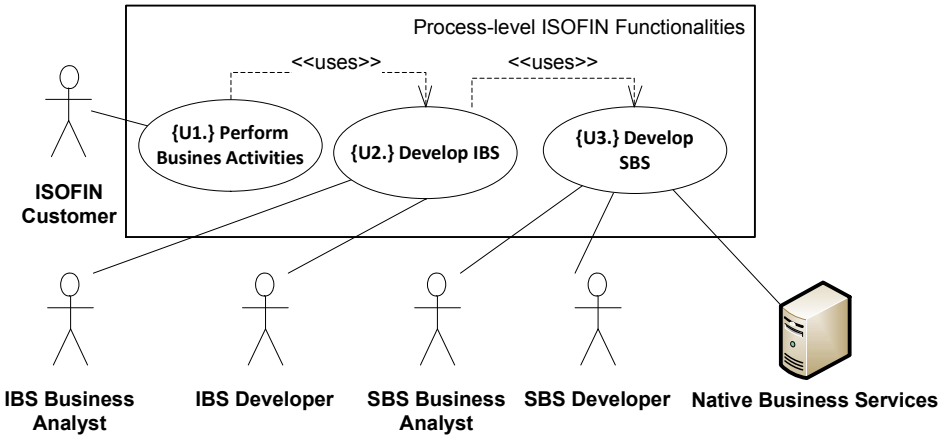

Figure 34 Use Case Model Regarding the ISOFIN Process-level Perspective Functionalities.

The process-level architecture focuses on two sets of functionalities: Interconnected Business Service (IBS) and Supplier Business Service (SBS). IBSs concern a set of functionalities that are exposed from the ISOFIN SaaS Platform to ISOFIN Customers. An IBS interconnects one or more SBSs and/or IBSs exposing functionalities that relate directly to business needs. SBSs are a set of functionalities that are exposed from the ISOFIN Supplier private cloud.

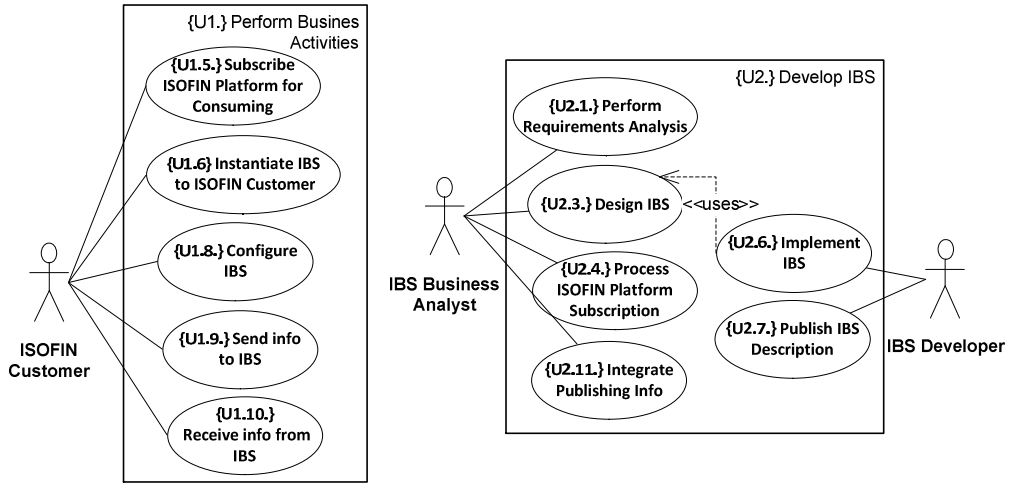

Figure 35 Refinement of Use Case 1 and Use Case 2 (subset).

In Figure 35, there is a description of the execution of a set of economically-related business processes within the context of the project. They are executed through the SaaS layer, since the software components and applications are hosted by third-party

service providers in the cloud. By accessing the services functionalities (represented by implemented IBSs), ISOFIN Customers fulfills their business needs.

Most of these processes, namely the ones regarding the design and implementation efforts, are executed through the PaaS layer. The defined processes will correspond to some of the services and applications that the ISOFIN Platform will support, when executed in the SaaS layer. The model encompasses the analysis, design and implementation of IBSs, accessed externally, through the SaaS layer, and providing ISOFIN Customers with added business value.

4.4 Process-level 4SRS as an Elicitation Method

The 4SRS method allows for the transformation of user requirements into an architectural model representation. This section presents an extension of the traditional (product-level perspective) usage of the 4SRS method (presented in (R. J. Machado, et al., 2006b)) to allow its application in a process-level perspective supporting the creation of context for the product-level requirements elicitation. This application differs from the traditional by defining a set of rules that must be observed when reasoning about the execution of the method steps. Our extension of the method also defines additional micro-steps to the existing ones. Alongside the method presentation there will be included some examples created during the method application to derive a logical architecture that acts as a basis for the requirements elicitation of a cloud SaaS solution, in this case, a subset of the ISOFIN project.

The 4SRS method takes as input a set of use cases describing the requirements for the cloud-specific processes that tackle the initial problem. These use cases are refined through successive 4SRS iterations, representing the intended cloud concerns of the involved business and technological stakeholders. Neither KobrA, RSEB, nor Tropos make use of techniques for refining use cases like the 4SRS method does. Application of the 4SRS method requires the creation of “architectural elements”

(AEs). The nature of AEs varies according to the type of system under study and also with the context where it is applied. In the specific context of logical architectures, the term architectural element refers to the pieces from which the final logical architecture can be built. We deliberately use this term to distinguish those artifacts from the components, objects or modules used in other well established contexts, like in the UML structure diagrams.

The execution of the 4SRS transformation steps can be supported in tabular representations as it can be seen in (R. J. Machado, et al., 2006b). Moreover, the usage of tables permits a set of tools to be devised and built, so that the transformations can be partially automated. These tabular representations constitute the main mechanism to automate a set of decision-assisted model transformation steps. Tabular transformations are supported in a table where the cells are filled with the set of decisions that were taken and made possible the derivation of a logical architecture for the cloud design. Each column of the table concerns a step/micro-step of the method execution. For readability purpose, the entire table was divided into five smaller tables (Tables 2 to 6). In the real context, we manipulate the entire table (seen on Figure 13: Tabular Transformation of the 4SRS Method) and not the smaller ones. The next sub-sections detail the extensions made to the process-level perspective of the 4SRS method and the added micro-steps (product-level 4SRS original steps are in (R. J. Machado, et al., 2006b)).

Step 1: Architectural Element Creation

This step regards the creation of AEs. The product-level 4SRS (R. J. Machado, et al., 2006b) rule of transforming each use case into three AEs is still valid in the process-level 4SRS. According to the MVC-like pattern applied in the product-level 4SRS, an interface, data and control AEs are created for each use case. *i-type*, *d-type*, or *c-type* stereotypes respectively are added to each AE and their names are prefixed with "AE" (the stereotypes definition will be detailed in micro-step 2i). No particular rationale or decision is required at this step since it concerns mainly the transformation of one

use case into three specific AEs. This step is represented in the 1st and 2nd columns of Table 2.

An addition to this step is the identification of glue elements resulting from the textual descriptions associated with the use case under analysis. If the use case depicts pre- or post-conditions in the form of validations, those can be expressed in this step as a *Glue AE*. These AEs have the *c-type* stereotypes since they require decisions to be made with computational support, that is, they must be supported by the system architecture to be represented. A sequential number is added to each Glue AE. Those elements will be used as generic process interfaces between generated AEs and act as pre- or post-condition process validations. Other AEs are expressed as *Generated AE*.

For example, {AE1.9.c2} *Validate Business User* was created as a result of the analysis of the use case {U1.9.} *Send info to IBS* with the description "*[…] Before sending commands to an IBS, ISOFIN Customers must subscribe […]".*

Table 2. Step 1 of the 4SRS method

Step 1 -architectural element creation	
Use Case	Description
{U1.9.}	Send info to IBS
{AE1.9.c2}	Glue AE
{AE1.9.i}	Generated AE

Step 2: Architectural Element Elimination

In this step, AEs are submitted to elimination tasks according to pre-defined rules. At this moment, the system architect decides which of the original three AEs (i, c, d) plus any glue element are maintained or eliminated taking into account the entire system.

The original step 2 of 4SRS is divided into seven micro-steps. We added a new micro-step, 2viii: Architectural Element Specification. With this addition, step 2 becomes more robust and detailed. It provides information to the next steps that was hard to obtain in the original version.

Micro-step 2i: Use Case Classification

In this step, each use case is classified according to the nature of its AEs, previously created in step 1. The nature of an AE is defined according to the suffix the AE was tagged with. This classification is represented in the 2nd column of Table 3 (the 1st column regards the AE identification). In the process-level perspective more than one of each AE type can be generated according to the textual description and in the model of the use case. Each AE type must be interpreted as follows:

- *i-type* – refer to interface. These represent process' interfaces with users, software or other processes. An AE belonging to or being classified in this category is due to its ability interact with other AEs external to itself;
- *c-type* – refer to control. These represent a process focusing on decision making and such decision must have a computational support given from the overall intended system;
- *d-type* – refer to generic decision repositories (data), not computationally supported from the overall intended system. This repository stores information for a given period of time, regardless of duration, comprising decisions based on physical repositories (like documents or databases) or verbal decisions taken and transmitted between humans.

In the process-level perspective, *c-type* and *d-type* AEs are related to decision-making processes. The difference resides on the computational support of the AE by then under design overall intended system (in hypotheses).

Micro-step 2ii: Local Elimination

This micro-step refers to determining which AEs must be eliminated in the context of a use case, guaranteeing its full representation. This is required since micro-step 2i disregards any representativeness concerns.

There are cases when there is an explicit place for a *d-type* AE and it is admittedly eliminated. Reasons for this are due to the process-level perspective: there is no need for certain types of decision repositories that only regard information for the final

product and not the process. This is the case, for example, in use case {U1.9.} *Send info to IBS*, where any possible repository (data object in the traditional 4SRS) that could exist would only reflect the product-level perspective and not the process. Other situation similar to the previous one is when a given *d-type* AE exists in the product-level perspective but also, and above it, exists in the process-level perspective. This is the case of {U1.6} *Instantiate IBS to Remote Business Program*, where {AE1.6.d} *IBS Configuration Decisions* represents the process for supporting the configuration process (process-level), not the configuration repository (product-level).

The 3rd column in Table 3 corresponds to the execution of micro-step 2ii. The cells are filled with “T” or “F”. “T” means the AE is going to be eliminated and “F” that the AE is kept alive.

Micro-step 2iii: Architectural Element Naming

In this micro-step (4th column of Table 3), AEs that survived the previous micro-step are given a name. The name must reflect the role of the AE within the entire use case, in order to semantically give hints on what it represents and not just copy the original use case name. Usually, the AE name reflects also the use case from which the AE was originated.

For better understanding of the role of the AE, it is advisable that the name given reflects the type (c, d or i) of the AE. For instance, since *d-type* refers to decision-making, in our model, we decided to name “IBS Configuration Decisions” to {AE1.6.d}. In glue AE cases, the naming of the AE should reflect the pre- or post-conditions that are executed. For instance, {AE2.4.3.d} *ISOFIN Platform Supplier Policy*, reflects the pre-condition “*The ISOFIN Supplier must accept […] to comply with the defined policy*”.

Table 3. Micro-steps 2i through 2iv of the 4SRS method

	Step 2 - architectural element elimination			
	2i - use case classification	**2ii - local elimination**	**2iii - architectural element naming**	**2iv - architectural element description**
{U1.9.}	i			
{AE1.9.c2}		F	Validate Remote Business Program	Execute the necessary verification procedures to ensure that the Remote Business Program is …
{AE1.9.i}		F	Send Commands to IBS	Send commands and associated information to the IBS in order to process a business request…

Micro-step 2iv: Architectural Element Description

This micro-step is represented in the 5th column of Table 3. The resulting AEs that were named in the previous micro-step must be described and the requirements that they represent must be addressed in the process-level perspective. This micro-step is where the transition is made from the problem domain to the solution domain, so the descriptions must detail, in process terms, how, why, when by whom that AE is going to be executed. This micro-step must explicitly describe the expected behavior of the AE execution, including which decisions will be made and how will they be supported.

Micro-step 2v: Architectural Element Representation

The purpose of this micro-step is to eliminate AE redundancy in the global process. In this micro-step, all AEs are considered and compared in order to identify if one AE is represented by any other one. The identification of AE representation is the most critical task in the 4SRS method application, because the elimination of redundancy assures a semantic coherence of the logical architecture and discovers anomalies in the use case model. Since the architecture being described concerns the process-level, the identification of AE redundancy takes in consideration facts like the execution context, actors involved, used artifacts, activities and tasks, among others. If all of these factors are similar, though the AEs are originated by different use cases,

the given AE can be considered to represent another. Other cases when an AE is considered to represent another:

In similar activities, if the same actor has the same role in the both AEs, despite different execution contexts (e.g., {AE2.4.1.i} *Perform ISOFIN Supplier Request Evaluation* is considered to be represented by {AE2.4.2.i} *Perform ISOFIN Customer Request Evaluation*, the IBS Business Analyst triggers both AEs – the first AE represents the second AE, because the actor interacts with the same type of information);

In similar activities, different actors participate in the AE, but the execution context is the same (e.g., {AE2.1.c} *Access Remote Catalogs* and {AE1.11.i} *Browse ISOFIN Catalogs*, the involved actors are different, but the execution platform is the same – both of them execute in the ISOFIN Platform, in the SaaS layer).

These cases are only applicable for *i-type* and *c-type* AEs. This set of rules cannot be applied to *d-type* AEs since they represent the decisions that need to be taken and whose computational support is not assured by the scope of the project under analysis. Also, *d-type* AEs are usually input for other decision processes (*c-type* AEs) requiring computational support.

Despite the decision making process may be similar, *d-type* AEs differ in the decision making purpose. This difference is required to assure the process variability, when the execution contexts are similar but the involved actors and activities are different. For example, {AE1.5.d} *Consumer Subscription Requirements* and {AE3.3.d} *SBS Catalog Subscription Requirements* cannot be represented by one AE, although the *i-type* related AEs – {AE1.5.i} and {AE3.3.i} – are represented by the same AE. The decision making regarding a specific purpose viewed from different perspectives concerns different purposes, even if, at first sight, the interface seems to be the same.

A potential concern when executing this micro-step regards the number of AEs involved. Since all living AEs must be accounted in the analysis, it is hard to keep track of all the processes they refer to in order to know if one can be represented by other.

In the product-level perspective, this step concerns the analysis if a given AE is complex enough to exist by itself or if there is any other AE whose functionalities can be incorporated in the one under analysis. This rule also applies to the process-level perspective, if three questions are considered:

- Is the analyzed AE suitable to be represented by other in his entire functionality?
- Is the target AE suitable to incorporate the AE under analysis functionalities without losing any of its own characteristics?
- If the target AE is complex and the extra-functionalities to be added increase the complexity will it be in a degree where its maintenance, description or scope are compromised?

If the activities or processes executed within the context of a given AE are to be executed by another AE and the target AE is subject to change, no extra complexity should be added to that target AE nor its core specification change in order to full represent the source AE.

The execution of micro-step 2v is presented in Table 4 in the 2nd and 3rd columns. The 2nd column, “represented by”, stores the reference of the AE that will represent the AE being analyzed. If the analyzed AE is going to be represented by itself, the corresponding “represented by” column must refer to itself. The 3rd column, “represent”, stores the references of the objects that the analyzed AE will represent.

Micro-step 2vi: Global Elimination

This micro-step (4th column in Table 4) refers to determining which AEs must be eliminated in the context of the global model, similar to micro-step 2ii, since its execution is automatic.

The AE that is represented by itself or represents other AEs is maintained. The rest (i.e., AEs that are represented by other AEs) are eliminated. This is a fully “automatic” micro-step, since it is based on the results of the previous one. If the AE is

represented by itself, cell is filled with “T”, meaning that the AE is represented by other AE and thus, eliminated, and “F” if the AE is going to be kept alive.

Micro-step 2vii: Architectural Element Renaming

In this micro-step (5th column in Table 4), AEs that have not been eliminated in micro-step 2vi are renamed. In cases where the AE under analysis results of the representation of more than one AE, the new name must reflect the global execution of the AE in the project context.

Micro-step 2viii: Architectural Element Specification

This micro-step (6th column in Table 4) has never been considered in previous versions of the traditional 4SRS method. Though it is similar to micro-step 2iv, this micro-step intends to describe AEs that, in micro-step 2v, are considered to represent other AEs. The decision of creating this micro-step arises from the need to clearly define the proper behavior of the “new” AE in a way that is clear to system architects. Besides including the information regarding AEs eliminated in micro-step 2vi as a result of micro-step 2v, the AEs specifications must include the pre-conditions of the basic AEs, so it can properly support the associations to be established in step 4. For instance, if the extended description of {AE1.9.c1} does not include the conditions described in {AE1.1.c1}, that information would be lost since {AE1.1.c1} has been eliminated in micro-step 2vi and, as such, is not considered in step 4. If those references are not preserved in any surviving AEs, they will be permanently lost and thus, disregarded in the construction of the logical diagram model.

The specification must also include execution sequence references of the AEs. For instance, {AE2.9.i} must reference the ISOFIN Application catalog described by {AE1.3.d}, which is also eliminated in micro-step 2v, to create the association in step 4. The specification information is required in the transformation from the process-level perspective to the product-level perspective, to infer the necessary

requirements of a given product based on the processes of which the product is composed.

This micro-step contributes to a better description of AEs that result from joining other AEs. By adding this information, the designer can clearly express their thoughts and decisions concerning the creation of the AE under analysis as a result of the potentially added extra-complexity resulting from micro-step 2v.

Table 4. Micro-steps 2v through 2viii of the 4SRS method

	Step 2 - architectural element elimination				
	2v - architectural element representation		2vi - global elimination	2vii - architectural element renaming	2viii - architectural element specification
	represented by	represent			
{U1.9.}					
{AE1.9.c2}	{AE1.9.c2}	{AE1.1.c2}	F	Validate Platform Access	Execute the necessary verification procedures to ensure that subscribed ISOFIN Customers…
{AE1.9.i}	{AE1.9.i}		F	Send Commands to IBS	

It is necessary to pay a special attention to the AEs that represent other AEs in micro-step 2v. The specification must clarify system architects in what way the AE is executed and how its execution represents an eliminated AE.

Step 3: Packaging and Aggregation

Like in the traditional 4SRS method, in this step (2nd column in Table 5), the remaining AEs (those that were maintained after executing step 2), for which there is an advantage in being treated in a unified process, should give the origin to aggregations or packages of semantically consistent AEs. This step supports the construction of a truly coherent process-level model.

In order to correctly package AEs, it is necessary to consider the model as a whole, so that all relevant processes (in a high-level order of abstraction) are identified. Then, when justifiable, the AEs are associated to a package. The packaging technique contributes for a temporary obtainment of a more comprehensive and understandable process model. Typically, aggregation is used when there is a part of the process that constitutes a legacy sub-system, or when the design has a pre-defined reference architecture that constricts the model.

