A Demonstration Case on Steps and Rules for the Transition from Process-Level to Software Logical Architectures in Enterprise Models*

Nuno Ferreira[1], Nuno Santos[2], Pedro Soares[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,psoares}@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. At the analysis phase of an enterprise information system development, the alignment between the process-level requirements (information systems) with the product-level requirements (software system) may not be properly achieved. Modeling the processes for the enterprise's business is often insufficient for implementation teams, and implementation requirements are often misaligned with business and stakeholder needs. In this paper, we demonstrate, though a real industrial case, how transition steps and rules are used to assure that process- and product-level requirements are aligned, within an approach that supports the creation of the intended requirements. The input for the transition steps is an information system logical architecture, and the output is a product-level (software) use case model.

Keywords: Enterprise Information Systems, Enterprise Modeling, Requirement Elicitation, Model Transformation, Transition to Software Requirements.

1 Introduction

During an enterprise information system development process, assuring that functional requirements fully support the stakeholder's business needs may become a complex and inefficient task. Additionally, the "newfound" paradigm of IT solutions (*e.g.*, Cloud Computing) typically results in more difficulties for defining a business model and for eliciting product-level functional requirements for any given project. If stakeholders

* This work has been supported by project ISOFIN (QREN 2010/013837), Fundos FEDER through Programa Operacional Fatores de Competitividade – COMPETE and by Fundos Nacionais through FCT – Fundação para a Ciência e Tecnologia within the Project Scope: FCOMP-01-0124-FEDER-022674.

J. Grabis et al. (Eds.): PoEM 2013, LNBIP 165, pp. 277–291, 2013.

experience such difficulties then software developers will have to deal with incomplete or incorrect requirements specifications, resulting in a real problem.

When there are insufficient inputs for a product-level approach to requirements elicitation, using a process-level perspective is a possible approach, in order to create an information system logical architecture which is used for eliciting software (product-level) requirements.

The first effort should be to specify the requirements of the overall system in the physical world; then to determine necessary assumptions about components of that physical world; and only then to derive a specification of the computational part of the control system [1]. There are similar approaches that tackle the problem of aligning domain specific needs with software solutions. For instance, goal-oriented approaches are a way of doing so, but they don't encompass methods for deriving a logical representation of the intended system processes with the purpose of creating context for eliciting product-level requirements.

Our main problem, and the main topic this paper addresses, is assuring that product-level (IT-related) requirements are perfectly aligned with process-level requirements, and hence, are aligned with the organization's business requirements. The process-level requirements express the need for fulfilling the organization's business needs, and we detail how they are characterized within our approach further in section 2. These requirements may be supported by analysis models, that are implementation agnostic [2]. According to [2], the existing approaches for transforming requirements into an analysis model (i) don't require acceptable user effort to document requirements, (ii) are efficient enough (*e.g.*, one or two transformation steps), (iii) are able to (semi-)automatically generate a complete (*i.e.*, static and dynamic aspects) consistent analysis model, which is expected to model both the structure and behavior of the system at a logical level of abstraction.

In this paper we present a demonstration case in which we illustrate the transition between the process-level requirements of the intended system and the technological requirements that the same system must comply with. The transition is part of an approach that expresses the project goals and allows creating context to implement a software system. The entire approach is detailed in [3] as a V + V process, based on the composition of two V-shaped process models (inspired in the "Vee" process model [4]). This way, we formalize the transition steps between perspectives that are required in order to align the requirements the V+V process presented in [3]. The requirements are expressed through logical architectural models and stereotyped sequence diagrams [5] in both a process- and a product-level perspective.

This paper is structured as follows: section 2 briefly presents the macro-process for information system's development based on both process- and product-level V-Model approaches; section 3 describes the transition steps and rules between both perspectives; in section 4 we present a real industrial case on the adoption of the transition steps between both V-Model executions; in section 5 we compare our approach with other related work; and in section 6 we present the conclusions.

2 A Macro-Process Approach to Software Design

The development process of information systems can be regarded (in a simple way) as a cascaded lifecycle (*i.e.*, a development process only initiates when the previous has ended), if we consider typical and simplified phases: analysis, design and implementation. Our approach encompasses two V-shaped process models hereafter referred as the V+V process. The main difference from our proposed approach to other information system development approaches is that it is applicable for eliciting product-level requirements in cases where there is no clearly defined context for eliciting product requirements within a given specific domain, by first eliciting process-level requirements and then evolving to the product-level requirements, using a transition approach that assures an alignment between both perspectives. Other approaches (described further in section 5) typically apply to a single perspective.

