Aligning Domain-related Models for Creating Context for Software Product 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
nuno.ferreira@i2s.pt
[2] CCG - Centro de Computação Gráfica, Campus de Azurém, Guimarães, Portugal
nuno.santos@ccg.pt
[3] Centro ALGORITMI, Escola de Engenharia, Universidade do Minho, Guimarães, Portugal
rmac@dsi.uminho.pt
[4] School of Computing and Information Systems, Athabasca University, Canada
dgasevic@acm.org

Abstract. A typical software product is developed so that it can fulfill the defined business domain, based on a proper product design context. Although, assuring an alignment between the technological developments with the business domain is a demanding task. With the purpose of clarifying the relations between the models that support the business and the software representations, we present in this paper a V-Model based approach to align the business domain requirements with the context for product design. This V-Model encompasses the models that support the initial definition of the project goals, expressed through organizational configurations, and the analysis and design of models that result in a process-level perspective of the system's logical architecture. Our approach adopts a process-level perspective with the intent to create context for product-level requirement elicitation. We present a case study as a demonstration and assessment of the applicability of our approach. Since the case study is extremely complex, we illustrate how to use the ARID method to evaluate the obtained process-level architecture.

Keywords: Software Engineering; Requirements Engineering; Model Alignment; Logical Architectures.

1 Introduction

One of the top concerns of information technologies (IT) managers for almost thirty years relates to software and the business domain alignment [1]. The importance of aligning the 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 [2]. 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 [3]. 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 premises that there is no clearly defined context for eliciting product requirements within a given specific domain. We propose a "Vee" Model-based adaptation (V Model) [4], 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.

This paper is structured as follows: section 2 presents the related work associated with our work; section 3 presents our V-Model representation to promote domain and software; section 4 includes a case study and details the pertinence of using the chosen presented models for creating context to product design. It also explains 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; section 6 presents our conclusions and future work.

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. 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, 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) [5]. 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) [6]. 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 [7], 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 [8, 9]. 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.* [10], Hofmeister *et al.* [11] and Krutchen [9]. 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 [12] 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 [13], 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 [14].

The process architecture represents a fundamental organization of service development, service creation, and service distribution in the relevant enterprise context [15]. 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 [16], FORM [17] or KobrA [18]. These all relate to the product-level perspective. In a process-level perspective, Tropos [19] 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 [20], but it also supports a process-level perspective [21, 22]. 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 [9].

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 [23], 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.* [3], 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 paper 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 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) [24], *A-Type* and *B-Type* Sequence Diagrams [25], use cases and process-level logical architecture diagrams. All these models are briefly described in this section and properly exemplified in the case study section of this paper, where more detail is given on how to derive a model from the previous models.

The OC model is a high-level representation of the activities (interactions) that exist between the business-level entities of a given domain. Fig. 1 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.

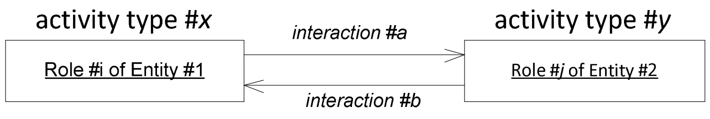

Fig. 1. Organizational Configuration

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 paper 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 Fig. 2.

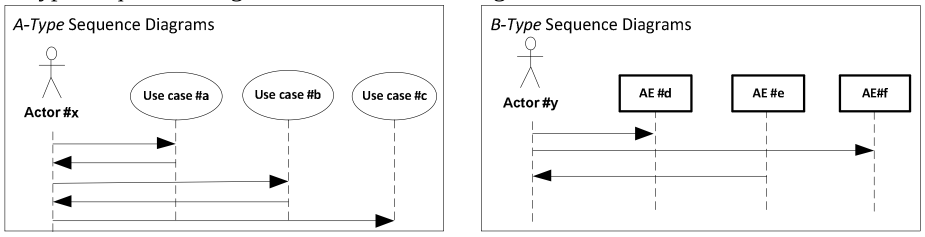

Fig. 2. *A-* and *B-Type* Sequence Diagrams

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 on Fig. 4. The V-Model representation [4] 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 [22] 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 [20] and the process-level perspective in [21, 22]. 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 [20-22] 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.

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 automatically create 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 to represent aggregations (of, for instance, existing legacy systems); and Step 4 (architectural element association) whose goal is to represent associations between the remaining architectural elements.

