Derivation of Process-Oriented Logical Architectures: An Elicitation Approach for Cloud Design*

Nuno Ferreira[1], Nuno Santos[2], Ricardo J. Machado[3], and Dragan Gašević[4]

[1] I2S Informática, Sistemas e Serviços S.A., Porto, Portugal
[2] CCG - Centro de Computação Gráfica, Campus de Azurém, Guimarães, Portugal
[3] Centro ALGORITMI, Escola de Engenharia, Universidade do Minho, Guimarães, Portugal
[4] School of Computing and Information Systems, Athabasca University, Canada

Abstract. The benefits of cloud computing approaches are well known but designing logical architectures for that context can be complicated. Prior to designing a logical architecture, a proper requirements elicitation must be executed. When requirements are not properly elicited, and there are insufficient inputs for a product approach to requirements elicitation, a process-level perspective is an alternative way for achieving the intended base requirements for the logical design. Our proposed solution regards the adaptation and extension of the 4SRS (Four-Step-Rule-Set) method to derive logical architectural models, in a process-level perspective. This perspective creates context for the product-level requirements elicitation conducing to cloud design. We present a real industrial case where the method was applied and assessed. The method application results in the creation of a validated architectural model and in the uncovering of hidden requirements for the intended cloud design.

Keywords: Requirements Elicitation, Logical Architectures, Application Architectures, Development Methods for Cloud Applications.

1 Introduction

The design of software architectures for systems to be executed in a cloud computing environment brings many difficulties to system architects. Instead of designing a cloud computing architecture based on user requirements traditionally defined in a product-level perspective, we propose the use of a process-level perspective for the requirements definition and design of the logical model of the 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.

The term *process*, in a generic context, is hard to define. In the definition given in [1], a process is a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs. Software architecture deals with the design and implementation of the high-level structure of the software [2].

* This work has been supported by project ISOFIN (QREN 2010/013837).

This paper describes the extensions introduced into the 4SRS method to be adopted at the process-level perspective in large-scale projects. The 4SRS method was first defined and detailed in [3, 4]. 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. This paper additionally illustrates the applicability of the proposed approach in a real industrial case: the ISOFIN project (Interoperability in Financial Software). This project aims to deliver a set of cloud-based functionalities enacting the coordination of independent services relying on private clouds. The resulting ISOFIN platform will allow the semantic and application interoperability between enrolled financial institutions (Banks, Insurance Companies and others). 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 [5]. The transformation of such context into product-level requirements does not belong to the scope of the present work.

This paper is structured as follows: section 2 describes the problem associated to the real industrial case study which is presented in this paper, as well as some related work concerning the core topics and the reason for the new approach; section 3 presents the main differences between the traditional approach of the 4SRS method and the proposed process-level approach; section 4 presents the designed logical architecture as context for elicitation; and in section 5, we present our conclusions and some future work.

2 Problem Overview

This work 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 [6] 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 [7]. A proper requirements elicitation task must encompass an understanding of the organizational environment, through their business processes [8].

An accurate requirements elicitation can be assured through the use of requirements elicitation methodologies, methods or techniques. The Work System Method [9] 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) [10] 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. There are several approaches to support the design of software architectures, in a product-level perspective, like RSEB [11], FAST [12], FORM [13], KobrA [14] and QADA [15]. The product-level perspective of the 4SRS [4] method also promotes functional decomposition of software systems.

Tropos [16] and 4SRS (in [17]) 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 [17]. 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 [18], 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 [19]. A software process is composed of a set of activities related to the software development life-cycle. 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 [20]. The process architecture represents the fundamental organization of service development, service creation, and service distribution in the relevant enterprise context [21]. A process architecture can also be defined as an arrangement of the activities and their interfaces in a process [22], takes into account some non-functional requirements, such as performance and availability [2], and can be represented with components, connectors, systems/configurations of components and connectors, ports, roles, representations and rep-maps [23], as well as by architectural elements' static and temporal features [24]. 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 paper 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.

2.1 The ISOFIN Project

The logical process-level architecture of the ISOFIN solution [25] 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 paper only presents a subset of the proposed process-level architecture related to the customer perspective, as seen in Fig. 1. 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.

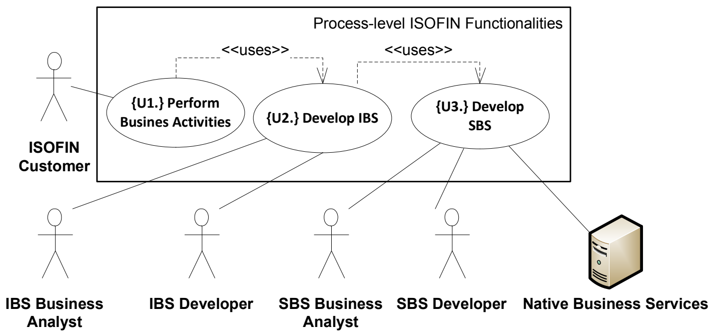

Fig. 1. Use Case Model Regarding the ISOFIN Process-level Perspective Functionalities

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.