Table 5. Step 3 of the 4SRS method

	Step 3 - packaging & aggregation
{U1.9.}	
{AE1.9.c2}	{P6} ISOFIN Platform Management
{AE1.9.i}	{P2.4} IBS

Step 4: Architectural Element Association

Decisions on the identification of associations between AEs can be based in information contained in the use case model and in micro-step 2i. Thus, step 4 was divided in two micro-steps: micro-step 4i: Direct Associations and 4ii: Use Case Associations. It is also important to point out that any textual references to eliminated AEs in micro-step 2vi, must be included in micro-step 2viii, making it another source of information for step 4.

In the traditional 4SRS application, this step is executed in a single step. We propose to do it in two micro-steps in order to ease identification of unnecessary direct associations, as well as associations originated by textual description of eliminated AEs. This division, by separating the associations by its source, also helps to adjust the model when there are changes due to refinements or corrections in the previous steps execution.

Micro-step 4i: Direct Associations

Direct associations (2nd column of Table 6) are the ones that derive from AEs originated by the same use case. These associations are depicted from the classification given in the method micro-step 2i. For example, {AE1.6.d} *IBS Configuration Decisions* and {AE1.6.i} *Configure pre-runtime IBS* are directly

associated since they are originated by the same use case, {U1.6} *Instantiate IBS to Remote Business Program*.

Micro-step 4ii: Use Case Model Associations

Use Case Model Associations are the ones that can be inferred from the textual descriptions of use cases, that is, when a use case description refers, implicitly or explicitly to another use case, the associations inferred imply that the use cases are connected. This micro-step is represented in the 3rd column of Table 6.

Table 6. Step 4 of the 4SRS method

	Step 4 - architectural element association	
	4i - Direct Associations	**4ii - UC Model Associations**
{U1.9.}		
{AE1.9.c2}	{AE1.1.i}, {AE1.9.c1}, {AE1.9.i}.	{AE3.3.i}.
{AE1.9.i}	{AE1.9.c1}, {AE1.9.c2}.	{AE1.7.i}, {AE2.9.i}, {AE3.3.i}.

As an example for these situations, the use case textual description of {U3.7.1.} *Publish in Platform Catalog* in the use case model refers that "*The SBS […] is available for access to IBS Business Analyst (see use case {U2.2.} Choose SBS Specs, use case {U2.3.1.} Define IBS Internal Structure and use case {U2.5.} Choose SBS Implementation) and to the SBS Developer (see use case {U2.6.} Implement IBS)*". Thus, the generated surviving AE – {AE3.7.1.i} *Remote SBS Publishing Interface* – is associated with {AE2.1.c}, {AE2.3.1.c}, and {AE2.6.1.i}.

The ISOFIN Process-level Logical Architecture

The initial request for the ISOFIN project requirements resulted in mixed and confusing sets of misaligned information. Even when a requirement found a consensus in the consortium, the intended behavior or definition was not easily understood by all the stakeholders. Our proposal of adopting a process-level perspective was agreed on and, after being executed, resulted in a set of information that the consortium sustainably used to evolve to the traditional (product-level) development scenario. Elicited requirements in a process-level perspective describe

the processes in a higher level of abstraction, making them understandable by business stakeholders. At the same time, definitions and intended behavior of the system, expressed in the architecture that results from the process-level 4SRS method, describe the system to technological stakeholders.

The turning point for eliciting requirements was the usage of the 4SRS method in the process-level perspective, which allowed the transformation of process-level requirements into the logical diagram. Due to the diagram's complexity, we only present a subset in Figure 36. This diagram represents the logical architecture of the process-level ISOFIN functionalities. The architecture is composed by the AEs that survived after the execution of step 2. The packaging executed in step 3 allows the identification of major processes. The associations identified in step 4 are represented in the diagram by the connections between the AEs (for readability purposes, the "direct associations" were represented in dashed lines, and the "use case model associations" in straight lines).

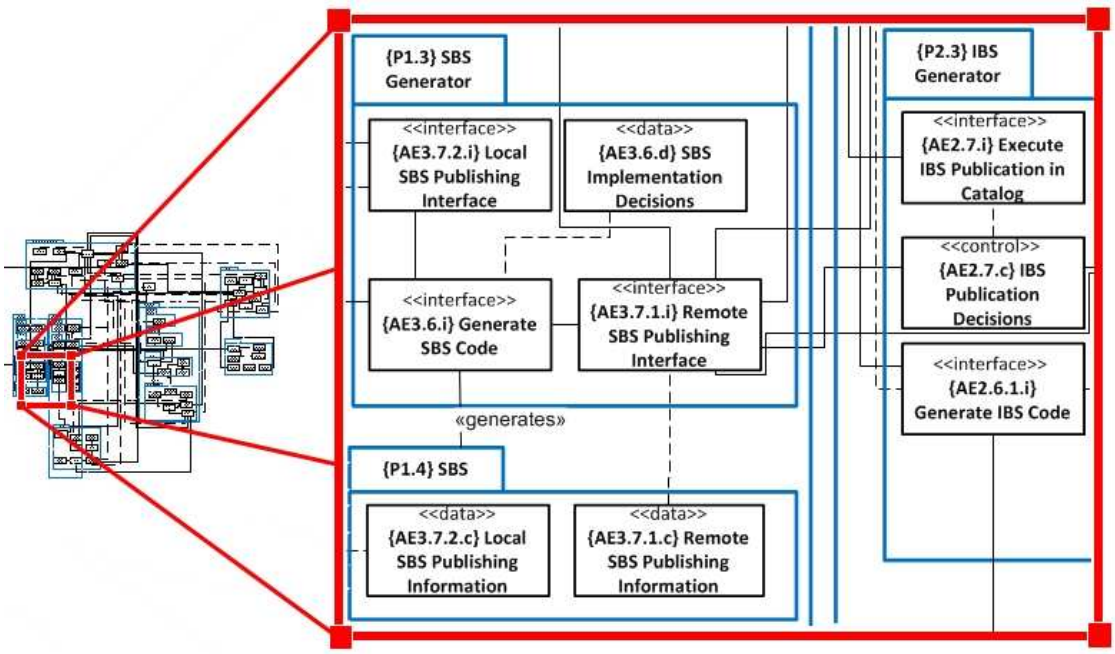

Figure 36: Subset of the process-level logical architecture

As seen previously, the process-level architecture focuses on IBS and SBSs, acting as services in the cloud environment and allowing interoperability between the insurance domain business entities. In this context, there are two external business

domain entities with access to the ISOFIN Platform: ISOFIN Customers and ISOFIN Suppliers. An ISOFIN Customer is an entity whose domain of interactions resides in the scope of consuming, for economic reasons, the functionalities exposed by IBSs. An ISOFIN Supplier is a company that interacts with the ISOFIN SaaS Platform by supplying the platform with functionalities (SBSs) that reside in their private clouds.

SBSs are made available in the ISOFIN Supplier private cloud by the use of generators ({AE3.6.i} Generate SBS Code) and are composed, in the public cloud where the ISOFIN SaaS Platform resides ({AE2.6.1.i} Generate IBS Code) to implement an IBS. Composition of basic SBSs into IBSs give origin to more powerful functionalities that are exposed by the platform.

Due to the lack of consensus in the requirements elicitation in this “newfound” paradigm of IT solutions (Cloud Computing), our approach changed the traditional product-level perspective to the described process-level perspective. This new perspective allows the proper elicitation of requirements in Cloud Computing projects.

4.5 Conclusion

This chapter started by presenting the ISOFIN project as the context for the problem we had to tackle. Following that contextualization, we introduced both perspectives on the 4SRS method: the product- and process-level. We also detailed the extensions made to the traditional application of the 4SRS method, for creating context for requirements elicitation and later derivation of logical architectural diagrams from use cases in a process-level perspective.

By using the proposed approach, we succeeded to define the requirements in such a way that the requirements were understood by all the project stakeholders, uncovering more information: as an example, we started with 39 use cases and ended with 74 documented AEs (not counting associations). This means that we

added more details to the problem description and that all the involved project stakeholders understand the information.

On the other hand, the manual execution of the method is prone to errors and very time consuming. In addition, by adopting first the process-level perspective instead of the product-level perspective, time for delivering documentation to implementation teams increased. The process-level perspective allowed us to overcome difficulties when adopting a product-level perspective and the time (cost) dedicated to producing such level of details was repaid the development process and the overall improvement of understandability for all the stakeholders.

The ISOFIN project aims to deliver a set of functionalities that help forward interoperability in the Insurance application domain. The obtained process-level logical architecture is mainly devoted to be used by IT-professionals and not by business stakeholders. Based on the main constructors presented in the architecture Figure 36, the diagram represented in Figure 29 and in Figure 30 emerged with the aim to be presented to any technical role engaged in the ISOFIN project and be used to explain in a simple way that in the bottom layer there are SBSs that connect to IBSs in the ISOFIN Platform layer and that the later are connected to a ISOFIN Customer role.

Chapter 5

Process- and Product-level Logical Architectures

Chapter Contents

5

Process- and Product-level Logical Architectures

The hardest single part of building a software system is deciding precisely what to build.

Frederick Brooks

This chapter presents an approach that supports the creation of a service-oriented logical architecture, beginning in a process-level perspective and evolving to a product-level perspective through successive derivation of models with the purpose of creating context for the implementation teams. The requirements are expressed through models, namely logical architectural models and stereotyped sequence diagrams. We define a V+V process approach, based on V-Models, that defines the flow of model derivation in both a process-level and in a product-level perspective.

5.1 Introduction

A typical business software development project is coordinated so that the resulting product properly aligns with the business model intended by the leading stakeholders. The business model normally allows for eliciting the requirements by providing the

product's required needs. In situations where organizations focused on software development are not capable of properly eliciting requirements for the software product, due to insufficient stakeholder inputs or some uncertainty in defining a proper business model, a process-level requirements elicitation is an alternative approach.

The process-level requirements assure that organization's business needs are fulfilled. However, it is necessary to assure that product-level (IT-related) requirements are properly aligned with process-level requirements, and hence, are aligned with the organization's business requirements.

Using a process-level perspective, in order to create an information system logical architecture that is used for eliciting service-based software (product-level) requirements, is a possible approach. Services in Cloud Computing environments have earned much attention because, amongst other aspects, they enable interoperability and rapid development of large scale distributed applications in various application domains (Chen & Tsai, 2010). Composing such services in a more powerful service brings more functionality to the system (Yipeng, Hailong, Xudong, Jin, & Shangda, 2009). The strategy of composing services results in a straightforward development process for cloud applications.

The first effort should be to specify the requirements of the overall system in the physical world; then to determine necessary assumptions about components of that physical world; and only then to derive a specification of the computational part of the control system (Maibaum, 2006). There are similar approaches that tackle the problem of aligning domain specific needs with software solutions. For instance, goal-oriented approaches are a way of doing so, but they do not encompass methods for deriving a logical representation of the intended system processes with the purpose of creating context for eliciting product-level requirements.

Our main problem is assuring that product-level (IT-related) requirements are perfectly aligned with process-level requirements, and hence, are aligned with the organization's business requirements. The process-level requirements express the need for fulfilling the organization's business needs, and we detail how they are

characterized within our approach further in section 2. These requirements may be supported by analysis models, that are implementation agnostic (Yue, et al., 2011). According to (Yue, et al., 2011), the existing approaches for transforming requirements into an analysis model (i) do not require acceptable user effort to document requirements, (ii) are efficient enough (*e.g.*, one or two transformation steps), and (iii) are able to (semi-)automatically generate a complete (*i.e.*, static and dynamic aspects) consistent analysis model, which is expected to model both the structure and behavior of the system at a logical level of abstraction.

One of the possible representations of an information system is its logical architecture (Castro, et al., 2002), resulting from a process of transforming business-level and technological-level decisions and requirements into a representation (model). This representation is fundamental and mandatory to analyze and validate a system but is not enough for achieving a full transformation of the requirements into a model able to implement stakeholders' decisions. It is necessary to promote an alignment between the logical architecture and other supporting models, like organizational configurations, products, processes, or behaviors.

A logical architecture can be considered a view of a system composed of a set of problem-specific abstractions supporting functional requirements (Sofia Azevedo, et al., 2009). A process architecture can be defined as an arrangement of the activities and their interfaces in a process (Browning & Eppinger, 2002), that takes into account some non-functional requirements, such as performance and availability (Kruchten, 1995), and that can be represented with components, connectors, systems/configurations of components and connectors, as well as with architectural elements' static and temporal features (Kazman, 1996). The ANSI/IEEE 1471-2000 Recommended Practice for Architectural Description of Software Intensive Systems defines architecture as the "*fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution*" (IEEE Computer Society, 2000).

In order to properly support technological requirements that comply with the organization's business requirements, in this chapter, we present an approach

composed of two V-Models (Haskins & Forsberg, 2011), i.e., the V+V process. The requirements are expressed through logical architectural models and stereotyped sequence diagrams (R. Machado, et al., 2007) in both a process- and a product-level perspective. The first execution of the V-Model regards eliciting requirements from a high-level business level to create context for product design (*CPD*). The second execution of the V-Model regards a product-level perspective and outputs a context for product implementation (*CPI)*. This approach also assures a proper compliance between the process- and the product-level requirements through a set of transition steps between the two perspectives.

We present an approach framed within a macro-process for information systems development, as presented in Figure 37. The approach encompasses domain analysis, requirements elicitation, modeling and design of logical architectures. Regarding the design, our approach deals, in particular, with the architectural and mechanistic design of the logical architecture. By mechanistic, we mean that not only do regard the general structure but also non-functional requirements, and behavioral mechanisms that are imbued in the representation by means of design decisions that bridge the gap to implementation issues. Each V-Model is self-contained regarding inner-validation for macro-process evolution.

The process-level V-Model acts in the analysis phase, creating the *CPD*. The vertex is assured by the process-level 4SRS method execution (Nuno Ferreira, et al., 2012a). The process-level 4SRS method execution results in the creation of a validated architectural model which allows creating context for the product-level requirements elicitation and in the uncovering of hidden requirements for the intended product design. The product-level V-Model (the second V-Model) enables the transition from analysis to design through the execution of the product-level 4SRS method (R. J. Machado, et al., 2005). The resulting architecture is then considered a design artifact that contributes for the *CPI* as information required by implementation teams.

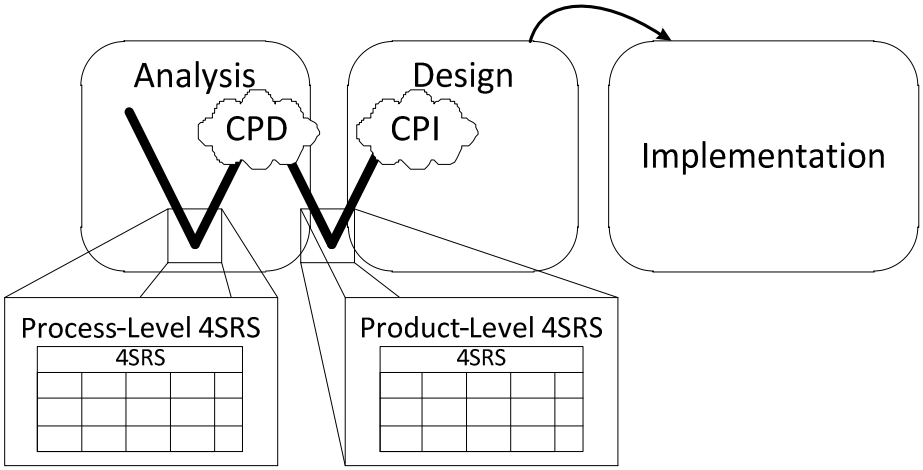

Figure 37: V+V process framed in the development macro-process

This chapter is structured as follows: we begin by presenting the macro-process based on both the process- and product-level V-Models; we follow by describing the transition steps between both perspectives; In the next section we present the applicability of our approach in the context of the ISOFIN project and make and assessment of the transition steps between process- and product-level perspectives; we also include a comparison of our approach with other related works to promote discussion on the subject.

5.2 A Macro-process Approach to Software Design

The development process of information systems can be regarded (in a simple way) as a cascaded lifecycle (*i.e.*, a development process only initiates when the previous has ended), if we consider typical and simplified phases: analysis, design and implementation. Our approach encompasses two V-shaped process models hereafter referred as the V+V process.

The first V-Model (at process-level) is composed by *Organizational Configurations* (OC), *A-type* and *B-type* sequence diagrams, and *Use Case* models (UCs) that are used to derive (and, in the case of *B-type* sequence diagrams, validate) a process-level logical architecture (i.e., the information system logical architecture). We frame the process-level V-Model (the first V-Model of Figure 37) in the analysis phase, creating

the context for product design (*CPD*). In its vertex, the process-level 4SRS method execution assures the transition from the problem to the solution domain by transforming process-level use cases into process-level logical architectural elements, and results in the creation of a validated architectural model which allows creating context for the product-level requirements elicitation and in the uncovering of hidden requirements for the intended product design.

One of the outputs of any of our V-Models is the logical architecture diagram for the intended system. This diagram is considered a design artifact but the design itself is not restricted to that artifact. We have to execute a V+V process to gather enough information in the form of models (logical architecture diagrams, *B-type sequence diagrams* and others) to deliver to the implementation teams the correct specifications for product realization.

Regarding the first V-Model, we refer that it is executed at a process-level perspective. How the term *process* is applied in this approach can lead to inappropriate interpretations. Since the term *process* has different meanings depending on the context, in our process-level approach we acknowledge that: (1) real-world activities of a business software production process are the context for the problem under analysis; (2) in relation to a software model context (Conradi & Jaccheri, 1999), a software process is composed of a set of activities related to software development, maintenance, project management and quality assurance. For scope definition of our work, and according to the previously exposed acknowledgments, we characterize our process-level perspective by: (1) being related to real-world activities (including business); (2) when related to software, those activities encompass the typical software development lifecycle. Our process-level approach is characterized by using refinement (as one kind of functional decomposition) and integration of system models. Activities and their interface in a process can be structured or arranged in a process architecture (Browning & Eppinger, 2002).

Our V-Model approach (inspired by the “Vee” process model (Haskins & Forsberg, 2011)) suggests a roadmap for product design based on business needs elicited in an early analysis phase. The approach requires the identification of business needs and

then, by successive artifact derivation, it is possible to transit from a business-level perspective to an IT-level perspective and at the same time, aligns the requirements with the derived IT artifacts. Additionally, inside the analysis phase, this approach assures the transition from the business needs to the requirements elicitation.

In this section, we present our approach, based on successive and specific artifacts generation. We use *Organizational Configurations* (OC) (Evan, 1965), *A-type* and *B-type sequence diagrams* (R. Machado, et al., 2007), (business) *Use Case models* (UCs) and a process-level logical architecture diagram. The generated artifacts and the alignment between the business needs and the context for product design can be inscribed into the first V-Model (at process-level).

The V-Model representation provides a balanced process representation and, simultaneously, ensures that each step is verified before moving into the next. The artifacts are generated based on the rationale and in the information existing in previously defined artifacts, i.e., A-type diagrams are based on OCs, (business) use case model is based on *A-type sequence diagrams*, the logical architecture is based on the (business) use case model, and *B-type sequence diagrams* comply with the logical architecture. The V-Model also assures validation of artifacts based on previously modeled artifacts (e.g., besides the logical architecture, *B-type sequence diagrams* are validated by *A-type sequence diagrams*). The aim of this section if not to detail the inner execution of the V-Model (that was done in chapter 3 of this thesis), rather it is to explain, justify and exemplify the rules that enable the transition from the process-level V-Model to the product-level V-Model within the macro-process of information systems development.