According with 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 Fig. 3. 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 it makes possible to adjust the results of the transformation to changing requirements. Tabular transformations are thoroughly described in [22, 26].

As suggested by the V-Model represented in Fig. 4, 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).

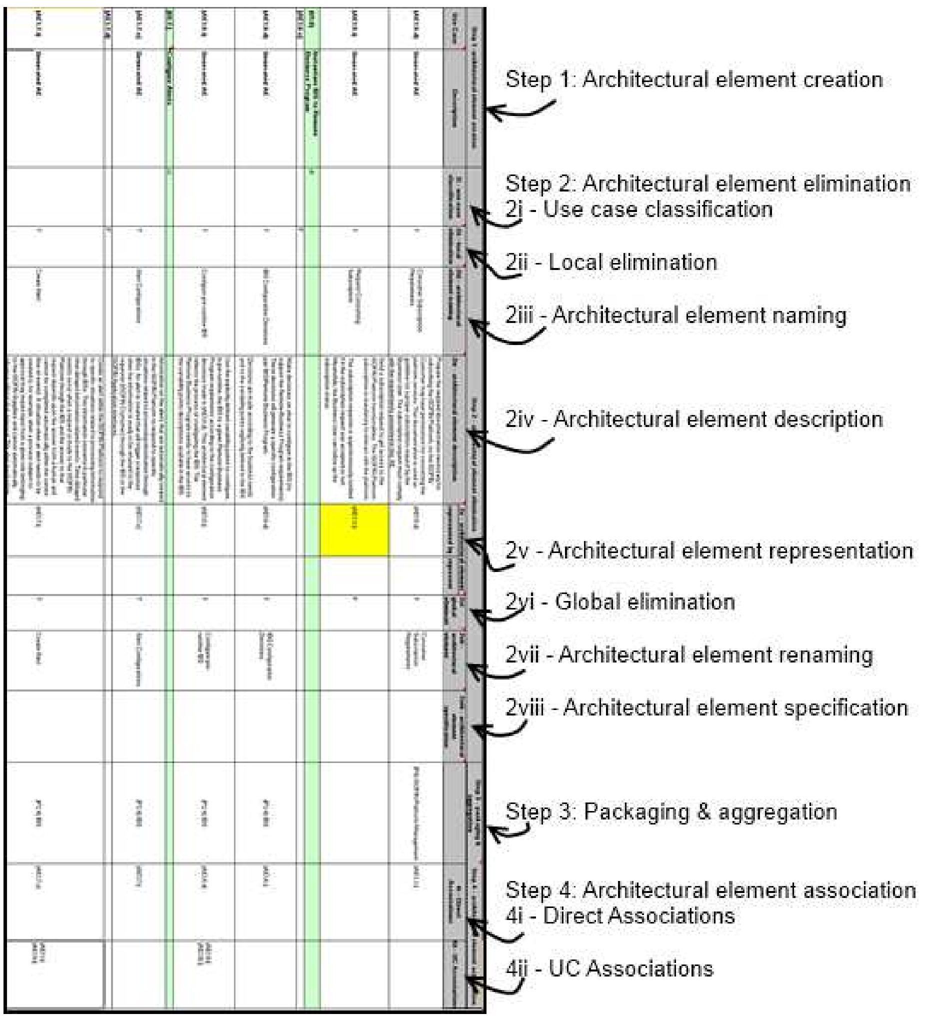
Step 1: Architectural element creation
Step 2: Architectural element elimination
2i - Use case classification
2ii - Local elimination
2iii - Architectural element naming
2iv - Architectural element description
2v - Architectural element representation
2vi - Global elimination
2vii - Architectural element renaming
2viii - Architectural element specification
Step 3: Packaging & aggregation
Step 4: Architectural element association
4i - Direct Associations
4ii - UC Associations

Fig. 3. Tabular Transformation of the 4SRS Method

The V-Model representation promotes the alignment between the models on the problem domain and the models on the solution domain. The presented models are created in succession, by manipulating the information that results from one 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.