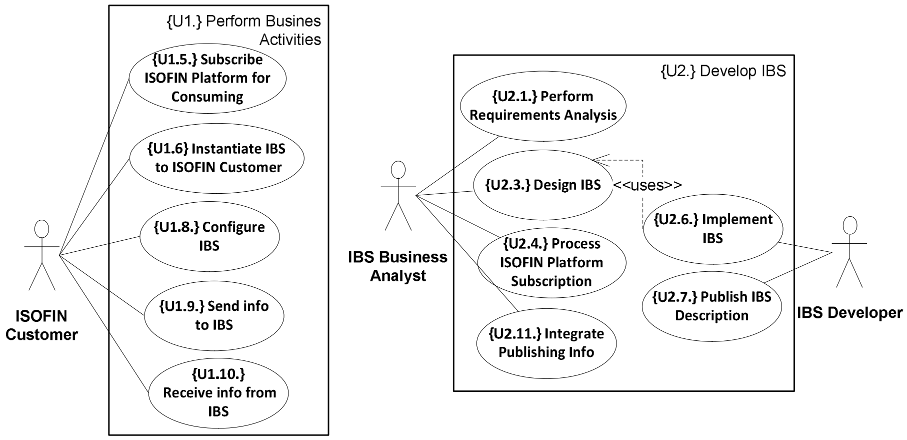
{U1.} Perform Busines Activities
{U1.5.} Subscribe ISOFIN Platform for Consuming
{U1.6} Instantiate IBS to ISOFIN Customer
{U1.8.} Configure IBS
{U1.9.} Send info to IBS
{U1.10.} Receive info from IBS
ISOFIN Customer
{U2.} Develop IBS
{U2.1.} Perform Requirements Analysis
{U2.3.} Design IBS
<<uses>>
{U2.6.} Implement IBS
{U2.4.} Process ISOFIN Platform Subscription
{U2.7.} Publish IBS Description
{U2.11.} Integrate Publishing Info
IBS Business Analyst
IBS Developer

Fig. 2. Refinement of Use Case 1 and Use Case 2 (subset)

In Fig. 2 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.

3 Process-Level 4SRS as an Elicitation Method for Cloud Design

The 4SRS method allows for the transformation of user requirements into an architectural model representation. This paper presents an extension of the traditional (product-level perspective) usage of the 4SRS method (presented in [4]) 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 trough 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 [4]. 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 1 to 5). In the real context, we manipulate the entire table 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 [4]).

3.1 Step 1: Architectural Element Creation

This step regards the creation of AEs. The product-level 4SRS [4] 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 1.

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 1. 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**

3.2 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 2 (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 2 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 2), 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 2. Micro-steps 2i trough 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 2. 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 3 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 3) 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 3), 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 3) 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 approach to the product-level approach, 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 3. Micro-steps 2v trough 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.

3.3 Step 3: Packaging and Aggregation

Like in the traditional 4SRS method, in this step (2nd column in Table 4), 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 4. 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

3.4 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 to easily identify 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 5) 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 5.

Table 5. 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}.

4 The ISOFIN Process-Level Logical Architecture

The ISOFIN project [25] is executed in a consortium comprising eight entities (private companies, public research centers and universities). 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, 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 is sustainably using 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 size limitation for this paper and also to the diagram's complexity, we only present a subset in Fig. 3. 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).

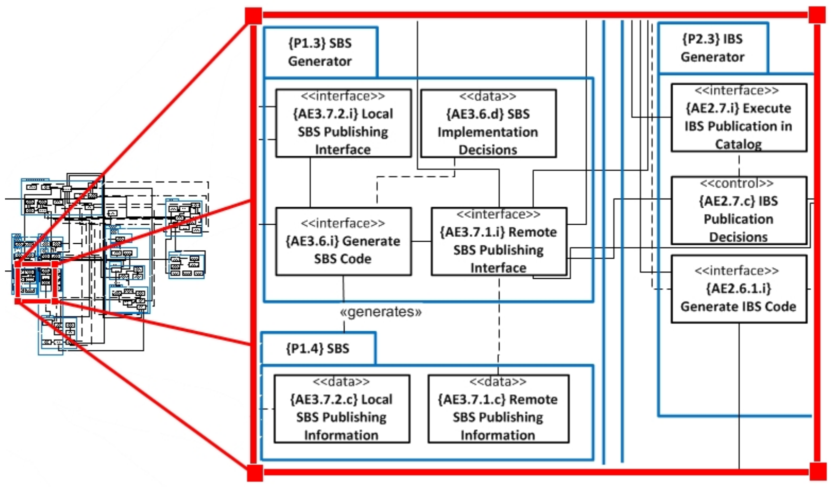
{P1.3} SBS Generator
<<interface>> {AE3.7.2.i} Local SBS Publishing Interface
<<data>> {AE3.6.d} SBS Implementation Decisions
<<interface>> {AE3.6.i} Generate SBS Code
<<interface>> {AE3.7.1.i} Remote SBS Publishing Interface
«generates»
{P1.4} SBS
<<data>> {AE3.7.2.c} Local SBS Publishing Information
<<data>> {AE3.7.1.c} Remote SBS Publishing Information
{P2.3} IBS Generator
<<interface>> {AE2.7.i} Execute IBS Publication in Catalog
<<control>> {AE2.7.c} IBS Publication Decisions
<<interface>> {AE2.6.1.i} Generate IBS Code

Fig. 3. 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.