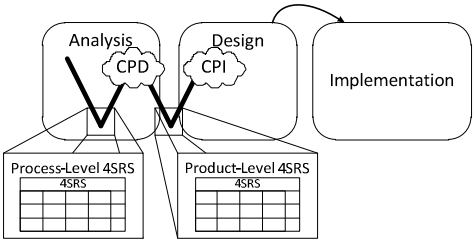

Fig. 1. V+V process framed in the development macro-process

The first V-Model (at process-level) is composed by *Organizational Configurations* (OC) [5], *A-type* and *B-type* sequence diagrams [5] (stereotyped sequence diagrams that use, respectively, use cases and architectural elements from the logical architecture), and (business) *Use Case* models (UCs) that are used to derive (and, in the case of *B-type* sequence diagrams, validate) a process-level logical architecture (*i.e.*, the information system logical architecture). Use cases are mandatory to execute the 4SRS method. Since the term *process* has different meanings depending on the context, in our process-level approach we acknowledge that: (i) real-world activities of a software production process are the context for the problem under analysis; (ii) in relation to a software model context [8], a software process is composed of a set of activities related to software development, maintenance, project management and quality assurance. For the scope definition of our work, and according to the previously exposed acknowledgments, we characterize our process-level perspective by: (i) being related to real-world activities (including business); (ii) when related to software, those activities encompass the typical software development lifecycle. Our process-level approach is characterized by using refinement (as one kind of functional decomposition) and integration of system models. Activities and their interface in a process can be structured or arranged in a process architecture [9]. We frame the process-level V-Model (the first V-Model of Fig. 1) in the analysis phase, creating the context for product design (*CPD*). In its vertex, the process-level 4SRS (Four-Step Rule-Set) method execution (see [6] for details about the process-level 4SRS method) assures the transition from the problem to the solution domain by transforming

process-level use cases into process-level logical architectural elements, and results in the creation of a validated architectural model which allows creating context for the product-level requirements elicitation and in the uncovering of hidden requirements for the intended product design.

The second V-Model (at product-level) is composed by *Mashed UCs* model (a use case model composed by use cases derived from the transition steps but not yet being the final product use case model), *A-type* and *B-type* sequence diagrams, and (software) *Use Case* models (UCs) that are used to derive (and, validate) a product-level logical architecture (*i.e.*, the software system logical architecture). By product-level, we refer as the typical software requirements. The second execution of the V-Model is done at a product-level perspective and its vertex is supported by the product-level 4SRS method detailed in [7]. The product-level V-Model gathers information from the *CPD* in order to create a new model referred as *Mashed UCs*. The creation of this model is detailed in the next section of this paper as transition steps and rules. The product-level V-Model (the second V-Model of Fig. 1) enables the transition from analysis to design trough the execution of the product-level 4SRS method (see [7] for details about the product-level 4SRS method). The resulting architecture is then considered a design artifact that contributes for the creation of context for product implementation (*CPI*) as information required by implementation teams. Note that the design itself is not restricted to that artifact, since in our approach it also encompasses behavioral aspects and non-functional requirements representation.

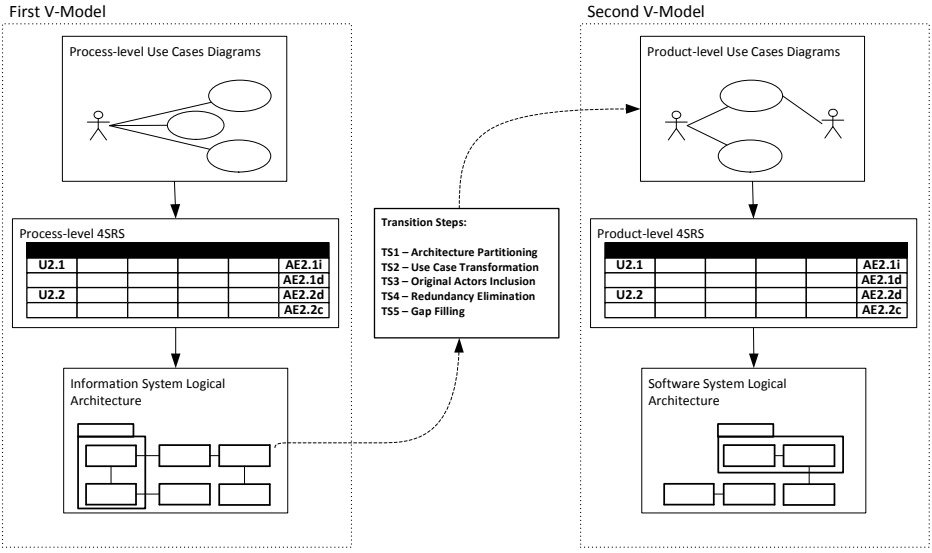

Fig. 2. Derivation of software system logical architectures by transiting from information system logical architectures

The information regarding each of the models and their usage within the V+V process is detailed in [3] and as such is not our purpose to thoroughly describe them, leaving the reader with just a brief explanation on their meaning. The OC model is a high-level representation of the activities (interactions) that exist between the

business-level entities of a given domain. *A-type* and *B-type* sequence diagrams are stereotyped sequence diagram representation to describe interactions in early analysis phase of system development. *A-type* sequence diagrams use actors and use cases. *B-type* sequence diagrams use actors and architectural elements depicted in the both the information system or software system logical architectures.

As depicted in Fig. 2, the result of the first V-Model (process-level) execution is the information system logical architecture. The architectural elements that compose this architecture are derived (by performing transition steps) into product-level use cases (*Mashed UCs* model). The result of the second V-Model (product-level) execution is the software system logical architecture.