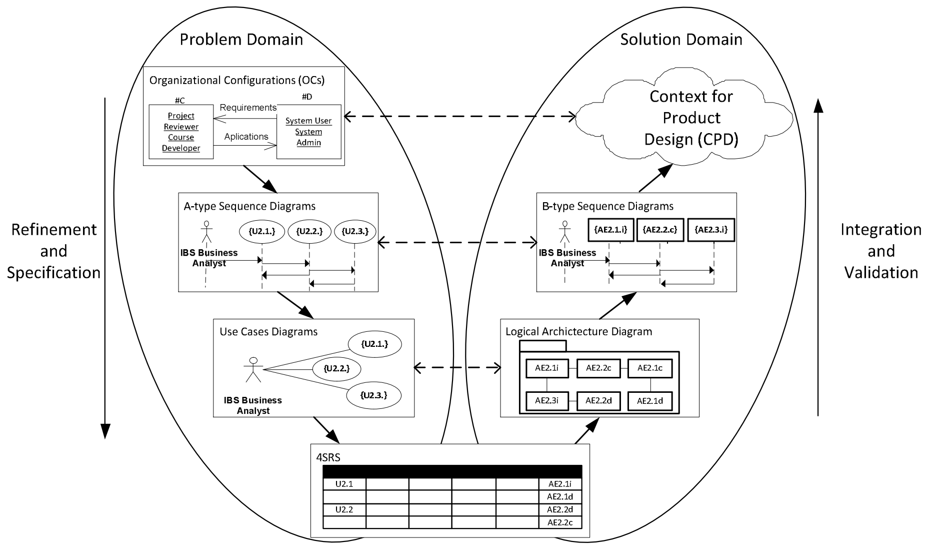

Fig. 4. V-Model Adaption for Domain and Software Alignment

To assess the V-Model approach, we present a process regarding our real 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 detail on required functionalities in order 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.

4 Case Study: 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 project: the ISOFIN project (Interoperability in Financial Software) [27]. This 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, allows for the semantic and application interoperability between enrolled financial institutions (Banks, Insurance Companies and others), as depicted in Fig. 5.

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.

If there is no 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, communality 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 authors of this paper proposed a process-level approach to tackle 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.

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. Fig. 5 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 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.

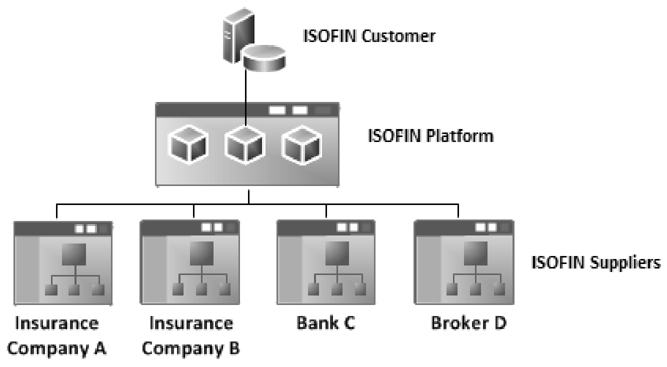

Fig. 5. Desirable Interoperability in ISOFIN

4.1 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 Fig. 6: OCs represents the first level; different types of Stereotyped UML Sequence Diagrams, presented as *A-Type* and *B-Type* Diagrams (later described) represent the other two.

Today's business is based on inter-organizational relations [24], having an impact on an organization's business and software strategy [28]. 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. An OC characterization must contain information on the performed activities (economical [21] or non-economical [29]), the several professional profiles (actors and skills) that participate in the activity execution and also the exchange of information or artifacts.

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:

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

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

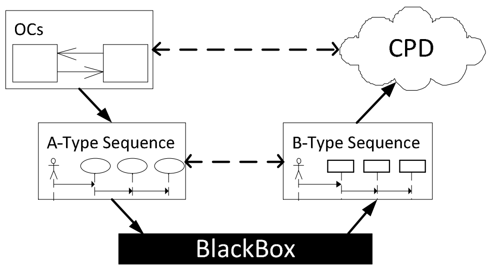

Fig. 6. Organizational Configurations and Interactions Alignment

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 Fig. 7, 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 paper, so only brief and simple examples are presented and only the types of activities are described.

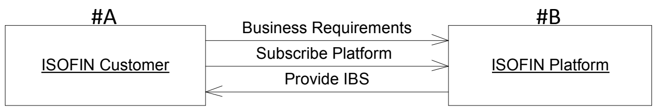

Fig. 7. Organizational Configuration Example

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 [30]. 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 [25]. We create *A-Type* Sequence Diagrams, as shown in Fig. 8. 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 Fig. 13).