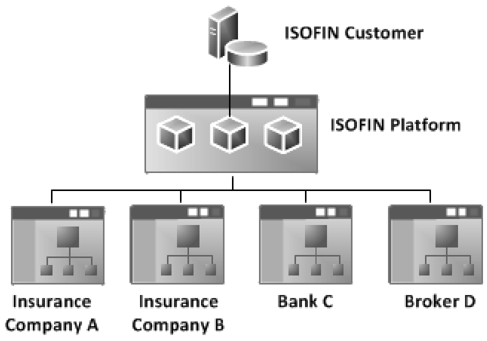

Fig. 4. Interoperability in ISOFIN

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.

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 (Fig. 3), Fig. 4 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.

5 Conclusion and Outlook

This paper presents the extensions 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 they 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 the information is understood by all involved. The process-level perspective allowed us to overcome difficulties when adopting a product-level perspective.

On the other hand, the manual execution of the method is prone to errors and very time consuming. Also, by adopting first the process-level perspective instead of the product-level perspective, time for delivering documentation to implementation teams increased. These are opportunities for improvement. We will address these drawbacks as future work. Additionally, we plan to study the required transformations to support the evolution of the process-level logical architecture into a product-level logical architecture that is needed to formally start the design phase of the cloud solution. We will also incorporate traceability features between process requirements, process-level logical architectures, product requirements and product-level logical architectures.

References

1. Davenport, T.H.: Process innovation: reengineering work through information technology. Harvard Business Press (1993)
2. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Softw. 12, 42–50 (1995)
3. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of UML Models for Service-Oriented Software Architectures. In: Proceedings of the 12th IEEE ECBS, pp. 173–182. IEEE Computer Society (2005)
4. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Refinement of Software Architectures by Recursive Model Transformations. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 422–428. Springer, Heidelberg (2006)
5. National Institute of Standards and Technology, `http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf`

6. Abran, A., Moore, J.W., Dupuis, R., Dupuis, R., Tripp, L.L.: In: Bourque, P., Dupuis, R., Abran, A., Moore, J.W. (eds.) Guide to the Software Engineering Body of Knowledge (SWEBOK). IEEE Press (2001, 2004)
7. Zowghi, D., Coulin, C.: Requirements elicitation: A survey of techniques, approaches, and tools. In: Engineering and Managing Software Requirements, pp. 19–46. Springer, Heidelberg (2005)
8. Cardoso, E.C.S., Almeida, J.P.A., Guizzardi, G.: Requirements engineering based on business process models: A case study. In: 13th Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, pp. 320–327 (2009)
9. Alter, S.: The work system method for understanding information systems and information systems research. Communications of the Association for Information Systems 9, 6 (2002)
10. Checkland, P.: Soft systems methodology: a thirty year retrospective. Systems Research 17, S11–S58 (2000)
11. Jacobson, I., Griss, M., Jonsson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison Wesley Longman (1997)
12. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional (1999)
13. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Sw Engineering (1998)
14. Bayer, J., Muthig, D., Göpfert, B.: The library system product line. A KobrA case study. Fraunhofer IESE (2001)
15. Matinlassi, M., Niemelä, E., Dobrica, L.: Quality-driven architecture design and quality analysis method. In: A Revolutionary Initiation Approach to a Product Line Architecture. VTT Tech, Research Centre of Finland (2002)
16. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems (2002)
17. Machado, R.J., Fernandes, J.: Heterogeneous Information Systems Integration: Organizations and Methodologies. In: Oivo, M., Komi-Sirviö, S. (eds.) PROFES 2002. LNCS, vol. 2559, pp. 629–643. Springer, Heidelberg (2002)
18. Hammer, M.: Beyond reengineering: How the process-centered organization is changing our work and our lives. Harper Paperbacks (1997)
19. Conradi, R., Jaccheri, M.L.: Process Modelling Languages. In: Derniame, J.-C., Kaba, B.A., Wastell, D. (eds.) Promoter-2 1998. LNCS, vol. 1500, pp. 27–52. Springer, Heidelberg (1999)
20. Azevedo, S., Machado, R.J., Muthig, D., Ribeiro, H.: Refinement of Software Product Line Architectures through Recursive Modeling Techniques. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 411–422. Springer, Heidelberg (2009)
21. Winter, R., Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. In: 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), p. 30 (2006)
22. Browning, T.R., Eppinger, S.D.: Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans. on Eng. Management 49, 428–442 (2002)
23. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering 26, 70–93 (2000)
24. Kazman, R.: Tool support for architecture analysis and design. In: Sw Arch. Workshop (ISAW-2) and Intern. Workshop on Multiple Perspectives in Sw. Dev (Viewpoints 1996) on SIGSOFT 1996 Workshops, pp. 94–97. ACM, San Francisco (1996)
25. ISOFIN Research Project, `http://isofincloud.i2s.pt`