The presented approach encompasses two V-Models, hereafter referred as the V+V process and depicted in Figure 38. The first V deals with the process-level perspective and its vertex is supported by the process-level 4SRS method detailed in (Nuno Ferreira, et al., 2012a). The purpose of this first execution of the V-Model regards eliciting requirements from a high-level business level to create context for product design (*CPD*), that can be considered a business elicitation method (like the Business Modeling discipline of RUP).

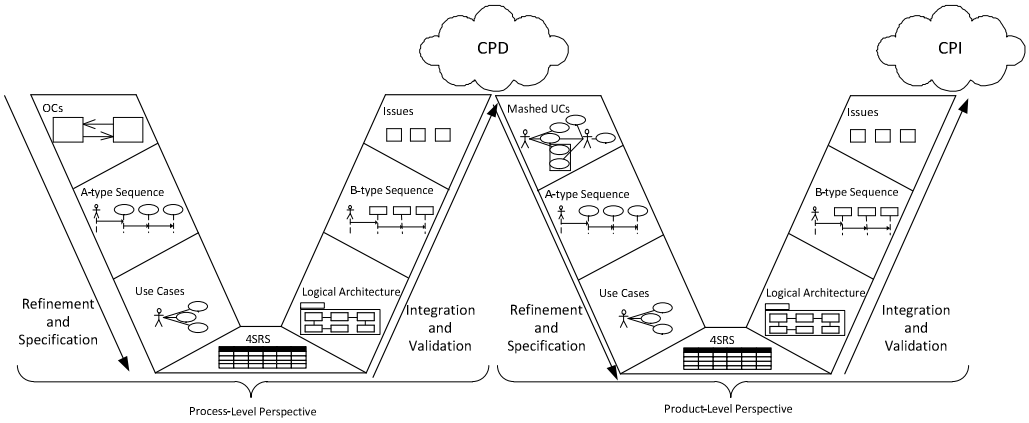

Figure 38: The V+V process approach

The second execution of the V-Model is done at a product-level perspective and its vertex is supported by the product-level 4SRS method detailed in (R. J. Machado, et al., 2005). The product-level V-Model gathers information from the *CPD* in order to create a new model referred as *Mashed UCs*. The creation of this model is detailed in the next section of this chapter as transition steps and rules. Using the information present in the *Mashed UCs* model, we create *A-Type Sequence Diagrams*, detailed in (R. Machado, et al., 2007). These diagrams are input for the creation of (software) *Use Case Models* that have associated textual descriptions of the requirements for the intended system. Using the 4SRS method in the vertex, we derive those requirements into a *Logical Architecture model*. Using a process identical to the one used in the process-level V-Model, we create *B-type sequence diagrams* and assess the *Logical Architecture Model*.

Both V-Models produce *Logical Architecture Models*: the first V produces a process-level logical architecture (that can be considered the information system logical architecture); the second V produces a product-level logical architecture (that can be considered the business software logical architecture). Also, for each of the V-Models, in the descending side of the V (left side), models created in succession represent the refinement of requirements and the creation of system specifications. In the ascending side (right side of the V), models represent the integration of the discovered logical parts and their involvement in a cross-side oriented validating effort contributing for the inner-validation for macro-process evolution.

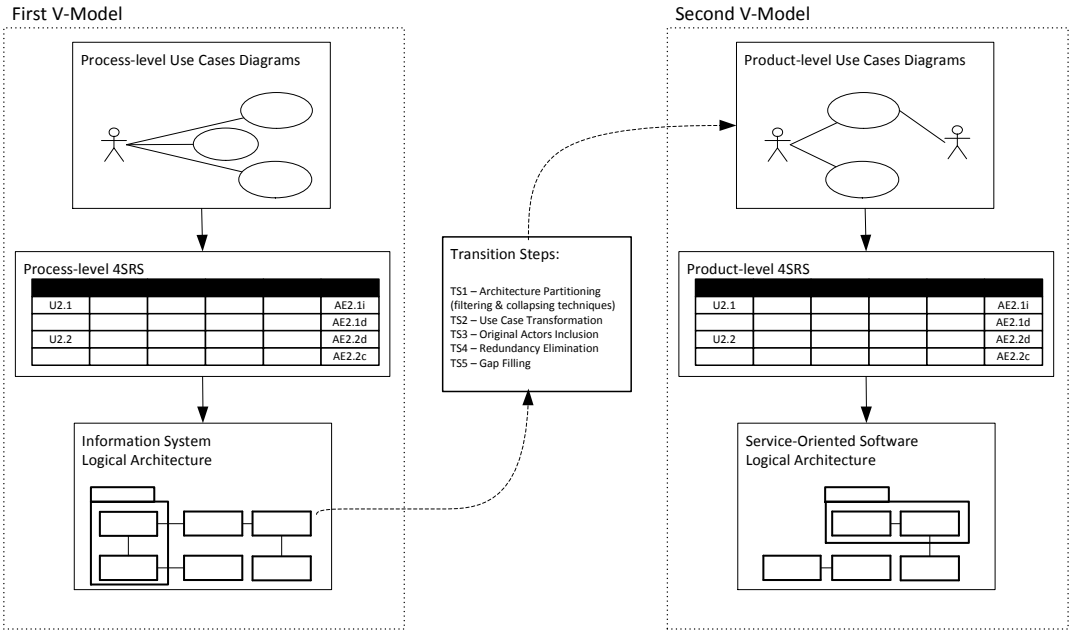

Figure 39: Derivation of service-oriented logical architectures by transiting from information system logical architectures.

As depicted in Figure 39, the result of the first V-Model (process-level) execution is the information system logical architecture. The architectural elements that compose this architecture are derived (by performing transition steps) into product-level use cases (Mashed UCs model). The result of the second V-Model (product-level) execution is the service-oriented software logical architecture.

In both V-Models execution, the assessments that result from comparing *A*- and *B-type sequence diagrams* produce *Issues* documents. These documents are one of the outputs of the previously presented ARID method used to assess each V-Model execution. ARID is able to conduct reviews regarding architectural decisions, namely on the quality attributes requirements and their alignment and satisfaction degree of specific quality goals. At the same time, ARID is able of performing evaluations on parts of the global architecture. Those features made ARID our method of choice regarding the evaluation of the in-progress logical architecture and in the assistance to determine the need of further refinements, improvements, or revisions before assuming that the architecture is ready to be delivered to the teams responsible for implementation. This delivery is called context for product implementation (*CPI)*.

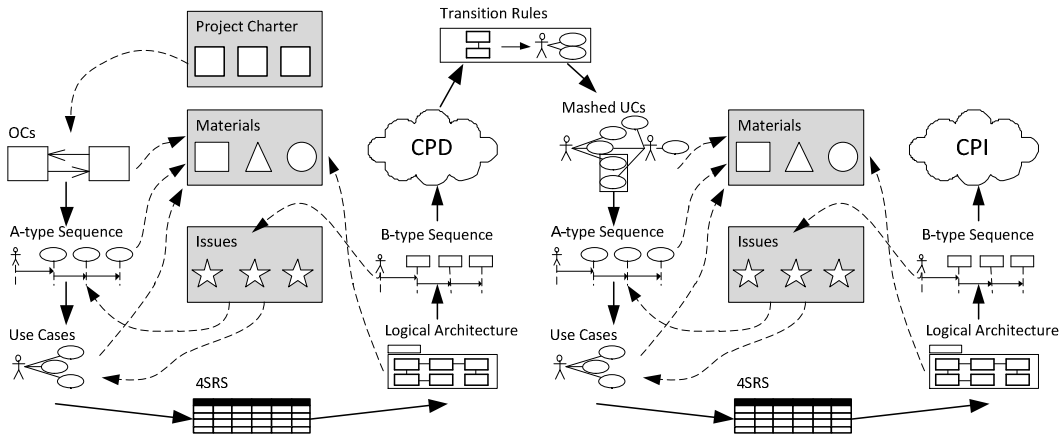

Figure 40: Assessment of the V+V execution using ARID

In Figure 40, we present the simplified interactions between the ARID-related models in the V+V process. In this figure, we can see the macro-process associated with both V-Models, the transition from one to the other (later detailed) and the ARID models that support the assessment of the V+V execution.

Our application of common architectural patterns include business, analysis, architectural and design patterns as defined in (S. Azevedo, Machado, Bragança, & Ribeiro, 2010). By applying them as early as possible in the development (in early analysis and design), it is possible to incorporate business requirements into the logical architectural model and at the same time assure that the resulting model is aligned with the organization needs and also complies with the established non-functional requirements. The design patterns are used in particular when there is a need to detail or refine parts of the logical architecture.

In the second V, after being positively assessed by the ARID method, the business software logical architecture model is considered a final design artifact that must be divided into products (applications) for latter implemented by the software teams.

5.3 Creating Context for Product Implementation

As stated before, a process-level V-Model can be executed for business requirements elicitation purposes, followed by a product-level V-Model for defining the software functional requirements. The V+V process is useful for both stakeholders,

organizations and technicians, but it is necessary to assure that they properly reflect the same system.

This section presents a set of transition steps whose execution is required to create the *Mashed UC* model referred in Figure 38 and in Figure 40. The purpose of these transition steps is to assure an aligned transition between the process- and product-level perspectives in the V+V process (Ferreira, Santos, Soares, Machado, & Gašević, 2013).

To allow the recursive execution of the 4SRS method (Sofia Azevedo, et al., 2009; R. J. Machado, Fernandes, Monteiro, & Rodrigues, 2006a), the transition from the first V-Model to the second V-Model must be performed by a set of steps. The output of the first V-Model must be used as input for the second V-Model; *i.e.*, we need to transform the information system logical architecture into product-level use case models.

The transition steps to guide this mapping must be able to support a business to technology changing (Ferreira, Santos, Soares, et al., 2013). By defining these transition steps, we assure that product-level (software) use cases (UCpt) are aligned with the architectural elements (AEpc's) from the process-level logical architecture diagram (AEpc); i.e., software use case diagrams are reflecting the needs of the information system logical architecture. The transition steps (TS), represented in Figure 41, are structured as follows:

- TS1 – *Architecture Partitioning*: By applying *collapsing and filtering techniques* as detailed in (R. J. Machado, et al., 2006b), it is possible to identify major groups of elements in the information system logical architecture that must be computationally supported by software. In this transition step, the AEpc's under analysis are classified by their computation execution context with the purpose of defining software boundaries to be transformed into UCpt's. The final software boundary is represented after the execution of filtering and collapsing techniques in the AEpc's. Each of the identified major groups of elements is subject to a separate execution in the following transition steps.

- TS2 – *Use Case Transformation*: This transition step is applied to each partition defined in the previous transition step (i.e., to each major groups of elements) with the purpose of transforming elements of the information system logical architecture (AEpc's) into software use cases and actors. In this transition step, AEpc's are transformed into software use cases and actors that represent the system under analysis. This is the most critical transition step of the transition process and, as such, we have devised a set of transition patterns that must be applied as rules that are later described in this section.

- TS3 – *Original Actors Inclusion*: For each defined partition, the original actors that were related to the (business) use cases from which the architectural elements of the process-level perspective are derived (in the first V execution) must be included in the representation. The purpose of this transition step is to introduce into the product-level perspective the necessary information regarding the skills and stakeholders of the originally defined processes. The traceability between the process-level (business) use cases and the AEpc's is assured by the process-level 4SRS execution (Nuno Ferreira, et al., 2012a).

- TS4 – *Redundancy Elimination*: In the previous transition steps there is a possibility of including redundancy in the model in the form of actors and use cases generated by the transition rules. For each partition defined in the first transition step, it is important to remove such redundancy by explicitly removing the unnecessary actors and use cases from the model.

- TS5 – *Gap Filling*: This final transition step intents to create, in the form of use cases to be added to the model, the necessary information of any requirement that is intended to be part of the design and that is not yet present. Typical missing use cases are connections between existing use cases that were automatically created by the transition rules.

During the execution of these transition steps, transition use cases (UCtr) bridge the AEpc's and serve as basis to elicit UCpt's. UCtr's also provide traceability between process- and product-level perspectives using tags and annotations associated with each representation.

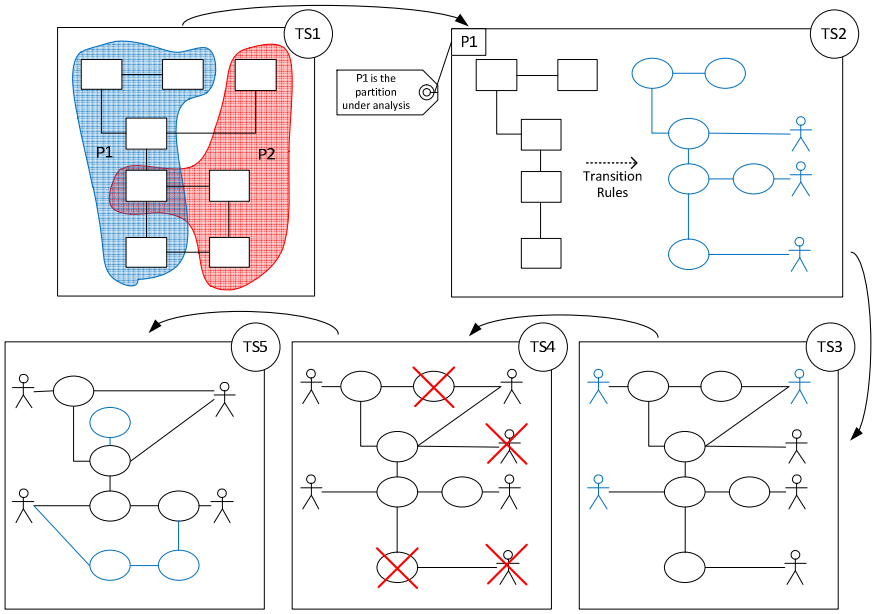

Figure 41: Process- to product-level transition

The rules to support the execution of transition step 2 (TS2) are applied in the form of transition rules and must be applied in accordance to the stereotype of the envisaged architectural element. There are three stereotyped architectural elements:

- *d-type*, which refer to generic decision repositories (data), representing decisions not supported computationally by the system under design;
- *c-type,* which encompass all the processes focusing on decision making that must be supported computationally by the system;
- *i-type*, which refer to process' interfaces with users, software or other processes.

The full descriptions and specifications of the three stereotypes are available in (Nuno Ferreira, et al., 2012a).

For the sake of understandability, in Figure 42, we present an excerpt of the UML extension that supports the creation of AEpc's, UCtr's and partitions. We consider that a partition is a container of AEpc's or UCtr's and acts as a border delimiter for the combinations of possible systems candidates to be analyzed. After delimiting all the partitions, it is necessary to focus on a particular one (called inbound partition) and

execute the required transformations considering all the remaining neighbor partitions (outbound partitions).

The identification of each partition is firstly made using the information that results from the packaging and aggregation efforts of the previous 4SRS execution (step 3 of the 4SRS method execution as described in (Ferreira, Santos, Machado, & Gasevic, 2012b)). Nevertheless, this information is not enough to identify properly the partitions. Information gathered in OC's and on the process-level *B-type* sequence diagrams must also be accounted. A partition is created by identifying all the relevant architectural elements that belong to all *B-type* sequence diagrams that correspond to a given organizational configuration scenario. By traversing the architectural elements that comply with the scenario definition (for each *B-type* sequence diagram and aligned with the packages and aggregations presented in the information system logical architecture), it is possible to properly identify the partitions that support the interactions depicted in the OC's.

A proper way of defining the transformations between models is by means of using OMG's QVT (OMG, 2011a). QVT is a set of languages (QVT-Operational, QVT-Relations, and QVT-Core) that enables models transformations. QVT-Operational enables unidirectional transformations of a given model into another. QVT-Relations allow bi-directional transformations. QVT-Core can be considered a subset of QVT-Relations. The QVT set of languages are associated with model-driven approaches. These model driven approaches are usually associated with design and implementation models and lack support to requirements and analysis models. The requirements specification (in any perspective) is a crucial task in any software development process. As such, models that support requirements specification should be integrated into model-driven methods.

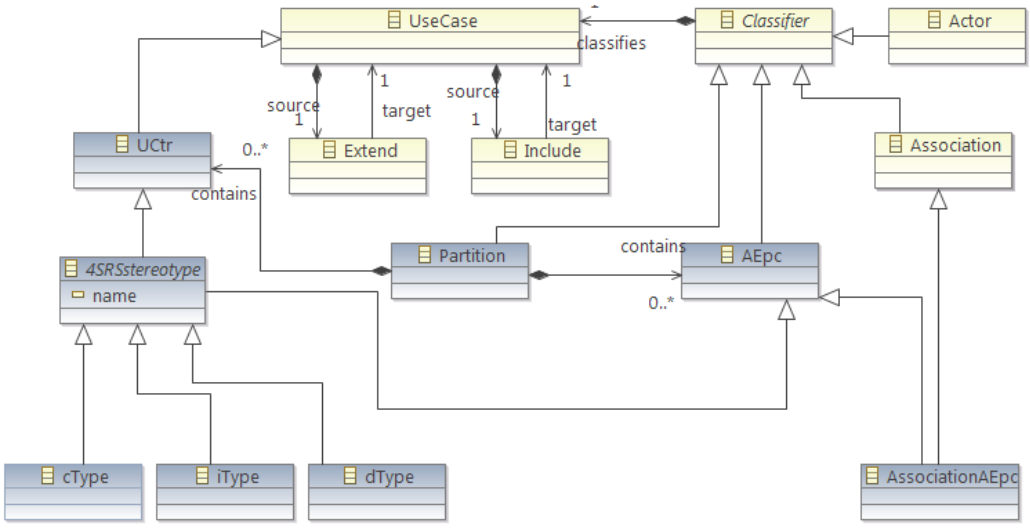

Figure 42: Excerpt of AEpc and UCtr Extension

In our proposed approach we have chosen QVT as a mean to transform AEpc's models into UCtr's models, or being more specific, transforming information system logical architectural models into *Mashed UC* models. This relates to integrating models that support requirements specifications into a model-driven approach. Associated with the transition rules, we present a subset of the QVT-Operational (-like) code that supports the transformation intended by a given rule. The defined transition rules, from the logical architectural diagram to the *Mashed UC* diagram are as follows:

- TR1: an inbound *c-type* or *i-type* AEpc is transformed into an UCtr of the same type (see Figure 43). By inbound we mean that the element is inside the partition under analysis;

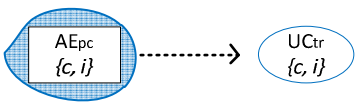

Figure 43: TR1 - transition rule 1

The QVT-like specification that supported the transformation implementation for TR1 is as follows:

```
if (AEpc.Partition=inbound) and (AEpc.4SRSstereotype=cType or
AEpc.4SRSstereotype=iType) then {
UCtr.name:=Aepc.name;
UCtr.4SRSstereotype:=AEpc.4SRSstereotype}
```

```
endif;
```

- TR2: an inbound *d-type* AEpc is transformed into an UCtr and an associated actor (see Figure 44). This is due to the fact that *d-type* AEpc's corresponds to decisions not computationally supported by the system under design and, as such, it requires an actor to activate the depicted process.