In Fig. 8 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 Fig. 7).

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 [25]. 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.

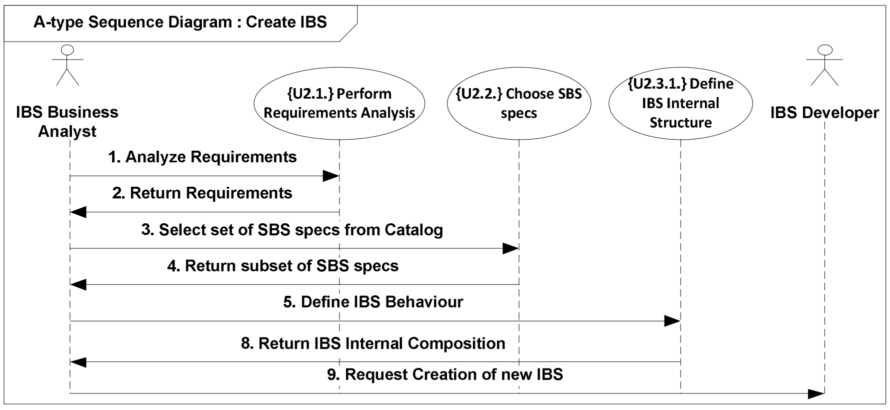
A-type Sequence Diagram : Create IBS
IBS Business Analyst
{U2.1.} Perform Requirements Analysis
{U2.2.} Choose SBS specs
{U2.3.1.} Define IBS Internal Structure
IBS Developer
1. Analyze Requirements
2. Return Requirements
3. Select set of SBS specs from Catalog
4. Return subset of SBS specs
5. Define IBS Behaviour
8. Return IBS Internal Composition
9. Request Creation of new IBS

Fig. 8. *A-Type* Sequence Diagram

One of the purposes of creating a software logical architecture is to support the system's functional requirements [9]. 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 (presented in section 4.3). 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 Fig. 9), we adopt different stereotype of UML sequence diagrams, where AEs (presented in the logical architecture), actors and packages (if justifiable) interactions are modeled. In Fig. 9, 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.

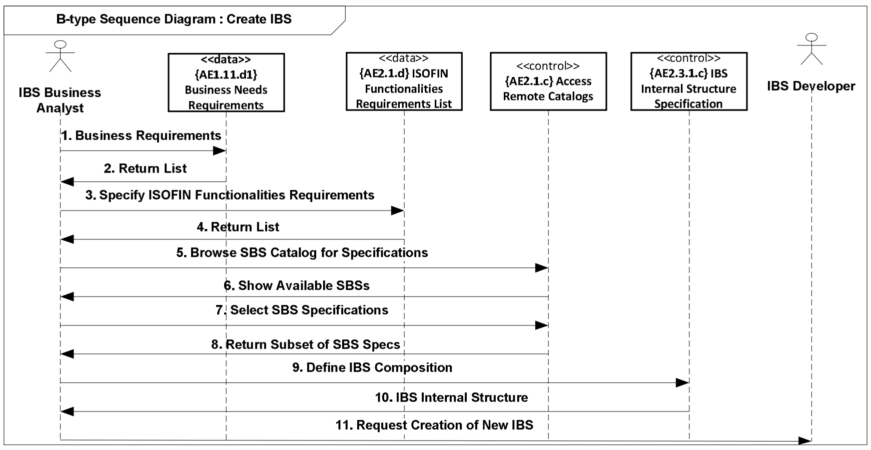
B-type Sequence Diagram : Create IBS
IBS Business Analyst
<<data>> {AE1.11.d1} Business Needs Requirements
<<data>> {AE2.1.d} ISOFIN Functionalities Requirements List
<<control>> {AE2.1.c} Access Remote Catalogs
<<control>> {AE2.3.1.c} IBS Internal Structure Specification
IBS Developer
1. Business Requirements
2. Return List
3. Specify ISOFIN Functionalities Requirements
4. Return List
5. Browse SBS Catalog for Specifications
6. Show Available SBSs
7. Select SBS Specifications
8. Return Subset of SBS Specs
9. Define IBS Composition
10. IBS Internal Structure
11. Request Creation of New IBS

Fig. 9. *B-Type* Sequence Diagram

4.2 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 [30]. We present in the left side of Fig. 10 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 Fig. 10). 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.

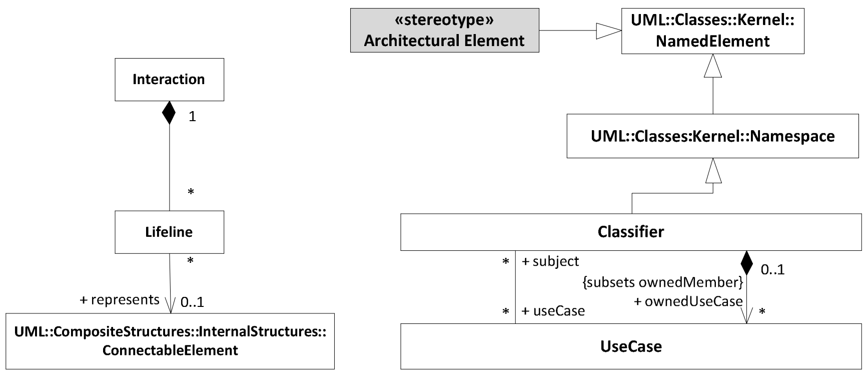

Fig. 10. The Proposed Extension to the UML Metamodel for Representing *A-Type* and *B-Type* Sequence Diagrams [29]

Like the *ConnectableElement* class, *UseCase* class is also generalized by *NamedElement* class. The information regarding abstract syntax, concrete syntax, well-formedness and semantics [31] 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*.

4.3 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 [22], and so detailing it is not in the scope of this work, being, as such, treated like a black box in the V-Model description as represented in Fig. 11. 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.

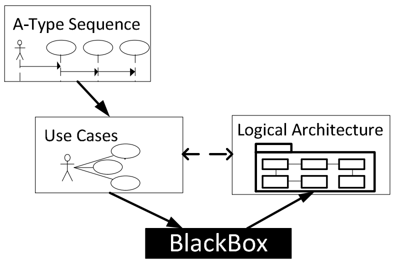

Fig. 11. Derivation of Process-Oriented Logical Architectures

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.

The use case model specifies the required usages of the ISOFIN Platform. In Fig. 13, 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.

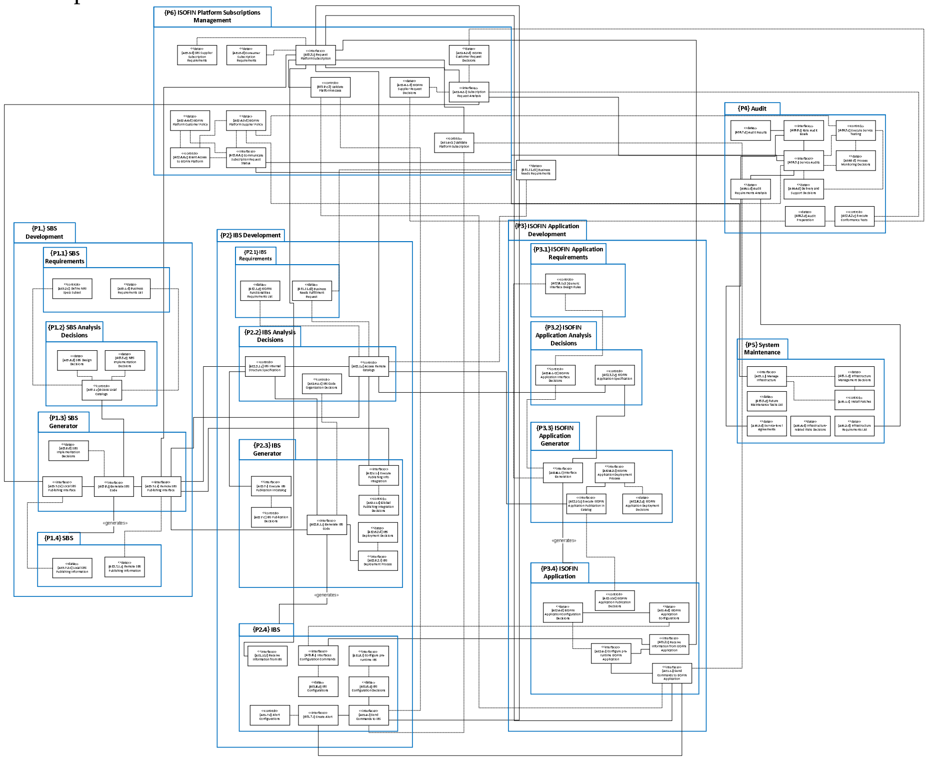
(P6) ISOFIN Platform Subscriptions Management
(P4) Audit
(P1) SBS Development
(P1.1) SBS Requirements
(P1.2) SBS Analysis Decisions
(P1.3) SBS Generator
(P1.4) SBS
(P2) IBS Development
(P2.1) IBS Requirements
(P2.2) IBS Analysis Decisions
(P2.3) IBS Generator
(P2.4) IBS
(P3) ISOFIN Application Development
(P3.1) ISOFIN Application Requirements
(P3.2) ISOFIN Application Analysis Decisions
(P3.3) ISOFIN Application Generator
(P3.4) ISOFIN Application
(P5) System Maintenance
«generates»
«generates»
«generates»