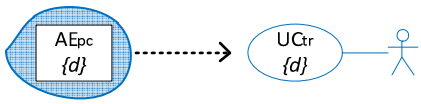

Figure 44: TR2 - transition rule 2

TR2 is supported by the following:

```
if (AEpc.Partition=inbound) AND (AEpc.4SRSstereotype=dType) then {
UCtr.name:=AEpc.name;
UCtr.4SRSstereotype:=AEpc.4SRSstereotype;
Actor.name:=self.name;
Actor.association:=UCtr}
endif;
```

Rules TR1 and TR2 are the most basic ones and the patterns they express are the most used in transition step 2.

- TR3: an inbound AEpc, with a given name *x*, which also belongs to an outbound partition, is transformed into an UCtr of name *x*, and an associated actor, of name *y*, being the responsible for representing the outbound actions associated with UCtr*x* (see Figure 45).

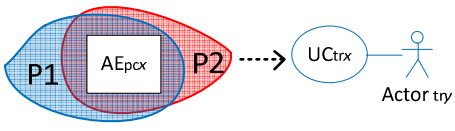

Figure 45: TR3 - transition rule 3

The specification for TR3 is:

```
if (AEpc.Partition=multiple) and (AEpc.4SRSstereotype=cType) then
{
UCtr.name:=AEpc.name;
UCtr.4SRSstereotype:=AEpc.4SRSstereotype;
```

```
Actor.name:=self.name;
Actor.association:=UCtr }
endif;
```

The connections between the use cases and actors produced by the previous rules must be consistent with the existing associations between the AEpc's. The focus of this analysis is UCtr's and they are addressed by the following two transition rules.

- TR4: an inbound *d-type* UCtr of name *x* with connections to an (any type) UCtr of name *y* and to an actor *z*, gives place to two UCtr's, *x* and *y*, maintaining the original types (see Figure 46). Both are connected to the actor *z*. This means that all existing connections on the original *d-type* AEpc that were maintained during execution of TR2 or TR3 are transferred to the created actor.

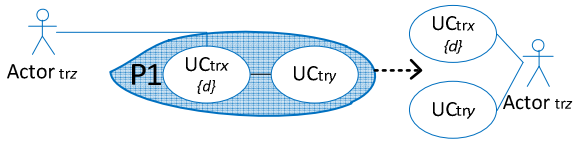

Figure 46: TR4 - transition rule 4

Regarding TR4, the necessary specification is:

```
if (UCtr.Partition=inbound) and (UCtr.4SRSstereotype=dType) and
(Actor.associations().FilterByPartition(UCtr).Count > 1) then {
Actor.Association:=
Actor.associations().FilterByPartition(UCtr _
).GetUCtr()) }
endif;
```

- TR5: an inbound UCtr of name *x* with a connection to an outbound AEpc of name *y* (note that this is still an AEpc, since it was not transformed into any other concept by the previous transition rules) gives place to both an UCtr named *x* and to an actor named *y* (see Figure 47). AEpc's that were not previously transformed are now transformed by the application of this TR5; this means that all AEpc's which exist outside the partition under analysis having connections with inbound UCtr's will be transformed into actors. These actors will support the representation of required external inputs to the inbounds UCtr's created during application of TR1, TR2, or TR3.

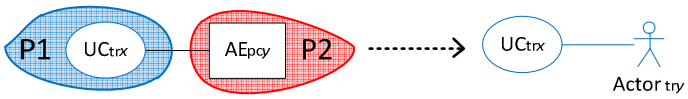

Figure 47: TR5 - transition rule 5

For TR5, the supporting specification is:

```
if (AEpc.Partition=outbound) then {
Actor.name:=Aepc.name
Actor.Association:= Actor.associations(). _
FilterByPartition(UCtr).GetUCtr()) }
endif;
```

A special application of TR5 can be found in Figure 48 where we can see an UCtr with a connection to an outbound AEpc and another connection to an actor. In this case, TR5 is applied and the resulting UCtr is also connected to the original actor.

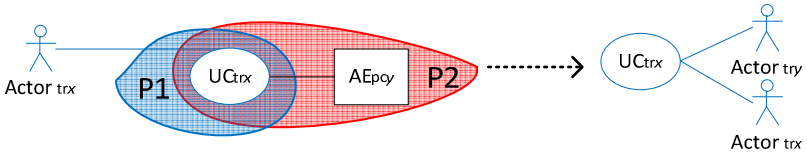

Figure 48: TR5.1 - transition rule 5.1

The application of these transition steps and rules to all the partitions of an information system logical architecture gives origin to a set of *Mashed UC* models. In the next section, we present a case study where the logical architecture of an information system is transformed into a product-level *Mashed UC* model by executing the transition steps. In the remaining transition steps, the purpose is to promote completeness and reliability in the model. The model is complete after adding the associations that initially connected actors (the ones who triggers the AEpc's) and the AEpc's, and then by mapping those associations to the UCtr's. The model is reliable since the enforcement of the rules eliminates redundancy and assures that there are no gaps in the UCtr's associations and related actors. Only after the execution of all the transition steps we consider the resulting model as containing product-level use cases (UCpt's).

5.4 The V+V Model in the ISOFIN Project

The applicability of the proposed approach was assessed with a real project that is analyzed in this thesis as a case study: the ISOFIN project (ISOFIN Project Consortium, 2010). This project aimed to deliver a set of coordinating services in a centralized infrastructure, enacting the coordination of independent services relying on separate infrastructures. The resulting ISOFIN platform, allows for the semantic and application interoperability between enrolled financial institutions (Banks, Insurance Companies and others).

From the case study, we first present the process-level logical architecture, that resulted from the execution of the 4SRS method at a process-level perspective (Nuno Ferreira, et al., 2012a). In Figure 49, we depict the execution of TS1, i.e., the partitioning of the process-level logical architecture, which resulted in two partitions:

(i) the ISOFIN platform execution functionalities (in the area marked as P1);

(ii) the ISOFIN supplier execution functionalities (in the area marked as P2).

The identification of the partitions will enable the application of the transition steps to allow the application of the second V-Model to follow the macro-process execution into the product (software) implementation.

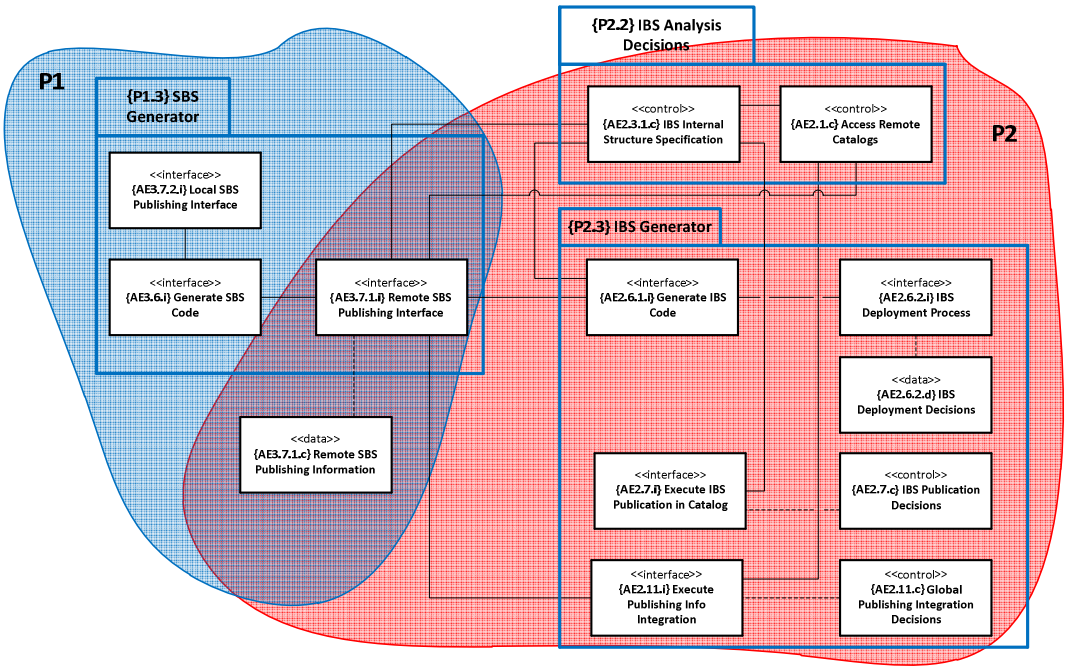

Figure 49: Partitioning of the process-level logical architecture (TS1)

The identification of each partition is firstly made using the information that results from the packaging and aggregation efforts of the previous 4SRS execution (step 3 of the 4SRS method execution as described in (Nuno Ferreira, et al., 2012a)). Nevertheless, this information is not enough to properly identify the partitions. Information gathered in organizational configurations and on the process-level *B-type sequence diagrams* must also be accounted. A partition is then created by identifying all the relevant architectural elements that belong to all *B-type sequence diagrams* that correspond to a given organizational configuration scenario. By traversing the architectural elements that comply with the scenario definition (aligned with the packages and aggregations presented in the logical architecture), it is possible to properly identify the partitions.

Figure 50 shows the filtered and collapsed diagram that resulted from the P2 partition, which (in the case study), is the partition under analysis. P2 includes the architectural elements that belong to both partitions and that must be considered when applying the transition rules. After being filtered and collapsed, the partitioned logical architecture is composed not only by the architectural elements that belong to the partition under analysis, but also by some additional architectural element belonging to any other partition having associations with architectural elements belonging to the

partition under analysis (e.g., *{AE3.6.9.i} Generate SBS Code* belongs to P1, but possesses an association with *{AE3.7.1.i} Remote SBS Publishing Interface* that belongs to P1 and P2 partitions). The keeping of these outbound AEpc's assures that outbound interfaces information is preserved.

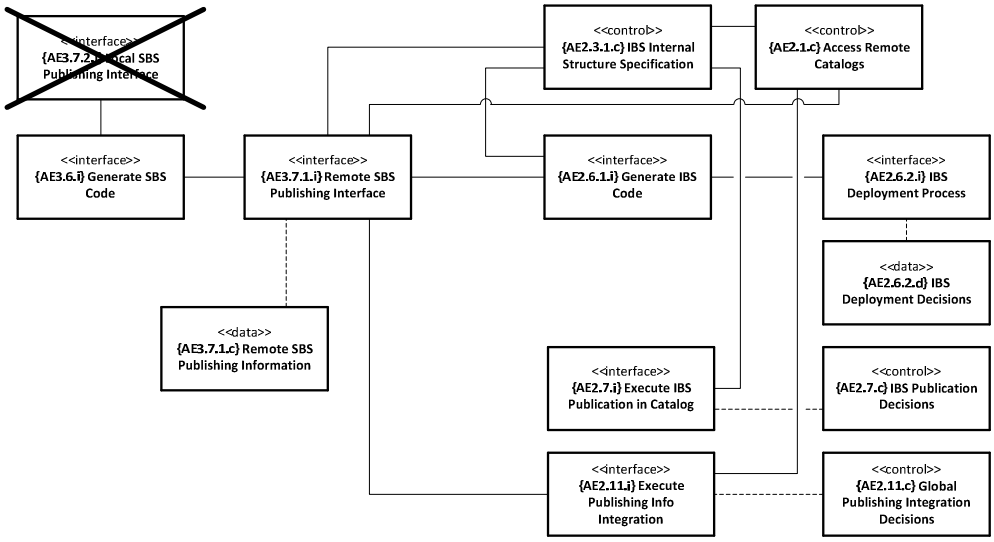

Figure 50: Filtered and collapsed architectural elements (TS1)

In. Figure 51 we depict an example of a subset of an information system logical architecture composed of architectural elements that represent processes, already partitioned (for the sake of understandability, AEpc's are colored as presented in the transition rules, in this case, for P2 and the ones that are common to both, the blank AEpc is an outbound).

Table 7: Transition Steps Overview

Transition Step	Description	Perspective
TS1	the AEpc's under analysis are classified by their computation execution context	process-level
TS2	AEpc's are transformed into software use cases and actors that represent the system under analysis through a set of transition patterns that must be applied as rules	product-level
TS3	the original actors that were related to the use cases from which the architectural elements of the process-level perspective are derived (in the first V execution) must be included in the representation	product-level
TS4	the model is analyzed for redundancies	product-level
TS5	the necessary information of any requirement that is intended to be part of the design and that is not yet present is added, in the form of use cases	product-level

Table 7 indicates that the transition process starts in the process-level perspective with AEpc's. After TS1, the transition is still dealing with AEpc's as input; the execution of TS2 results in the perspective transition, since UCtr's relate to product-level; in the remaining transition steps, the purpose is to promote completeness and reliability in the model. The model is complete after adding the associations that initially connected actors (the ones who triggers the AEpc's) and the AEpc's, and then by mapping those associations to the UCtr's. The model is reliable since the enforcement of the rules eliminates redundancy and assures that there are no gaps in the UCtr's associations and related actors. Only after the execution of all the transition steps we consider the resulting model as containing product-level use cases (UCpt's).

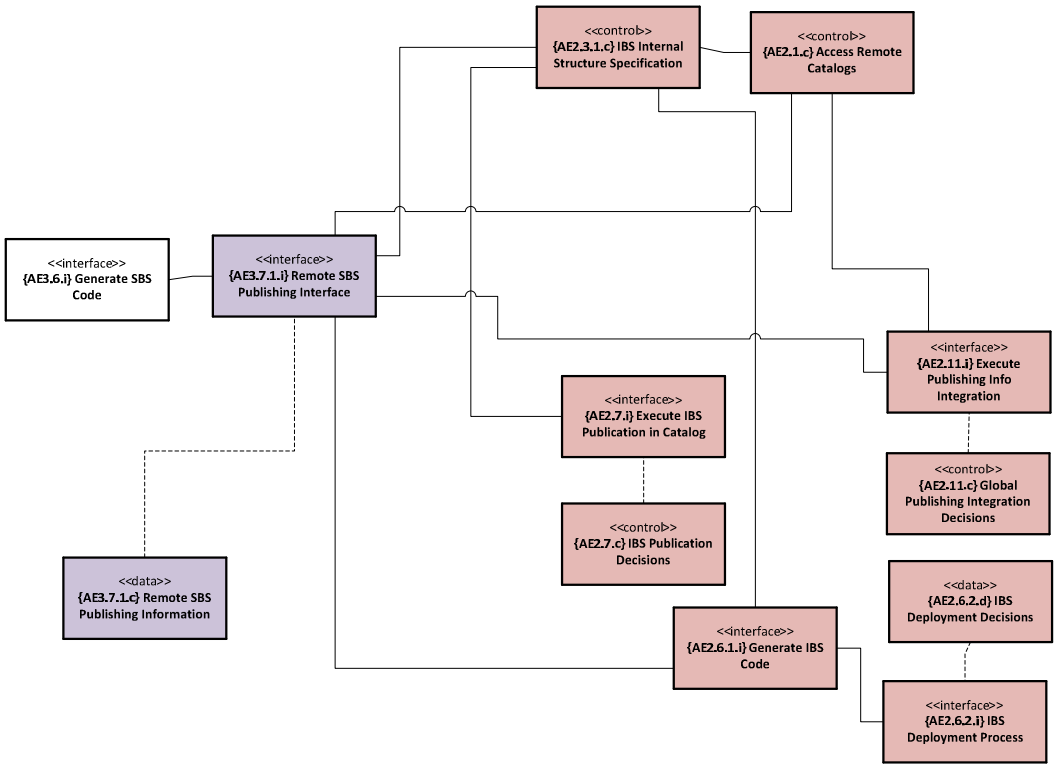

Figure 51: Information system logical architecture example

In Figure 52, we depict the final mashed use case model (the first product-level artifact in the second V), resulting from the execution of the transition rules 2 to 5. In this work we only show the result of the execution of these four transition steps altogether. The complete description can be found in the ISOFIN Technical Deliveries (ISOFIN Project Consortium, 2010). The resulting mashed use cases are the result of the application of the transition rules in TS2. It is possible to objectively recognize the effect of the application of some transition rules previously described. TR1 was the most applied transition rule and one example is the transformation of the AEpc named *{AE2.1.c} Access Remote Catalogs* into one UCtr named *{U2.1.c} Access Remote Catalogs*. One example of the application of TR2 is the transformation of the AEpc named *{AE2.6.2.d} IBS Deployment Decisions* into the UCtr named *{U2.6.2.d} Define IBS Deployment* and the actor named *IBS Developer*. TR3 was applied, for instance, in the transformation of the AEpc named *{AE3.7.1.c} Define SBS Information* into the UCtr named *{U3.7.1.c} Define SBS Information* and the actor named *SBS Publisher*. Finally, we can recognize the application of TR5.1 in the transformation of the AEpc named *{AE3.6.9.i} Generate SBS Code* into the actor named *SBS Developer*.

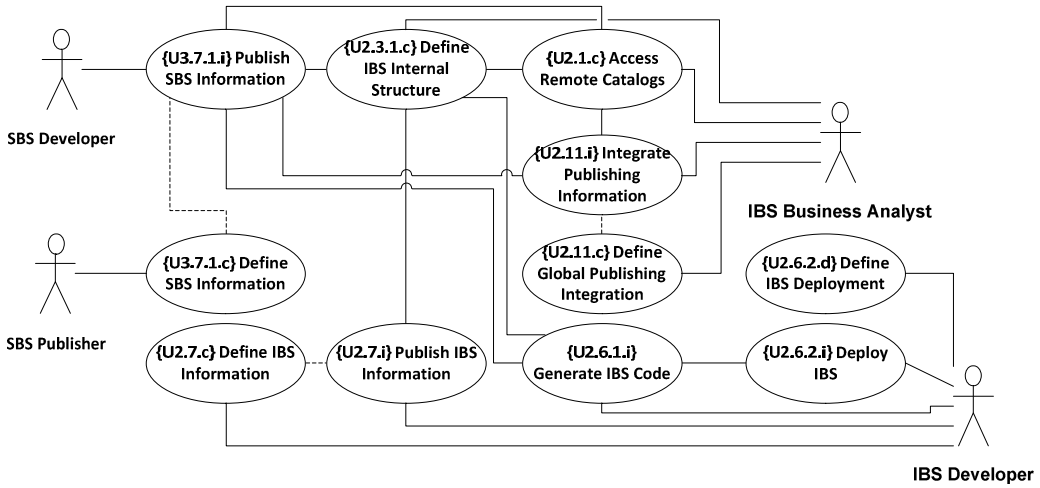

Figure 52: Mashed UC model resulting from the transition from process- to product-level

Table 8: Executed transformations to the model

Process-level (transformation source)	TR	Product-level (transformation target)
AEpc *{AE2.1.c} Access Remote Catalogs*	TR1	UCtr *{U2.1.c} Access Remote Catalogs*
AEpc *{AE2.3.1.c} IBS Internal Structure Specification*	TR1	UCtr *{U2.3.1.c} Define IBS Internal Structure*
AEpc *{AE2.6.1.i} Generate IBS Code*	TR1	UCtr *{U2.6.1.i} Generate IBS Code*
AEpc *{AE2.6.2.d} IBS Deployment Decisions*	TR2	UCtr *{U2.6.2.d} Define IBS Deployment*; Actor *IBS Developer*
AEpc *{AE2.6.2.i} IBS Deployment Process*	TR1	UCtr *{U2.6.2.i} Deploy IBS*
AEpc *{AE2.7.i} Execute IBS Publication in Catalog*	TR1	UCtr *{U2.7.i} Publish IBS Information*
AEpc *{AE2.7.c} IBS Publication Decisions*	TR1	UCtr *{U2.7.c} Define IBS Information*
AEpc *{AE2.11.i} Execute Publishing Info Integration*	TR1	UCtr *{U2.11.i} Integrate Publishing Information*
AEpc *{AE2.11.c} Global Publishing Integration Decisions*	TR1	UCtr *{U2.11.c} Define Global Publishing Information*
AEpc *{AE3.6.i} Generate SBS Code*	TR5.1	Actor *SBS Developer*
AEpc *{AE3.7.1.i} Remote SBS Publishing Interface*	TR3	UCtr *{U3.7.1.i} Publish SBS Information*; Actor *SBS Developer*
AEpc *{AE3.7.1.c} Remote SBS Publishing Information*	TR3	UCtr *{U3.7.1.c} Define SBS Information*; Actor *SBS Publisher*

All the other actors result from the execution of TS3. We must be referred, for instance, that the actor *SBS Developer* results from the execution of TS4, since the original actor and the actor resulting from an application of TR2 and TR5.1 and also the inclusion of the original actor in TS3, result in the same actor which brings the need to eliminate the generated redundancy. The resulting model allows to identify potential gaps in use cases or actors (in the execution of TS5), but in this case such wasn't required.