Fig. 12. ISOFIN Process-level Logical Architecture

Fig. 12 depicts the process-level logical architecture for the ISOFIN project and contains nearly eighty architectural elements. This figure is intentionally not zoomed in, 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 Fig. 14, detailing some of its constructors.

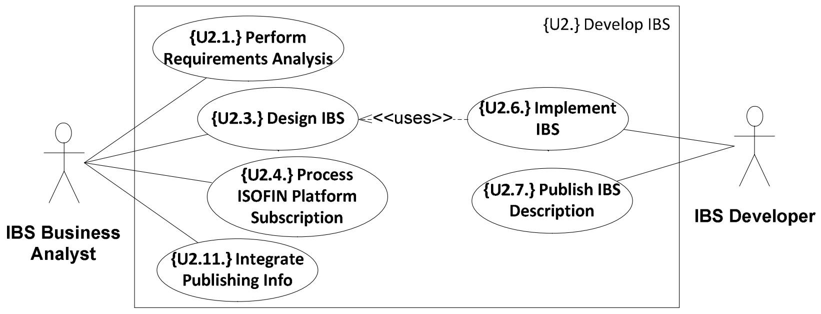
{U2.} Develop IBS
{U2.1.} Perform Requirements Analysis
{U2.3.} Design IBS
<<uses>>
{U2.6.} Implement IBS
{U2.4.} Process ISOFIN Platform Subscription
{U2.7.} Publish IBS Description
{U2.11.} Integrate Publishing Info
IBS Business Analyst
IBS Developer

Fig. 13. Subset of the Use Case Model from the ISOFIN Project

The 4SRS method execution results in a logical architecture diagram, presented in Fig. 12. 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 *{P6} 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.

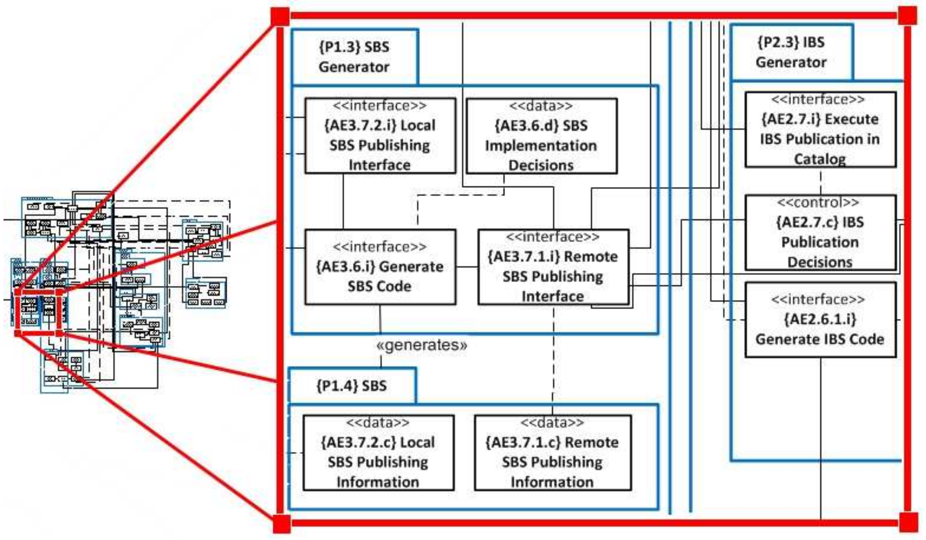
{P1.3} SBS Generator
<<interface>> {AE3.7.2.i} Local SBS Publishing Interface
<<data>> {AE3.6.d} SBS Implementation Decisions
<<interface>> {AE3.6.i} Generate SBS Code
<<interface>> {AE3.7.1.i} Remote SBS Publishing Interface
«generates»
{P1.4} SBS
<<data>> {AE3.7.2.c} Local SBS Publishing Information
<<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. 14. Subset of the ISOFIN Process-level Logical Architecture

The architectural elements that compose the logical architecture diagram result from the 4SRS method execution, taking as input the use cases and their textual descriptions from the use case model. In Fig. 14 we present a subset of the logical architecture diagram, correlated with the use case example from Fig. 13, where we can more easily depict such relation between the use cases and the architectural elements. For instance, the architectural element *{AE2.6.1.i} Generate IBS Code* derives from the 4SRS method execution applied to use *case {U2.6} Implement IBS*. Another example, the architectural elements *{AE2.7.c} IBS Publication Decisions* and *{AE2.7.i} Execute IBS Publication Deployment* derive from the use case *{U2.7} Publish IBS Description*.

4.4 V-Model Considerations and Comparison with Related Work

For creating a context for IT product design, the V-Model presented in this paper encompasses a set of artifacts through successive derivation. Our approach is different from existing ones [16-18], 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 [9]. This kind of representation also enables testing sequences of system actions that are meaningful at the software architecture level [32]. 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.

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 set of models (like GQM+Strategies [5], Balanced Scorecards [6] or COBIT [7]), all relate to aligning the domain-specific concerns with software solutions. As far as the authors of this paper 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.

4.5 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 [33].

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.

The focus of this section is not to present the ARID adaptation to our V-Model, which will be addressed in a future publication. Instead, we present a simplified diagram that encompasses major ARID representations required to align with our V-Model models, as seen on Fig. 15.

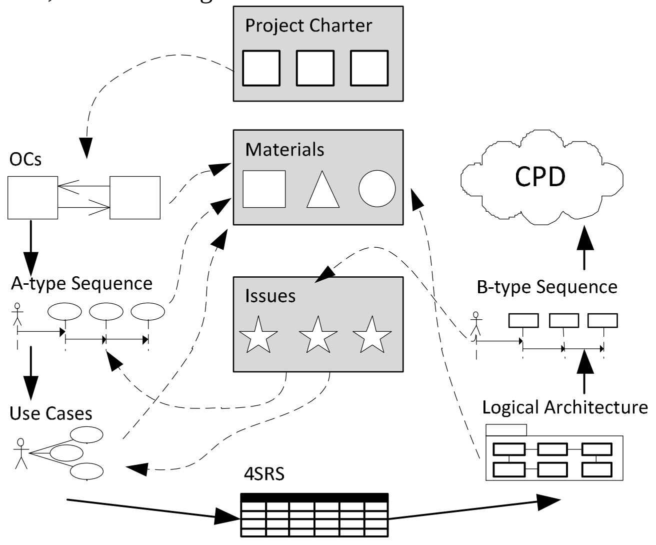

Fig. 15. ARID and the V-Model Intertwining

We 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 [34] 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.

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).

5 Conclusions and Outlook

In this paper, 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 use *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 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, decreasing, 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, contributes 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.

In 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.

As future work, we plan to study the derivation of the current process-level architecture into product-level models, maintaining business alignment.

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

References

1. Luftman, J., Ben-Zvi, T.: Key issues for IT executives 2010: judicious IT investments continue post-recession. MIS Quarterly Executive 9, 263-273 (2010)
2. Selic, B.: The pragmatics of model-driven development. Software, IEEE 20, 19-25 (2003)
3. Campbell, B., Kay, R., Avison, D.: Strategic alignment: a practitioner's perspective. Journal of Enterprise Information Management 18, 653-664 (2005)
4. Haskins, C., Forsberg, K.: Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities; INCOSE-TP-2003-002-03.2. 1. INCOSE (2011)
5. Basili, V.R., Lindvall, M., Regardie, M., Seaman, C., Heidrich, J., Munch, J., Rombach, D., Trendowicz, A.: Linking Software Development and Business Strategy Through Measurement. Computer 43, 57-65 (2010)
6. Kaplan, R.S., Norton, D.P.: The balanced scorecard–measures that drive performance. Harvard business review 70, 71-79 (1992)