After the execution of the transition steps, the Mashed UC model is used as input for the product-level 4SRS method execution in order to derive the service-oriented logical architecture for the ISOFIN platform. We depict in

Figure 53 the entire service-oriented software logical architecture obtained after the execution of the V+V process, having as input the information system logical architecture previously presented. The service-oriented software logical architecture is composed by architectural elements that represent services that are executed in the platform. It would be impossible to elicit requirements for a service-oriented logical architecture as complex as the ISOFIN platform by adopting an approach that only considers the product-level perspective. It is also possible to observe in

Figure 53 the alignment (supported by the transition steps) between the architecture elements in both perspectives.

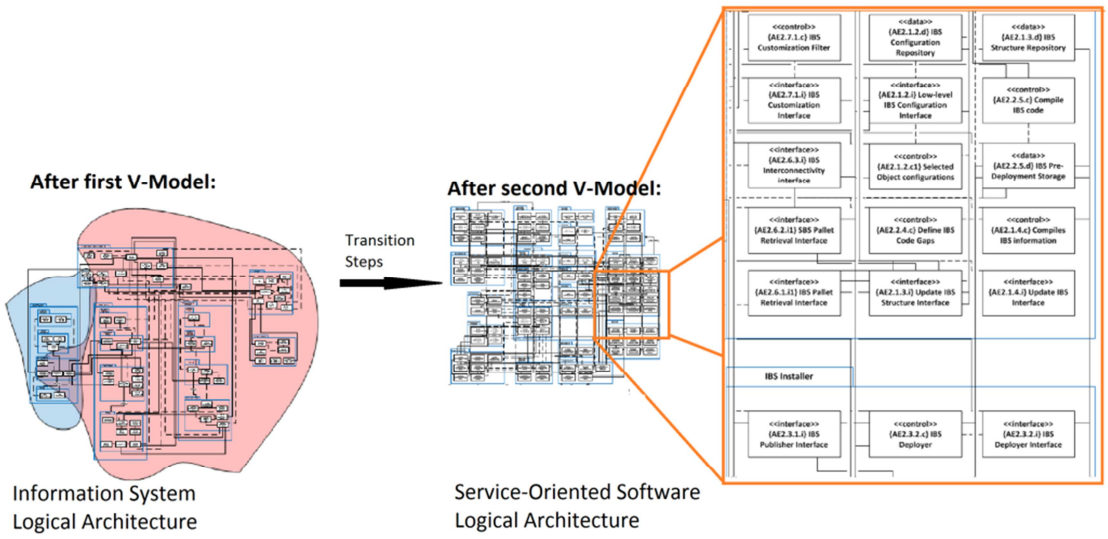

Figure 53: Subset of the ISOFIN service-oriented software logical architecture based on the information system logical architecture

5.5 Transition Rules in Other's Work

An important view considered in our approach regards the architecture. What is architecture? In the literature, there is a plethora of definitions but most agree that an architecture concerns both structure and behavior, with a level of abstraction that only regards significant decisions and may be in conformance with an architectural style, is influenced by its stakeholders and the environment where it is intended to be instantiated and also encompasses decisions based on some rationale or method.

It is acknowledged in software engineering that a complete system architecture cannot be represented using a single perspective (Kruchten, 1995; Sungwon & Yoonseok, 2005). Using multiple viewpoints, like logical diagrams, sequence diagrams or other artifacts, contributes to a better representation of the system and, as a consequence, to a better understanding of the system. Some architecture views can be seen in the works of Clements *et al* (P. Clements, et al., 2003), Hofmeister et *al* (Hofmeister, et al., 2000) and Krutchen (Kruchten, 1995). Krutchen's work refers that the description of the architecture can be represented into four views: logical, development, process and physical. The fifth view is represented by selected use cases or scenarios. Zou and Pavlovski (Zou & Pavlovski, 2006) add an extra view, the control case view, that complements the use case view to complete requirements across the collective system lifecycle views. Our stereotyped usage of sequence diagrams adds more representativeness value to the specific model than, for instance, the presented in Krutchen's 4+1 perspective (Kruchten, 1995). This kind of representation also enables testing sequences of system actions that are meaningful at the software architecture level (Bertolino, et al., 2001). Additionally, the use of this kind of stereotyped sequence diagrams at the first stage of analysis phase (user requirements modeling and validation) provides a friendlier perspective to most stakeholders, easing them to establish a direct correspondence between what they initially stated as functional requirements and what the model already describes.

The relation between what the stakeholders want and what implementation teams need requires an alignment approach to assure that there are no missing specifications on the transition between phases.

An approach that enacts the alignment between domain-specific needs and software solutions, is the goal oriented approach GQM+Strategies (Goal/Question/Metric + Strategies) (Basili, et al., 2010). The GQM+Strategies approach uses measurement to explicitly link goals and strategies from business objectives to project operations. Another goal-oriented approach is the Balanced Scorecard (BSC) (Kaplan & Norton, 1992). BSC links strategic objectives and measures through a scorecard in four perspectives: financial, customer, internal business processes, and learning and growth. It is a tool for defining strategic goals from multiple perspectives beyond a purely financial focus.

Another approach, COBIT (Information Technology Governance Institute (ITGI), 2012), is a framework for governing and managing enterprise IT. It provides a comprehensive framework that assists enterprises in achieving their objectives for the governance and management of enterprise IT. It is based on five key principles:

(i) meeting stakeholder needs;

(ii) covering the enterprise end-to-end;

(iii) applying a single, integrated framework;

(iv) enabling a holistic approach;

(v) separating governance from management.

In our understanding, none of the previous approaches encompasses processes for deriving a logical representation of the intended system processes with the purpose of creating context for eliciting product-level requirements. Those approaches have a broader specification concerning risk analysis, auditing, measurement, or best practices in the overall alignment strategy.

The process architecture represents a fundamental organization of service development, service creation, and service distribution in the relevant enterprise context.

Designing software architecture based on a process-level perspective provides a more accurate definition of the requirements. There are several approaches to supporting the proper design of software architectures, like FAST (D. M. Weiss & Lai, 1999), FORM (Kang, et al., 1998) or KobrA (Bayer, et al., 2001). These all relate to the product-level perspective. In a process-level perspective, Tropos (Castro, et al., 2002) is a methodology that uses notions of actor, goal and (actor) dependency as a foundation to model early and late requirements, architectural and detailed design. Our approach uses the functional refinement of use cases and uses them, alongside with textual descriptions, as input to the 4SRS method to derive a logical architecture. Logical architectures can be faced as a view of a system composed by a set of problem-specific abstractions supporting functional requirements (Kruchten, 1995) and thus giving detail to the design of the information system.

The defined and derived models suggested by our approach, used alone and unaligned with each other, are of a lesser use to organizations and stakeholders. Our approach begins in a domain-specific perspective (usually in the business-level), by defining the organizational configurations that represent major interactions, at a very high-level, in the chosen domain, and ends with a technological view of the system. From one perspective to the other, alignment must be assured. The alignment we refer to relates to domain-specific and software alignment (Campbell, 2005), and in our case, where the domain-specific needs must be instantiated into the creation of context for proper product design.

There are many approaches (Dijkman & Joosten, 2002a) (OMG, 2003b) (Bezivin, Dupé, Jouault, Pitette, & Rougui, 2003) that allow deriving at a given level a view of the intended system to be developed. Our approach clearly starts at a process-level perspective, and by a successive derivation of models creates the context for transforming the requirements expressed in an information system logical architecture

into product-level context for requirements specification. Other approaches provide similar results at a subset of our specification.

The work presented in (Dijkman & Joosten, 2002a) and (Dijkman & Joosten, 2002b), specifies a mapping technique and an algorithm for mapping business process models, using UML activity diagrams, and use cases, so functional requirements specifications support the enterprise's business process. In our approach, we use an information system logical architecture diagram instead of an activity diagram, since an information system logical architecture provides a fundamental organization of the development, creation, and distribution of processes in the relevant enterprise context (Winter & Fischer, 2006a).

In literature, model transformations are often related to the Model-Driven Architecture (MDA) (OMG, 2003b) initiative from OMG. A MDA-based approach uses model transformations in order to transform a high-level model (Platform-Independent Model – PIM) to a lower-level model (Platform-Specific Model – PSM). MDA-based model transformations are widely used but, as far as the authors know, the supported transformations do not regard perspective transition, *i.e.*, are perspective agnostic since they concern model transformations within a single perspective (typically the product-level one). For instance, (Kaindl & Falb, 2008) describes MDA-based model transformations from use cases and scenarios to components, but only in a product-level perspective. Even in cases when MDA model transformations are executed using different source and target modeling languages (for instance, in (Bauer, Müller, & Roser, 2004) a PIM is modeled in Business Process Modeling Notation – BPMN, and its model is transformed into a PSM modeled in Business Process Execution Language – BPEL), the transformation only regards a single perspective. The authors in (Bezivin, et al., 2003) present technological spaces and model transformations between them, but the technological space domains also only regard a single perspective. The concerns that must be assured by transiting from one perspective to the other are not dealt by any of the previous works.

The existing approaches for model transformation attempt to provide an automated or automatic execution. (Yue, et al., 2011) provides a systematic review and evaluation of

existing work on automating of transforming requirements into an analysis model and, according to the authors, none of the compared approaches provide a practical automated solution. The transition steps and rules presented in this work intent to provide a certain level of automation into our approach and improve the efficiency, validation, and traceability of the overall V+V process. The transitions depicted in the present work are able to be fully implemented in development tools that support QVT transformations, like the well-known Eclipse IDE.

5.6 Conclusions

In this chapter, we have presented an approach to creating context for business software implementation teams in contexts where requirements cannot be properly elicited. Our approach is based on successive models construction and recursive derivation of logical architectures, and makes use of model derivation for creating use cases, based on high-level representations of desired system interactions.

The approach assures that validation tasks are performed continuously along the modeling process. It allows for validating:

(i) the final software solution according to the initial expressed business requirements;

(ii) the *B-type sequence diagrams* according to *A-type sequence diagrams*;

(iii) the logical architectures by traversing it with *B-type sequence diagrams*. These validations task, specific to the V-Model, are subject of a future publication.

We also presented a set of transition steps and transition rules in order to execute the transition from process- to product-level perspective. These transition steps use as basis a process-level logical architecture and stereotyped sequence diagrams to output a product-level use case model. This approach allows requirements in a technological (product-level) perspective to be properly aligned with organizational business (process-level) requirements in a traceable way.

It is a common fact that domain-specific needs, namely business needs, are a fast changing concern that must be tackled. Process-level architectures must be in a way that potentially changing domain-specific needs are local in the architecture representation.

Our proposed V+V process encompasses the derivation of a logical architecture representation that is aligned with domain-specific needs and any change made to those domain-specific needs is reflected in the logical architectural model through successive derivation of the supporting models (OCs, *A*- and *B-type sequence diagrams*, and use cases). Additionally, traceability between those models is built-in by construction, and intrinsically integrated in our V+V process.

Software architecture representations serve two purposes: First, they act as a common abstraction of the system providing a representation of the system able to be understood by all the stakeholders regardless of their background. Second, the architecture is a model of the intended system to be built, modified or analyzed.

A system logical architecture can be viewed as a constructed set of the system's design decisions. By constructed, we mean that the architecture is built using a construction method that assures its correctness. Design decisions, at this level, can be analyzed by looking at the non-functional requirement that the system is intended to comply. For instance, if we intend our system to be secure, the architect should pay attention to the communication between architectural elements represented in the logical architecture diagram and also to the data flows between them or to the existence of special encryption or authentication elements. If the system is required to be redundant, the architect should care about redundant sub-systems or architectural elements.

The V+V-Model is able to conduct reviews regarding architectural decisions, namely on the quality attributes requirements and their alignment and satisfaction degree of specific quality goals that are imposed to the created scenarios (*A-type* sequence diagrams). The several models can be used supporting documentation that can be provided to stakeholders and for promoting the validation of described scenarios. These quality attributes reviews were not explicitly done in the ISOFIN project. Instead,

those requirements were imbued in design decisions related to the logical architecture.

Our approach uses software engineering techniques, such as operational model transformations to assure the execution of a process that begins with business needs and ends with a logical architectural representation of a service-based system. It is a common fact that domain-specific needs, namely business needs, are a fast changing concern that must be tackled. Information system architectures must be modeled in a way that potentially changing domain-specific needs are local in the architecture representation of the intended service. Our proposed V+V process encompasses the derivation of a logical architecture representation that is aligned with domain-specific needs and any change made to those domain-specific needs is reflected in the logical architectural model, and the transformation is properly assured. Since the *Mashed UC* model is derived from a model transformation based on mappings, traceability between AEpc's and UCpt's is guaranteed, thus any necessary change on product-level requirements due to a change on a given business needs is easily identified and propagated alongside the models that comprise the V+V process.

Chapter 6

Conclusion

Chapter Contents

6
Conclusion

This chapter presents the conclusion of this thesis. Here we present a final overview on the V+V-Model approach and then we synthesize the research efforts, the major contributions and an outlook of the research roadmap that should follow our efforts.

6.1 Focus of the Work

During an information system development process, assuring that functional requirements fully support the stakeholder's business needs may become a complex and inefficient task. Additionally, the "newfound" paradigm of IT solutions (*e.g.*, Cloud Computing) typically results in more difficulties for defining a business model and for eliciting product-level functional requirements for any given project, than properly specifying how the intended services should be provided and executed. If stakeholders experience such difficulties then software developers will have to deal with incomplete or incorrect requirements specifications, resulting in a real problem for them, increasing costs, time and effort deviations, inefficiencies in the process and eventually not delivering the project in the intended scope.

In this work, we have described a process that begins in a scenario where there is no consensus in the definition of the business model for the target (intended) solution (software product). We construct a set of models that encompass the V-model approach and then, by successive model derivation, derive an initial process-level logical architecture. By applying the detailed transition rules to the process-level logical architecture it is possible to create context for product design, that is, the initial requirements and use cases that describe the intended usage of the software solution. The method follows in a second V-model from the context for product design to a product-level logical architecture. This architectural representation is the basis for the development of the software solution and enables software developers to identify the requirements, usage scenarios, main modules, interfaces, data and control objects that need to be developed.

The first V-model represents the process-level perspective and the second V-model represents the product-level perspective. The transition from the process- to the product-level perspectives are assured by transition steps and rules that enable an alignment between process- and product-level requirements, creating the V+V process model.

Our approach is adopted to create context for business software implementation teams in situations where requirements cannot be properly elicited. The V+V process is based on successive models construction and recursive derivation of logical architectures, to create the desired requirements representation. The first models are based on use cases, derived from high-level representations of the desired system interactions, called usage scenarios. These scenarios called *A*- and *B-type sequence diagrams* represent the intended flow of messages between use cases (in the *A-type sequence diagrams*) or architectural elements (in the *B-type sequence* diagrams) of the model. The *A-type sequence diagrams* represent intentions of sequences and the *B-type sequence diagrams* represent the implementation of the *A-type sequence diagrams*, making them an assessment tool of the initial requirements.

The V+V model approach assures that validation tasks are performed continuously along the modeling process. It allows for validating:

(i) the final software solution according to the initial expressed business requirements: This is done by implementing the *A-type sequence diagrams* in *B-type sequence diagrams* for the first V-model (process-level) and for the second V-model (product-level) and assuring the execution of the transition rules for one perspective for the other;

(ii) the *B-type* sequence diagrams according to *A-type* sequence diagrams: The implementation, by itself, of *A-type* into *B-type sequence diagrams* assures that the logical architecture (process- or product- level) encompasses all the intended scenarios and that all paths are executable;

(iii) the logical architectures by traversing it with *B-type* sequence diagrams: the logical architecture is validated regarding its degree of fulfillment of the initial scenarios based on the architectural elements and also their connections (associations).

The V+V model approach is supported on a set of transition steps and transition rules in order to execute the transition from process- to product-level perspective. These transition steps use as basis an information system logical architecture to output a product-level use case model. The product-level requirements are specified in a service-oriented logical architecture, having as basis the information system logical architecture. By adopting the approach, requirements for specifying services are properly aligned with organizational information system requirements in a traceable way.

The approach also uses software engineering techniques, such as operational model transformations to assure the execution of a process that begins with business needs and ends with a logical architectural representation of a system. It is a common fact that domain-specific needs, namely business needs, are a fast changing concern that must be tackled. Information system architectures must be in a way that potentially changing domain-specific needs are local in the architecture representation.

Our proposed V+V process encompasses the derivation of a logical architecture representation that is aligned with domain-specific needs and any change made to those domain-specific needs is reflected in the logical architectural model, and the transformation is properly assured. Since the *Mashed UC* model is derived from a model transformation based on mappings (from AEpc's to UCtr's), traceability between AEpc's and UCpt's is guaranteed, thus any necessary change on product-level requirements due to a change on a given business needs is easily identified and propagated alongside the models that comprise the V+V process.

We conducted a case study in the ISOFIN project throughout this work, in order to validate our proposed approach. The initial problem that arose in the project team, the definition of the business model, was solved by executing P.L.AC.E. to derive the organizational configurations and then, the problems of not being able to identify the requirements were tackled by applying the process-level to product-level transitions in the V+V process model. Using the constructed product-level logical architecture, the development teams were able to build the cloud-based ISOFIN platform that provides interoperability to financial institutions.

Each of the V-models support conducting reviews regarding architectural decisions, based on detecting issues on the derived models. The architectural decisions concern defining goals (quality attributes reviews or documentation for example) that are imposed to the *A-type* and *B-type sequence diagrams*. Models used for supporting documentation that describe the scenarios and the validations done can be delivered to the stakeholders, providing a better understanding of the intended system (Ferreira, Santos, Machado, & Gasevic, 2013). Regarding the quality attributes reviews they were not explicitly done in the ISOFIN project. Instead, they turned into requirements that were imbued in design decisions related to the logical architecture and thus, enforcing, by design, their fulfillment. Similar to injecting quality reviews to individual V-models, the V+V-Model is also able to conduct such review by merging both V-Models.