7. Information Technology Governance Institute (ITGI): COBIT v5 - A Business Framework for the Governance and Management of Enterprise IT. ISACA (2012)
8. Sungwon, K., Yoonseok, C.: Designing logical architectures of software systems. 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. , pp. 330-337 (2005)
9. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Softw. 12, 42-50 (1995)
10. Clements, P., Garlan, D., Little, R., Nord, R., Stafford, J.: Documenting software architectures: views and beyond. pp. 740-741. IEEE (2003)
11. Hofmeister, C., Nord, R., Soni, D.: Applied software architecture. Addison-Wesley Professional (2000)
12. Zou, J., Pavlovski, C.J.: Modeling Architectural Non Functional Requirements: From Use Case to Control Case. e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on, pp. 315-322 (2006)
13. Conradi, R., Jaccheri, M.: Process Modelling Languages. Software Process: Principles, Methodology, and Technology, vol. 1500, pp. 27-52. Springer US (1999)
14. Browning, T.R., Eppinger, S.D.: Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans on Engineering Management 49, 428-442 (2002)
15. Winter, R., Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), pp. 30-30 (2006)
16. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional (1999)
17. 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)
18. Bayer, J., Muthig, D., Göpfert, B.: The library system product line. A KobrA case study. Fraunhofer IESE (2001)
19. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems (2002)
20. Machado, R.J., Fernandes, J., Monteiro, P., Rodrigues, H.: Refinement of Software Architectures by Recursive Model Transformations. In: Münch, J., Vierimaa, M. (eds.) Product-Focused Software Process Improvement, vol. 4034, pp. 422-428. Springer Berlin / Heidelberg (2006)
21. Machado, R.J., Fernandes, J.: Heterogeneous Information Systems Integration: Organizations and Methodologies. In: Oivo, M., Komi-Sirviö, S. (eds.) Product Focused Software Process Improvement, vol. 2559, pp. 629-643. Springer Berlin / Heidelberg (2002)
22. Ferreira, N., Santos, N., Machado, R.J., Gasevic, D.: Derivation of Process-Oriented Logical Architectures: An Elicitation Approach for Cloud Design. In: O. Dieste, A.J., and N. Juristo (ed.) 13th International Conference on Product-Focused Software Development and Process Improvement - PROFES 2012, vol. LNCS 7343, pp. 44–58. Springer-Verlag, Berlin Heidelberg, Germany Madrid, Spain (2012)
23. Campbell, B.: Alignment: Resolving ambiguity within bounded choices. (2005)
24. Evan, W.M.: Toward a theory of inter-organizational relations. Management Science 217-230 (1965)
25. Machado, R., Lassen, K., Oliveira, S., Couto, M., Pinto, P.: Requirements Validation: Execution of UML Models with CPN Tools. International Journal on Software Tools for Technology Transfer (STTT) 9, 353-369 (2007)
26. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of UML Models for Service-Oriented Software Architectures. Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, pp. 173-182. IEEE Computer Society (2005)

27. ISOFIN Research Project, http://isofincloud.i2s.pt
28. Barrett, S., Konsynski, B.: Inter-Organization Information Sharing Systems. MIS Quarterly 6, 93-105 (Dec., 1982)
29. Bensaou, M., Venkatraman, N.: Interorganizational relationships and information technology: A conceptual synthesis and a research framework. European Journal of Information Systems 5, 84-91 (1993)
30. Open Management Group (OMG), http://www.omg.org/spec/UML/2.4.1/
31. Atkinson, C., Kuhne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Softw. 20, 36-41 (2003)
32. Bertolino, A., Inverardi, P., Muccini, H.: An explorative journey from architectural tests definition down to code tests execution. Proceedings of the 23rd International Conference on Software Engineering, pp. 211-220. IEEE Computer Society, Toronto, Ontario, Canada (2001)
33. Clements, P.C.: Active Reviews for Intermediate Designs., Technical Note CMU/SEI-2000-TN-009. (2000)
34. Project Management Institute: A Guide to the Project Management Body of Knowledge (PMBOK® Guide) (2008)