6.2 Synthesis of Research Efforts

The required efforts to establish a process able to define a representation of a system from a set of misunderstood and badly specified requirements should be understood by the scientific community with the purpose of fully realizing the advantages of having an architecture derivation method that supports the design decisions in the process. We began this thesis by introducing the problem that we were facing and the activity elicitation technique PL.AC.E that we used to create context for the Organizational Configuration definition. We framed the design decisions for intended resulting system within cloud-related paradigms and then described the V-Model, able to derive the information system logical architecture.

The process-level 4SRS method was introduced and detailed to provide information on the vertex of the V-Model. A vertex, as any mathematician might say, is the strongest part of any shape, and in the V-Model approach, the 4SRS vertex assures the transition between domains and artifacts.

Next we introduced the V+V-Model approach. This approach is composed by two V-Models, one on the process-level perspective (left side), able to derive an information system architecture and other, on the product-level perspective (right side), able to derive a service-oriented logical architecture.

Either the V-Model as the V+V-Model approaches were assessed using ARID, an architectural evaluation method, adapted to our approach. The adapted ARID method proved itself useful by adding extra information to the modeled artifacts and also by promoting the refinement through iterations of the 4SRS method of the logical architecture representation.

All the presented research efforts were validated using the ISOFIN project, a real industrial case study. All the techniques and approach were also applied in the project.

6.3 Synthesis of Scientific Results

The work carried out in this thesis partially shows that by beginning an analysis in the information systems perspective has potential to create a more robust system. There is a small number of projects and initiatives that currently use the V-Model approach to ensure an accurate definition of the requirements. From those, we would like to enhance:

- MSc Thesis "Requirements and Logical Architecture of an Information System to Manage Innovation", to be presented in 2013 at the University of Minho. This thesis uses the process-level V-Model to create a logical representation of the processes that should be implemented in an enterprise content management system for supporting the Portuguese Standard for Research, Innovation and Development (NP4457);

- Project AA4ALL (http://www.aal4all.org/). This project uses the V-Model approach for requirements elicitation, executed in order to derive a process-level logical architecture diagram for an Ambient Assisted Living (AAL) platform. The intended AAL platform allows for interoperability between AAL software solutions and encompasses four AAL life settings that were identified in a roadmap for ageing and ICT development.

- The presented ISOFIN Project (http://isofincloud.i2s.pt/)

Concerning scientific publications, we would like to identify the following, all accepted in renowned conferences:

- Nuno Ferreira, Ricardo J. Machado, Dragan Gaševic. An Ontological Approach to Model-Driven Software Product Line Development. Proceedings of the 4th International Conference on Software Engineering Advances - ICSEA'2009, Session on SEDES'2009 Workshop, September, 2009, pp. 559-564, IEEE

Computer Society Press, Los Alamitos, California, U.S.A., [ISBN: 978-0-7695-3777-15].

- Nuno Ferreira, Nuno Santos, Ricardo J. Machado, Dragan Gaševic. Derivation of Process-Oriented Logical Architectures: An Elicitation Approach for Cloud Design. Oscar Dieste, Andreas Jedlitschka, Natalia Juristo (Eds.), Product-Focused Software Process Improvement, pp. 45-58, LNCS Series vol. 7343, Springer-Verlag, Berlin Heidelberg, Germany, June, 2012, [ISSN: 0302-9743], [ISBN: 978-3-642-31063-8]. (Proceedings of the 13th International Conference on Product Focused Software Development and Process Improvement - PROFES'2012, Madrid, Spain, June, 2012).

- Nuno Ferreira, Nuno Santos, Pedro Soares, Ricardo J. Machado, Dragan Gaševic. Transition from Process- to Product-Level Perspective by Recursive Derivation of Logical Architectures for Business Software. Proceedings of the 6th IFIP International Conference on Research and Practical Issues of Enterprise Information Systems - CONFENIS'2012, Track on Enterprise System Design, Ghent, Belgium, September, 2012, LNBIP Series, Springer-Verlag, Berlin Heidelberg, Germany.

- Nuno Ferreira, Nuno Santos, Ricardo J. Machado, Dragan Gaševic. Aligning Domain-related Models for Creating Context for Software Product Design. Proceedings of the 5th Software Quality Days Conference - SWQD'2013, Scientific Track, LNBIP Series, Springer-Verlag, Berlin Heidelberg, Germany, Vienna, Austria, January, 2013.

- Nuno Ferreira, Nuno Santos, Pedro Soares, Ricardo J. Machado, Dragan Gaševic. A Demonstration Case on Steps and Rules for the Transition from Process-Level to Software Logical Architectures in Enterprise Models. In Janis Grabis, Marite Kirikova, Jelena Zdravkovic & Janis Stirna (Eds.), The Practice of Enterprise Modeling - POeM'2013 (Vol. 165, pp. 277-291): Springer Berlin Heidelberg, November, 2013.

- Nuno Ferreira, Nuno Santos, Ricardo J. Machado, José Miguel Fernandes, Dragan Gasević. (2014). A V-Model Approach for Business Process Requirements Elicitation in Cloud Design. In Athman Bouguettaya, Quan Z. Sheng & Florian Daniel (Eds.), Advanced Web Services (pp. 551-578): Springer New York.

We are expecting the results from our submissions to the following:

- Nuno Santos, Juliana Teixeira, António Pereira, Nuno Ferreira, Ana Lima, Ricardo Simões, Ricardo J. Machado. A Demonstration Case on the Derivation of Process-Level Logical Architectures for Ambient Assisted Living Ecosystems. Book chapter submitted for reviewing on the Ambient Assisted Living Book (Taylor and Francis / CRC Press (USA))

- Nuno Ferreira, Nuno Santos, Ricardo J. Machado, Dragan Gašević. From Process-level to Product-level Architectures in Information System Development. (Scopus indexed journal)

- Nuno Costa, Nuno Santos, Nuno Ferreira, Ricardo J. Machado. Delivering User Stories for Implementing Logical Software Architectures by Multiple Scrum Teams. (LNCS Springer-Verlag).

- Nuno Ferreira, Nuno Santos, Ricardo J. Machado, Dragan Gaševic. Modularization of Logical Software Architectures for Implementation with Multiple Teams. (LNCS Springer-Verlag).

Additionally we are preparing submissions regarding the future work to be presented in the next section.

6.4 Future Work

We are conscious that our work does not covers all the problems that we felt related to the requirements elicitation methods and architecture derivation. Along the years that we tackled such problems, others arose and we would like to point out the major issues that the research community could embrace:

- Detail the use case input to the 4SRS. Establish a direct relation between the use cases and the architectural elements.
- Map the product-level logical architecture to development teams, namely SCRUM teams, by providing the architecture diagram and a *combined multiple view*, made of the *B-Type* sequence diagrams, the logical architecture portion that must be developed and the major components and interfaces that must be respected. This is partially based on the analysis made in Figure 63, found on Annex B.
- Define a set of patterns that obey to the target deployment logic and inject them in the 4SRS method, to generate aggregations and associations in the method, fully compliant with the intended software product.
- Promote the process-level V-Model as a method of creating context also for existing product architectures. The derived logical architecture of the process-level V-Model has information that is used to configure existing systems.

The development of the V+V model approach opened a research topic that joined multiple research teams and lecturers. It is expected to see in the near future a more detailed and refined version of the V+V-Model approach.

References

Abran, A., Moore, J. W., Dupuis, R., Dupuis, R., & Tripp, L. L. (2001). Guide to the software engineering body of knowledge (SWEBOK). *2004 ed P Bourque R Dupuis A Abran and JW Moore Eds IEEE Press*.

Alter, S. (2002). The work system method for understanding information systems and information systems research. *Communications of the Association for Information Systems, 9*(1), 6.

Alter, S. (2008). Service system fundamentals: Work system, value chain, and life cycle. *IBM Systems Journal, 47*(1), 71-85.

Ambler, S., & Lines, M. (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*: IBM Press.

Atkinson, C., & Kuhne, T. (2003). Model-Driven Development: A Metamodeling Foundation. *IEEE Softw., 20*(5), 36-41.

Azevedo, S., Machado, R. J., Bragança, A., & Ribeiro, H. (2010). Systematic Use of Software Development Patterns through a Multilevel and Multistage Classification. *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, 304.

Azevedo, S., Machado, R. J., Muthig, D., & Ribeiro, H. (2009). *Refinement of Software Product Line Architectures through Recursive Modeling Techniques* Paper presented at the On the Move to Meaningful Internet Systems: OTM 2009 Workshops. http://dx.doi.org/10.1007/978-3-642-05290-3_53

BABOK. BABOK - Guide to Business Analysis Body of Knowledge Retrieved March 2011, from http://www.theiiba.org/AM/Template.cfm?Section=Body_of_Knowledge

Barrett, S., & Konsynski, B. (Dec., 1982). Inter-Organization Information Sharing Systems. *MIS Quarterly, 6*(Special Issue: [1982 Research Program of the Society for Management Information Systems]), 93-105

Basili, V. R., Lindvall, M., Regardie, M., Seaman, C., Heidrich, J., Munch, J., . . . Trendowicz, A. (2010). Linking Software Development and Business Strategy Through Measurement. *Computer, 43*(4), 57-65. doi: 10.1109/mc.2010.108

Bauer, B., Müller, J. P., & Roser, S. (2004). *A Model-driven Approach to Designing Cross-Enterprise Business Processes*. Paper presented at the MIOS Workshop in OTM Conference.

Bayer, J., Muthig, D., & Göpfert, B. (2001). The library system product line. A KobrA case study. *Fraunhofer IESE*.

Bensaou, M., & Venkatraman, N. (1993). Interorganizational relationships and information technology: A conceptual synthesis and a research framework. *European Journal of Information Systems, 5*, 84-91.

Bertolino, A., Inverardi, P., & Muccini, H. (2001). *An explorative journey from architectural tests definition down to code tests execution*. Paper presented at the Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada.

Bezivin, J., Dupé, G., Jouault, F., Pitette, G., & Rougui, J. E. (2003). *First experiments with the ATL model transformation language: Transforming XSLT into XQuery*. Paper presented at the 2nd OOPSLA Workshop on Generative Techniques in the context of MDA, Anaheim, CA, USA.

Bosch, J. (2000). *Design and use of software architectures: adopting and evolving a product-line approach*: ACM Press/Addison-Wesley Publishing Co.

Bragança, A., & Machado, R. (2009). A model-driven approach for the derivation of architectural requirements of software product lines. [10.1007/s11334-009-0078-3]. *Innovations in Systems and Software Engineering, 5*(1), 65-78.

Browning, T. R., & Eppinger, S. D. (2002). Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Trans on Engineering Management, 49*(4), 428-442.

Campbell, B. (2005). *Alignment: Resolving ambiguity within bounded choices*.

Campbell, B., Kay, R., & Avison, D. (2005). Strategic alignment: a practitioner's perspective. *Journal of Enterprise Information Management, 18*(6), 653-664.

Cardoso, E. C. S., Almeida, J. P. A., & Guizzardi, G. (2009). *Requirements engineering based on business process models: A case study*. Paper presented at the Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: the Tropos project. *Information Systems*.

Cerpa, N., & Verner, J. M. (2009). Why did your project fail? *Commun. ACM, 52*(12), 130-134. doi: 10.1145/1610252.1610286

Checkland, P. (1981). *Systems Thinking, Systems Practice*: John Wiley & Sons.

Checkland, P. (1985). Achieving 'Desirable and Feasible' Change: An Application of Soft Systems Methodology. *The Journal of the Operational Research Society, 36*(9), 821-831.

Checkland, P. (2000). Soft systems methodology: a thirty year retrospective. *Systems Research, 17*, S11-S58.

Chen, Y., & Tsai, W. (2010). Service-Oriented Computing and Web Data Management, Kendall: Hunt Publishing.

Clements, P., Garlan, D., Little, R., Nord, R., & Stafford, J. (2003). *Documenting software architectures: views and beyond*.

Clements, P. C. (2000). Active Reviews for Intermediate Designs.: Technical Note CMU/SEI-2000-TN-009.

CMMI Product Team. (2006). Capability Maturity Model Integration version 1.2 *CMMI for Development*.

Conradi, R., & Jaccheri, M. (1999). *Process Modelling Languages*. Paper presented at the Software Process: Principles, Methodology, and Technology. http://dx.doi.org/10.1007/3-540-49205-4_3

Davenport, T. H. (1993). *Process innovation: reengineering work through information technology*: Harvard Business Press.

Dijkman, R. M., & Joosten, S. M. M. (2002a). *An algorithm to derive use cases from business processes*. Paper presented at the SEA'02, Cambridge, MA, USA.

Dijkman, R. M., & Joosten, S. M. M. (2002b). Deriving use case diagrams from business process models. *Technical report, CTIT Tecnhical Report*.

EABOK. EABOK - Guide to the Enterprise Architecture Body of Knowledge Retrieved March 2011, from http://www.mitre.org/work/tech_papers/tech_papers_04/04_0104/04_0104.pdf

Evan, W. M. (1965). Toward a theory of inter-organizational relations. *Management Science*, 217-230.

Fernandes, J., Machado, R. J., Monteiro, P., & Rodrigues, H. (2006). A Demonstration Case on the Transformation of Software Architectures for Service Specification. In B. Kleinjohann, L. Kleinjohann, R. Machado, C. Pereira & P. Thiagarajan (Eds.), *From Model-Driven Design to Resource Management for Distributed Embedded Systems* (Vol. 225, pp. 235-244): Springer Boston.

Ferreira, N., Machado, R. J., & Gašević, D. (September, 2009). *An Ontology-based Approach to Model-Driven Software Product Lines*. Paper presented at the 4th International Conference on Software Engineering Advances - ICSEA 2009, Sessions of SEDES'2009 Workshop, Oporto, Portugal.

Ferreira, N., Santos, N., Machado, R. J., Fernandes, J. E., & Gasevic, D. (2013). *A V-Model Approach for Business Process Requirements Elicitation in Cloud Design*. Paper presented at the Web Services Handbook 2012

Ferreira, N., Santos, N., Machado, R. J., & Gasevic, D. (2012a). *Derivation of Process-Oriented Logical Architectures: An Elicitation Approach for Cloud Design*. Paper presented at the 13th International Conference on Product-Focused Software Development and Process Improvement - PROFES 2012, Madrid, Spain.

Ferreira, N., Santos, N., Machado, R. J., & Gasevic, D. (2012b). *Derivation of Process-Oriented Logical Architectures: An Elicitation Approach for Cloud Design*. Paper presented at the PROFES'12, Madrid, Spain.

Ferreira, N., Santos, N., Machado, R. J., & Gasevic, D. (2013). *Aligning Domain-related Models for Creating Context for Software Product Design*. Paper presented at the SWQD'13, Vienna, Austria.

Ferreira, N., Santos, N., Machado, R. J., & Gaševic, D. (2013). *Aligning Domain-related Models for Creating Context for Software Product Design.* Paper presented at the 5th Software Quality Days Conference - SWQD'2013, Vienna, Austria.

Ferreira, N., Santos, N., Soares, P., Machado, R., & Gašević, D. (2013). A Demonstration Case on Steps and Rules for the Transition from Process-Level to Software Logical Architectures in Enterprise Models. In J. Grabis, M. Kirikova, J. Zdravkovic & J. Stirna (Eds.), *The Practice of Enterprise Modeling* (Vol. 165, pp. 277-291): Springer Berlin Heidelberg.

Ferreira, N., Santos, N., Soares, P., Machado, R. J., & Gasevic, D. (2012). *Transition from Process- to Product-level Perspective for Business Software*. Paper presented at the Proceedings of the 6th IFIP International Conference on Research and Practical Issues of Enterprise Information Systems - CONFENIS'2012, Track on Enterprise System Design, Ghent, Belgium

Frakes, W., Prieto-Diaz, R., & Fox, C. (1998). DARE: Domain analysis and reuse environment. *Annals of Software Engineering, 5*, 125-141.

Furht, B., & Escalante, A. (2010). *Handbook of Cloud Computing*: Springer.

G2SEBoK. G2SEBoK - Guide to Systems Engineering Body of Knowledge Retrieved March 2011, from http://g2sebok.incose.org/

Gillett, S. E., & Kapor, M. (1996). The Self-governing Internet: Coordination by Design. Retrieved from http://ccs.mit.edu/papers/CCSWP197/CCSWP197.html

Haitham, S. H. (2011). *A Domain Analysis Method for Evolvable Software Product Line Architectures*.

Hammer, M. (1997). *Beyond reengineering: How the process-centered organization is changing our work and our lives*: Harper Paperbacks.

Hanisch, J., & Corbitt, B. (2007). Impediments to requirements engineering during global software development. *European Journal of Information Systems, 16*(6), 793-805. doi: 10.1057/palgrave.ejis.3000723

Haskins, C., & Forsberg, K. (2011). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities; INCOSE-TP-2003-002-03.2. 1*.

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems: Theory and Practice* (Vol. 22): Springer.

Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied software architecture*: Addison-Wesley Professional.

IDC. IDC Cloud Research Retrieved November 2012, from http://www.idc.com/prodserv/idc_cloud.jsp

IEEE Computer Society. (2000). IEEE Recommended Practice for Architectural Description of Software Intensive Systems - IEEE Std. 1471-2000.

IEEE Guide to Software Design Descriptions. (1993). *IEEE Std 1016.1-1993*, 0_1.

Information Technology Governance Institute (ITGI). (2012). *COBIT v5 - A Business Framework for the Governance and Management of Enterprise IT*: ISACA.

Instituto Português da Qualidade (IPQ). (2007). NP 4457: 2007 - Gestão da Investigação Desenvolvimento e Inovação (IDI). Requisitos do sistema de gestão da IDI.

International Organization for Standardization. (2008-11-25). ISO/IEC TR 15504-7:2008 - Process assessment - Part 7: Assessment of organizational maturity. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50519.

ISOFIN Consortium. (2010). ISOFIN Research Project, from http://isofincloud.i2s.pt

ISOFIN Project Consortium. (2010). ISOFIN Research Project, from http://isofincloud.i2s.pt

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software Reuse: Architecture, Process and Organization for Business Success*: Addison Wesley Longman.

Kaindl, H., & Falb, J. (2008). *Can We Transform Requirements into Architecture?* Paper presented at the ICSEA'08.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study: Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Sw Engineering*.

Kaplan, R. S., & Norton, D. P. (1992). The balanced scorecard–measures that drive performance. *Harvard business review, 70*(1), 71-79.

Kazman, R. (1996). *Tool support for architecture analysis and design*. Paper presented at the Joint proc. of the second intern. sw arch. workshop (ISAW-2) and intern. workshop on multiple perspectives in sw. dev. (Viewpoints '96) on SIGSOFT '96 workshops, San Francisco, California, United States.

Kim, J., Park, S., & Sugumaran, V. (2008). DRAMA: A framework for domain requirements analysis and modeling architectures in software product lines. *J. Syst. Softw., 81*(1), 37-55. doi: http://dx.doi.org/10.1016/j.jss.2007.04.011

Kruchten, P. (1995). The 4+1 View Model of Architecture. *IEEE Softw., 12*(6), 42-50. doi: 10.1109/52.469759

Krutz, R. L., & Vines, R. D. (2010). *Cloud Security - A Comprehensive Guide to Secure Cloud Computing*: Wiley.

Luftman, J., & Ben-Zvi, T. (2010). Key issues for IT executives 2010: judicious IT investments continue post-recession. *MIS Quarterly Executive, 9*(4), 263-273.

Machado, R., Lassen, K., Oliveira, S., Couto, M., & Pinto, P. (2007). Requirements Validation: Execution of UML Models with CPN Tools. *International Journal on Software Tools for Technology Transfer (STTT), 9*(3), 353-369. doi: 10.1007/s10009-007-0035-0

Machado, R. J., & Amaral, L. (Fevereiro, 2011). Sobre os Actos da Profissão no âmbito do Colégio de Engenharia Informática. *INFO – Revista Informativa da Ordem dos Engenheiros Região Norte*.

Machado, R. J., & Fernandes, J. (2002). Heterogeneous Information Systems Integration: Organizations and Methodologies. In M. Oivo & S. Komi-Sirviö (Eds.), *Product Focused Software Process Improvement* (Vol. 2559, pp. 629-643): Springer Berlin / Heidelberg.

Machado, R. J., Fernandes, J., Monteiro, P., & Rodrigues, H. (2006a). *Refinement of Software Architectures by Recursive Model Transformations*. Paper presented at the PROFES'06. http://dx.doi.org/10.1007/11767718_38

Machado, R. J., Fernandes, J., Monteiro, P., & Rodrigues, H. (2006b). *Refinement of Software Architectures by Recursive Model Transformations*. Paper presented at the Product-Focused Software Process Improvement. http://dx.doi.org/10.1007/11767718_38

Machado, R. J., Fernandes, J. M., Monteiro, P., & Rodrigues, H. (2005). *Transformation of UML Models for Service-Oriented Software Architectures*. Paper presented at the Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems.

Maibaum, T. (2006). On specifying systems that connect to the physical world. *New Trends in Software Methodologies, Tools and Techniques*.

Matinlassi, M., Niemelä, E., & Dobrica, L. (2002). Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture: VTT Tech. Research Centre of Finland.

Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on, 26*(1), 70-93.

Mell, P., & Grance, T. (2009). The NIST Definition of Cloud Computing.

Mintzberg, H. (1989). *Mintzberg on Management - Inside our strange world of organizations*

Monteiro, P., Machado, R. J., & Kazman, R. (2009). *Inception of Software Validation and Verification Practices within CMMI Level 2*.

Neighbors, J. M. (1980). *Software construction using components.* PhD, University of California, Irvine

NIST. (2009). National Institute of Standards and Technology - The NIST Definition of Cloud Computing Retrieved January 2011, from http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf

OMG. (2003a). MDA Guide v1.0.1 Retrieved February 2009, 2008, from http://www.omg.org

OMG. (2003b). MDA Guide Version 1.0.1: OMG Std.

OMG. (2010). Business Motivation Model (BMM) v1.1, from http://www.omg.org/spec/BMM/1.1/

OMG. (2011a). Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), from http://www.omg.org/spec/QVT/1.1

OMG. (2011b). Unified Modeling Language (UML) Superstructure Version 2.4.1 Retrieved January 2012, from http://www.omg.org/spec/UML/2.4.1/

Project Management Institute. (2008). *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)* (4th ed.).

Reese, G. (2009). *Cloud Application Architectures*: O'Reilly.

Roberto, C. (2010). Cloud Computing: Oportunidades, Drivers de Sucesso & Cases Study PT Virtual Desktop. *Semana Informática*.

Ruparelia, N. B. (2010). Software Development Lifecycle Models. *SIGSOFT Softw. Eng. Notes, 35*(3), 8-13. doi: 10.1145/1764810.1764814

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum*: Prentice Hall PTR.

Selic, B. (2003). The pragmatics of model-driven development. *Software, IEEE, 20*(5), 19-25. doi: 10.1109/ms.2003.1231146

Simos, M., Creps, D., Klinger, C., Levine, L., & Allemang, D. (14 June 1996). Organization Domain Modeling (ODM) Guidebook, Version 2.0. . *Informal Technical Report for STARS, STARS-VC-A025/001/00*.

Simos, M. A. (1995). Organization domain modeling (ODM): formalizing the core domain modeling life cycle. *SIGSOFT Softw. Eng. Notes, 20*(SI), 196-205. doi: 10.1145/223427.211845

Sungwon, K., & Yoonseok, C. (2005). *Designing logical architectures of software systems*. Paper presented at the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. .

The Open Group. (2009). *TOGAF Version 9 - The Open Group Architecture Framework (TOGAF)*.

Vaishnavi, V. K., & Jr., W. K. (2008). *Design Science Research Methods and Patterns Innovating Information and Communication Technology*: Auerbach Publications.

Vargo, S. L., & Lusch, R. F. (2004). The four service marketing myths. *Journal of Service Research, 6*(4), 324.

Velte, A. T., Velte, T. J., & Elsenpeter, R. (2010). *Cloud Computing: A Practical Approach*: McGraw-Hill.

Weiss, D. (1998). Commonality Analysis: A Systematic Process for Defining Families

Development and Evolution of Software Architectures for Product Families. In F. van der Linden (Ed.), (Vol. 1429, pp. 214-222): Springer Berlin / Heidelberg.

Weiss, D. M., & Lai, C. T. R. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*: Addison-Wesley Professional.

Weiss, J. (2010). What Insurers Need to Know to Develop a Cloud Computing Strategy. *Insurance & Technology*. Retrieved from http://www.insurancetech.com/architecture-infrastructure/225800122

Winter, R., & Fischer, R. (2006a). *Essential Layers, Artifacts, and Dependencies of Enterprise Architecture*. Paper presented at the EDOCW'06.

Winter, R., & Fischer, R. (2006b). *Essential Layers, Artifacts, and Dependencies of Enterprise Architecture*. Paper presented at the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW).

Yin, R. K. (2003). *Case Study Research - Design Methods* (3rd ed.): SAGE Publications.

Yipeng, J., Hailong, S., Xudong, L., Jin, Z., & Shangda, B. (2009). *A Decentralized Framework for Executing Composite Services Based on BPMN*. Paper presented at the COMPUTATIONWORLD '09.

Yourdon Inc. (1993). *Yourdon systems method: model-driven systems development*: Prentice Hall International Editions.

Yue, T., Briand, L. C., & Labiche, Y. (2011). A Systematic Review of Transformation Approaches between User Requirements and Analysis Models. *Requirements Engineering, Vol.16*(Issue 2).

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Syst. J., 26*(3), 276-292.

Zou, J., & Pavlovski, C. J. (2006). *Modeling Architectural Non Functional Requirements: From Use Case to Control Case*. Paper presented at the e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on.

Zowghi, D., & Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches, and tools. *Engineering and managing software requirements, Springer, Heidelberg*, 19-46.

Appendix A

This annex presents the initial use case context for the process-level V-Model and the evolution of the process-level 4SRS logical architecture through iterations #1 to #4. We also present a view of the logical architecture packages associated with actors. This view, in Figure 59, allows having an understanding of the interactions that the packages will have with the actors.

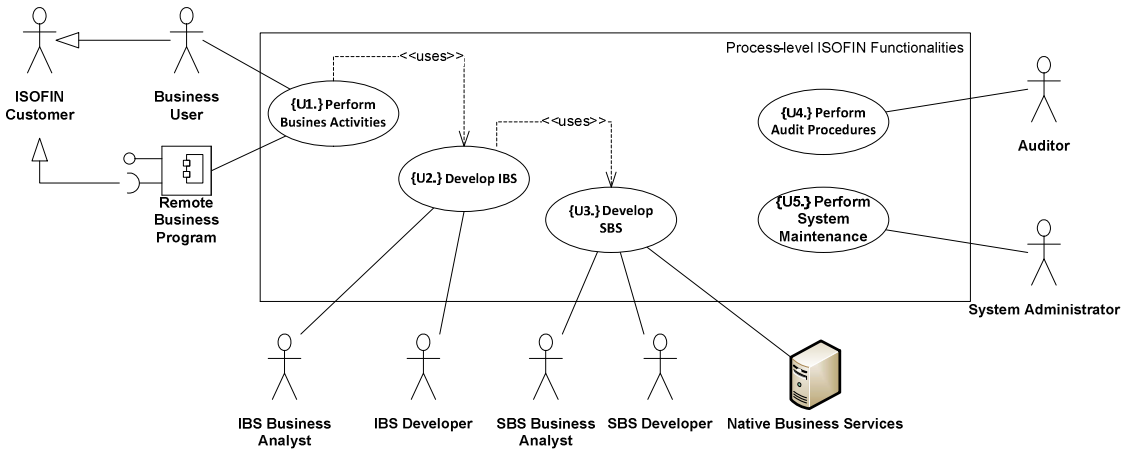

Figure 54: Process-level ISOFIN functionalities

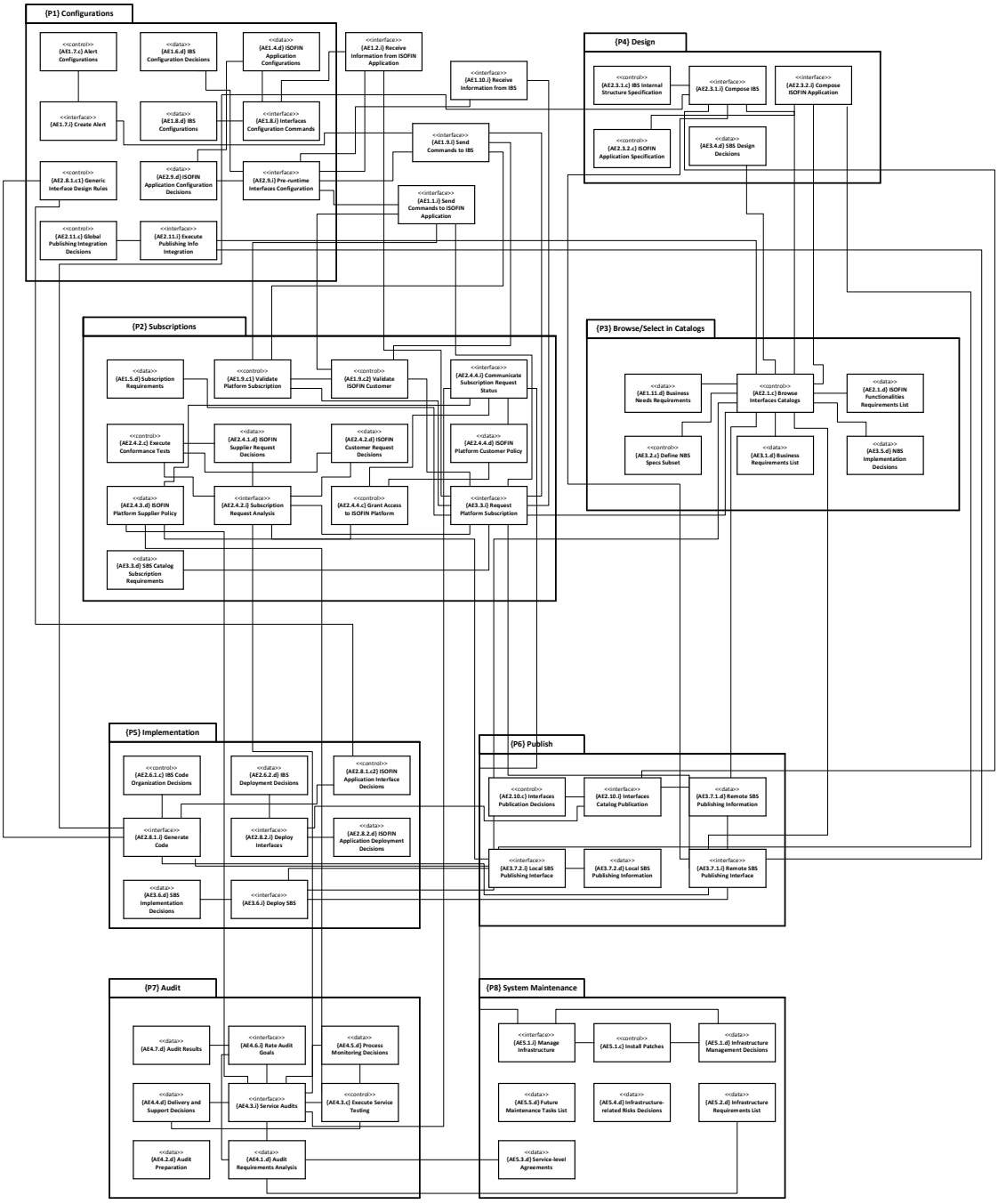

Figure 55: Process-level 4SRS iteration #1

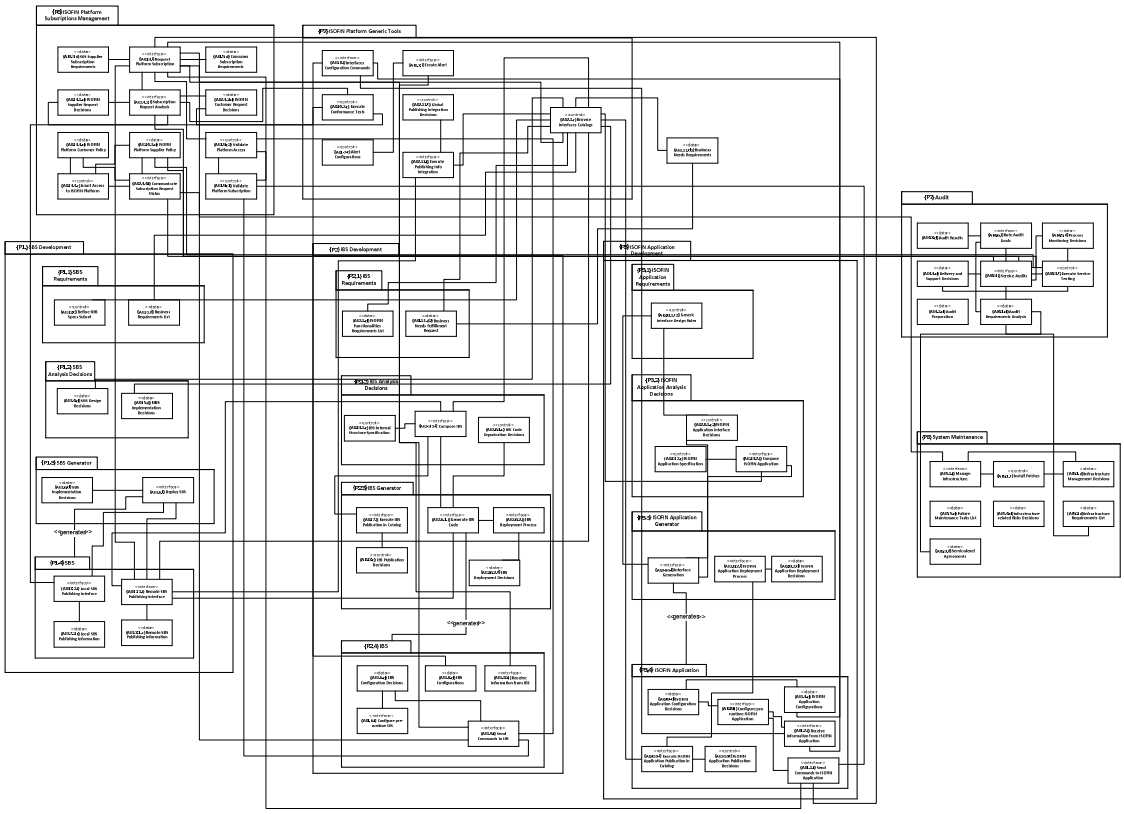

Figure 56: Process-level 4SRS iteration #2

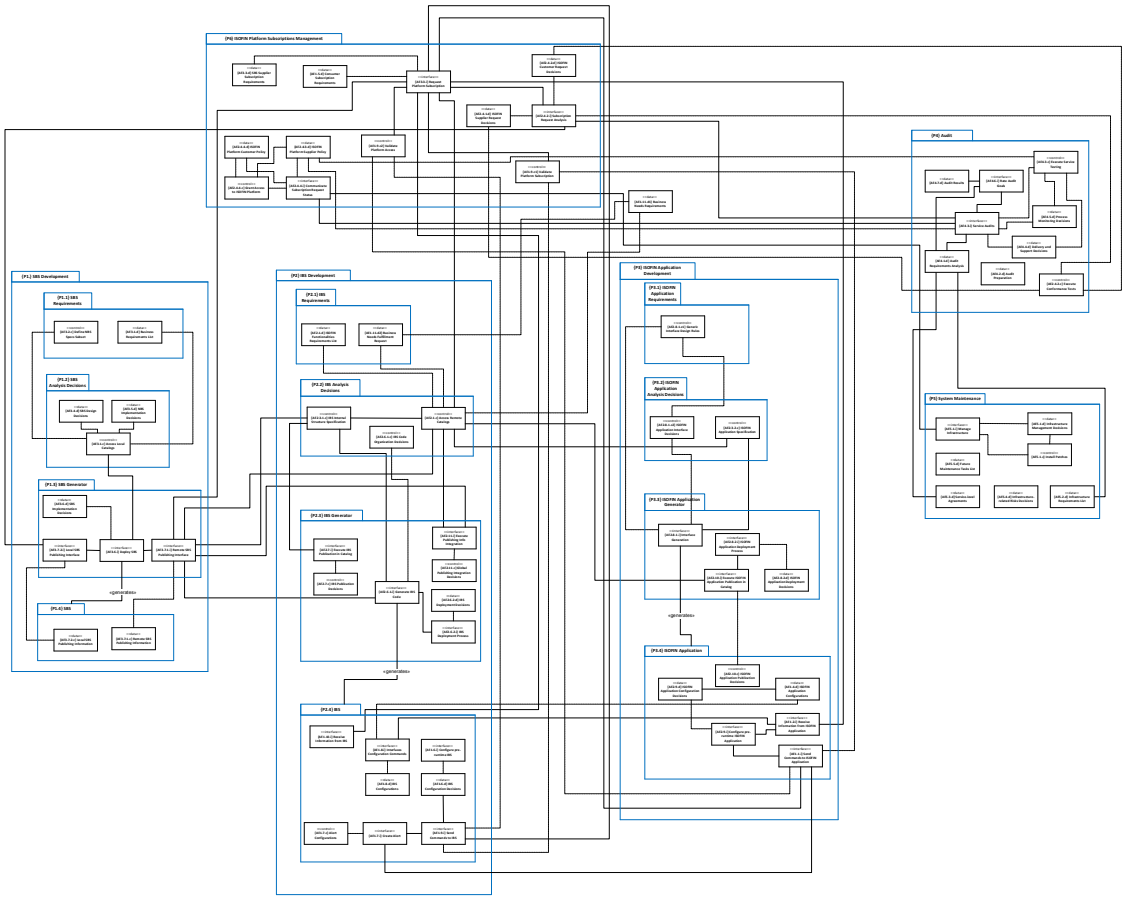

Figure 57:Process-level 4SRS iteration #3

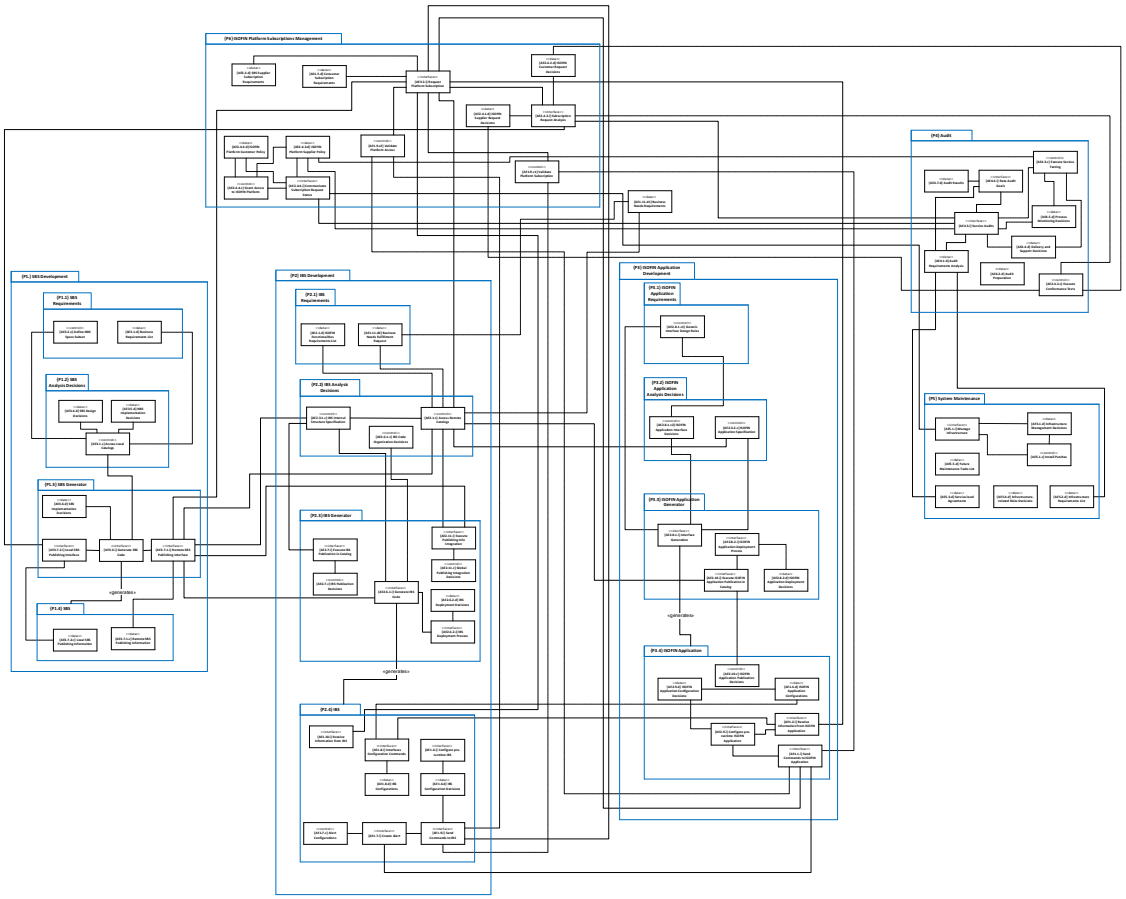

Figure 58: Process-level 4SRS iteration #4

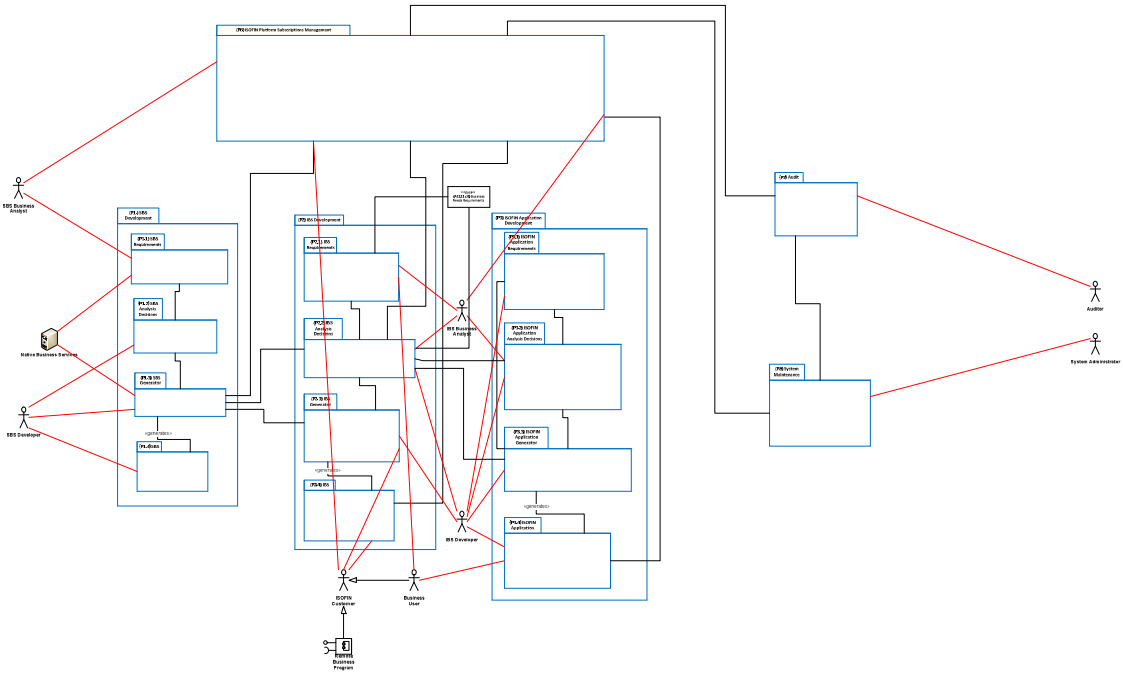

Figure 59: Logical Packages with Actors

Appendix B

This annex presents the initial use case context for the product-level V-Model and the evolution of the product-level 4SRS logical architecture through iterations #1 to #3 and also the full representation of the main software products that can be extracted from the logical architecture – see Figure 63: Product-level Logical Architecture Main Products. This main product representation allows to identifying the architectural elements that make up the intended software solution and overlaps regarding architectural elements.

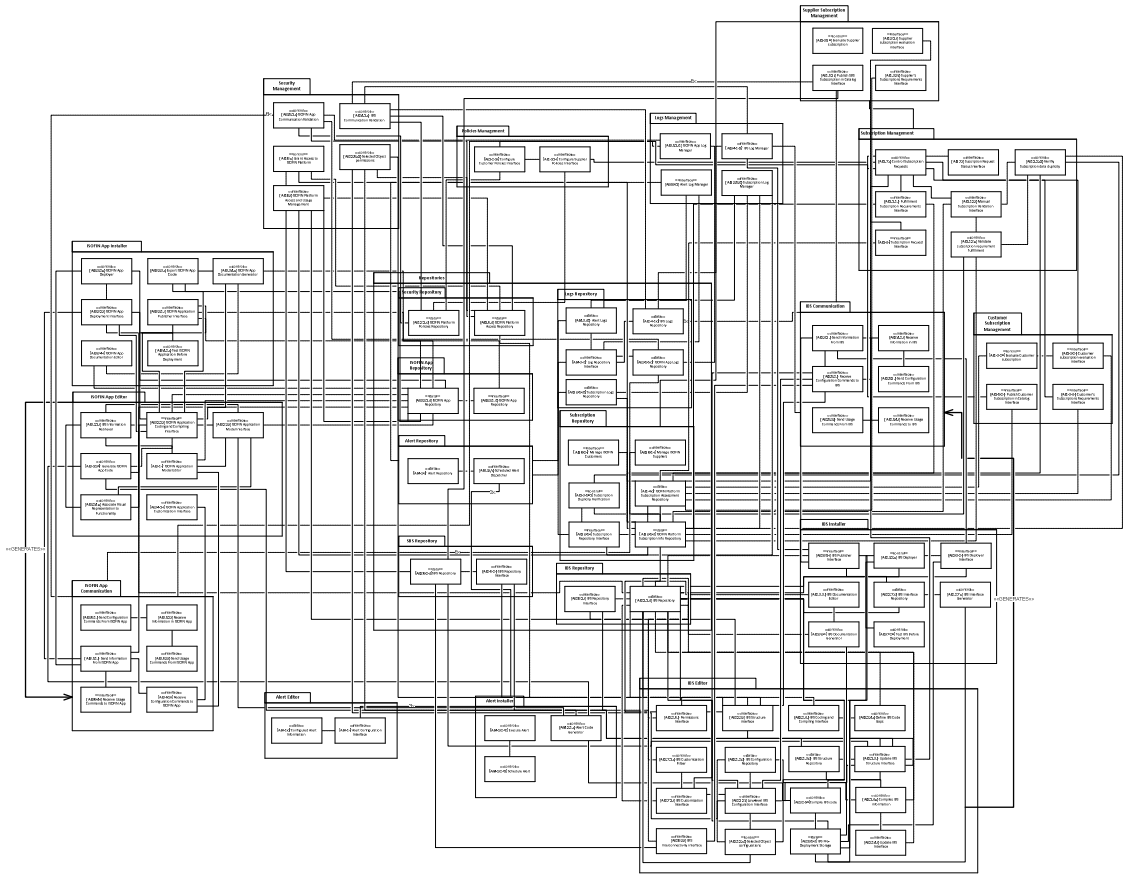

Figure 60: Product-level 4SRS iteration #1

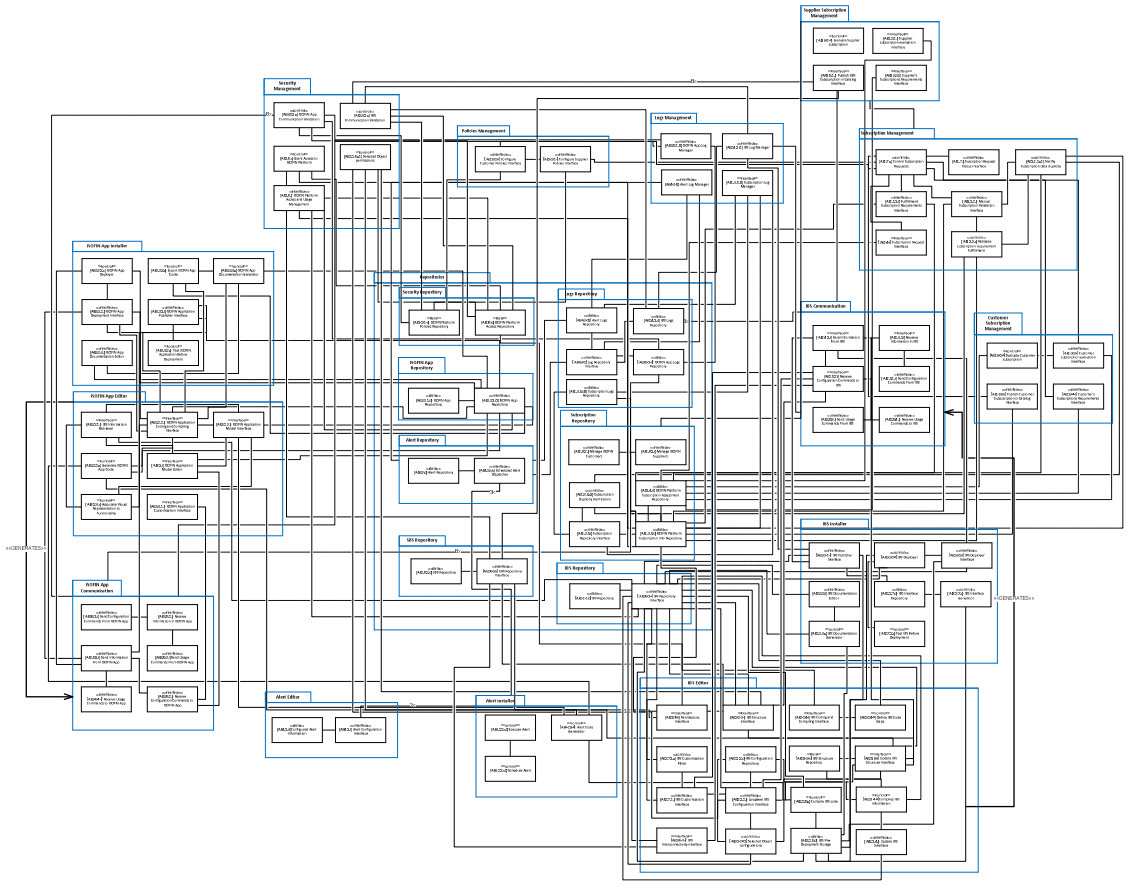

Figure 61: Product-level 4SRS iteration #2

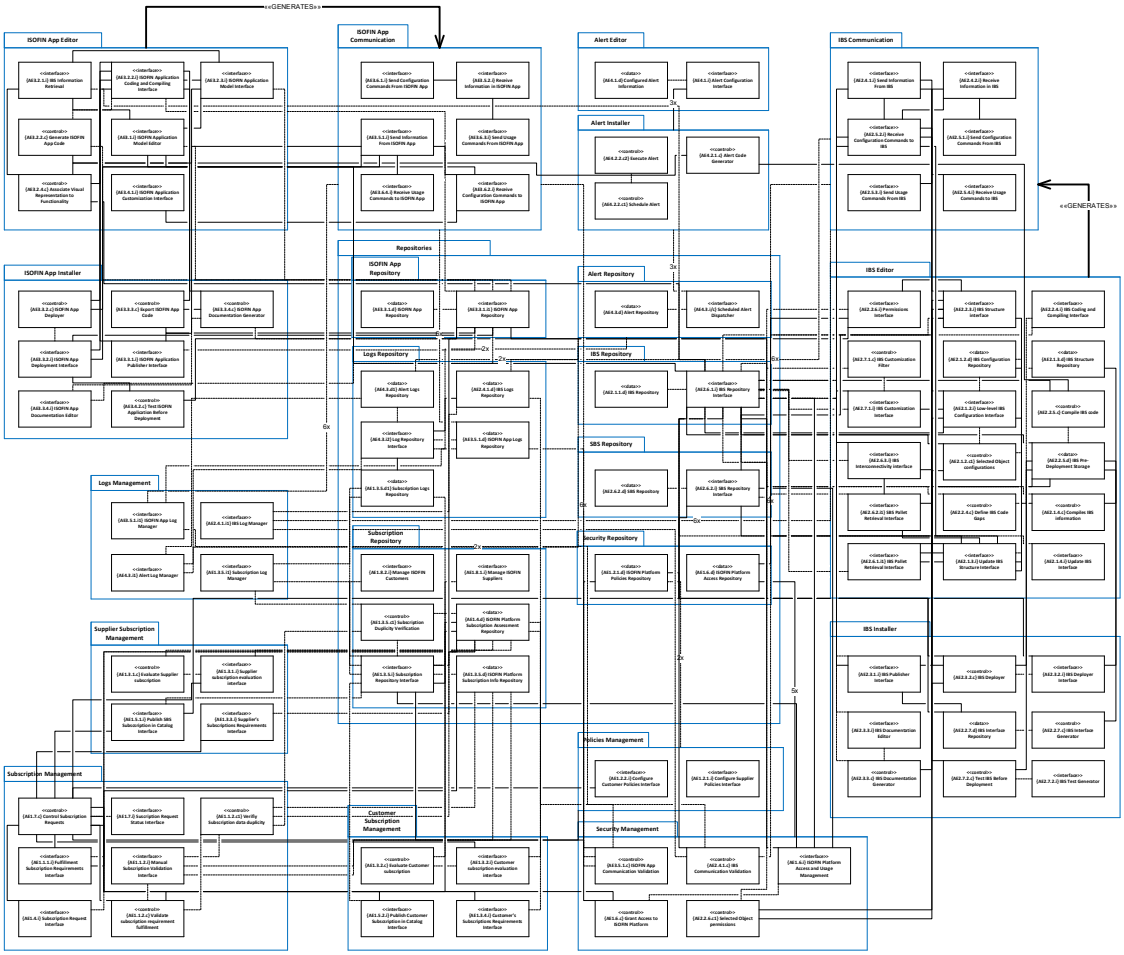

Figure 62: Product-level 4SRS iteration #3

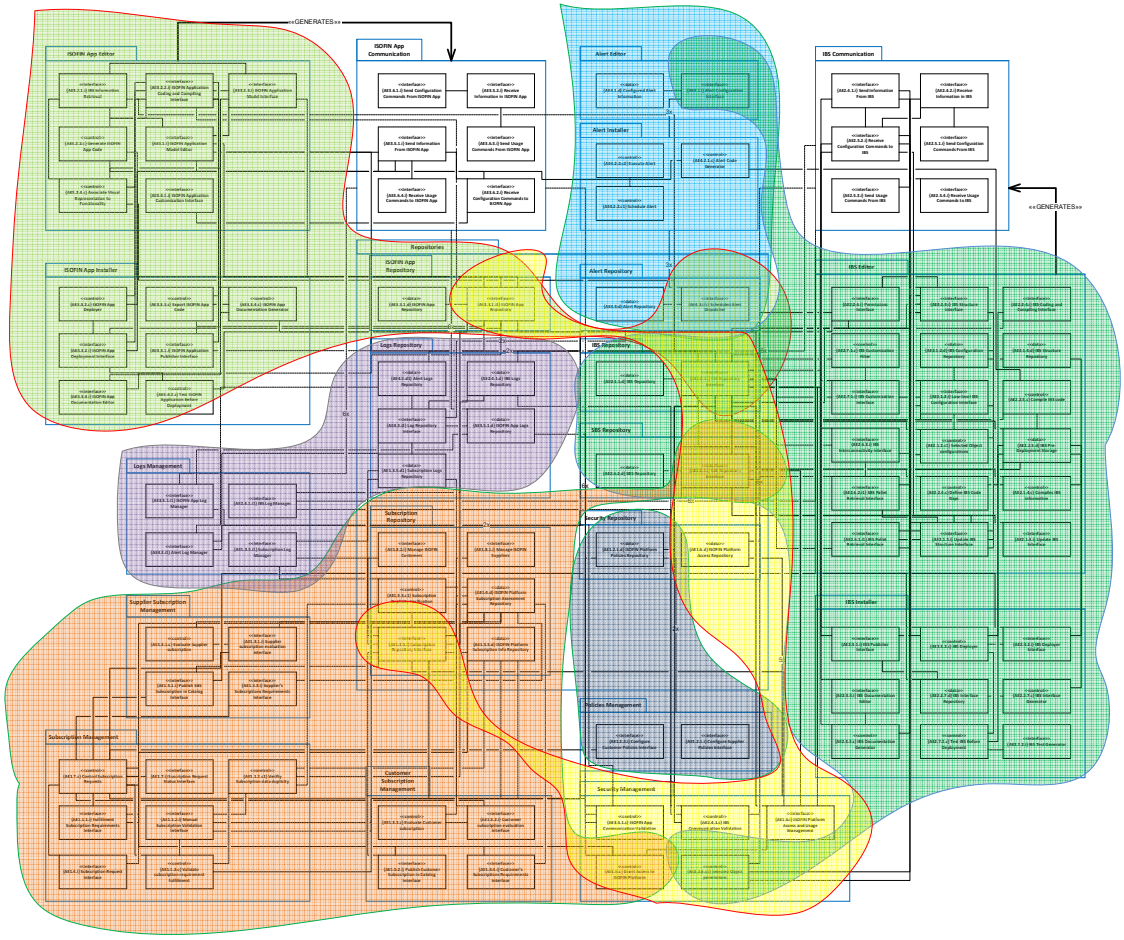

Figure 63: Product-level Logical Architecture Main Products