3 Process- to Product-Level Transition

As stated before, a first V-Model (at process-level) can be executed for business requirements elicitation purposes, followed by a second V-Model (at product-level) for defining the software functional requirements. The V+V process is useful for both stakeholders, organizations and technicians, but it is necessary to assure that they properly reflect the same system.

This section begins by presenting a set of transition steps whose execution is required to create the initial context for product-level requirements elicitation, referred to as *Mashed UC* model. The purpose of the transition steps is to assure an aligned transition between the process- and product-level perspectives in the V+V process, that is, the passage from the first V-Model to the second one. By defining these transition steps, we assure that product-level (software) use cases (UCpt's) are aligned with the architectural elements from the process-level (information system) logical architectural model (AEpc's); *i.e.*, (software) use case diagrams are reflecting the needs of the information system logical architecture. The application of these transition rules to all the partitions of an information system logical architecture gives origin to a set of *Mashed UC* models (preliminary product-level use case models).

To allow the recursive execution of the 4SRS method [10, 11], the transition from the first V-Model to the second V-Model must be performed by a set of steps. The output of the first V-Model must be used as input for the second V-Model; *i.e.*, we need to transform the information system logical architecture into product-level use case models. The transition steps to guide this mapping must be able to support business to technology changing. These transition steps (TS) are structured as follows:

TS1 - *Architecture Partitioning*, where the process-level architectural elements (AEpc's) under analysis are classified by their computation execution context with the purpose of defining software boundaries to be transformed into product-level (software) use cases (UCpt's.);

TS2 - *Use Case Transformation*, where AEpc's are transformed into software use cases and actors that represent the system under analysis through a set of transition patterns that must be applied as rules;

TS3 - *Original Actors Inclusion*, where the original actors that were related to the use cases from which the architectural elements of the process-level perspective are derived (in the first V execution) must be included in the representation;

TS4 - *Redundancy Elimination*, where the model is analyzed for redundancies;

TS5 - *Gap Filling*, where the necessary information of any requirement that is not yet represented, is added, in the form of use cases.

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

The identification of each partition is firstly made using the information that results from the packaging and aggregation efforts of the previous 4SRS execution (step 3 of the 4SRS method execution as described in [6]). Nevertheless, this information is not enough to properly identify the partitions. Information gathered in OC's and on the process-level *B-type* sequence diagrams must also be accounted. A partition is created by identifying all the relevant architectural elements that belong to all *B-type* sequence diagrams that correspond to a given organizational configuration scenario. By traversing the architectural elements that comply with the scenario definition (for each *B-type* sequence diagram and aligned with the packages and aggregations presented in the information system logical architecture), it is possible to properly identify the partitions that support the interactions depicted in the OC's.

The rules to support the execution of the transition step 2 are applied in the form of transition rules and must be applied in accordance to the stereotype of the envisaged architectural element. There are three stereotyped architectural elements: *d-type*, which refer to generic decision repositories (data), representing decisions not supported computationally by the system under design; *c-type,* which encompass all the processes focusing on decision making that must be supported computationally by the system; and *i-type*, which refer to process' interfaces with users, software or other processes. The full descriptions of the three stereotypes are available in [6].

The defined transition rules (TR), from the logical architectural diagram to the *Mashed UC* diagram are as follows:

TR1 - an inbound *c-type* or *i-type* AEpc is transformed into an UCtr of the same type (see Fig. 3). By inbound we mean that the element belongs to the partition under analysis;

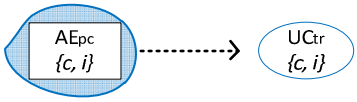

Fig. 3. TR1 - transition rule 1

TR2 - an inbound *d-type* AEpc is transformed into an UCtr and an associated actor (see Fig. 4). This is due to the fact that *d-type* AEpc's corresponds to decisions not computationally supported by the system under design and, as such, it requires an actor to activate the depicted process.

Fig. 4. TR2 - transition rule 2

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

TR3 - an inbound AEpc, with a given name *x*, which also belongs to an outbound partition, is transformed into an UCtr of name *x*, and an associated actor, of name *y*, being the responsible for representing the outbound actions associated with UCtr*x* (see Fig. 5).

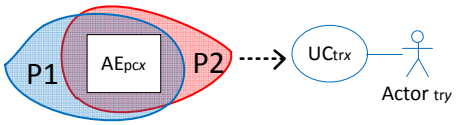

Fig. 5. TR3 - transition rule 3

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

TR4 - an inbound *d-type* UCtr of name *x* with connections to an (any type) UCtr of name *y* and to an actor *z*, gives place to two UCtr's, *x* and *y*, maintaining the original types (see Fig. 6). Both are connected to the actor *z*. This means that all existing connections on the original *d-type* AEpc that were maintained during execution of TR2 or TR3 are transferred to the created actor.

Fig. 6. TR4 - transition rule 4

The previous rule is executed after TR1, TR2 or TR3, so it only needs to set the required association between the UCtr's and the actors, that is to say, after all transformations are executed (TR1, TR2, and TR3), a set of rules are executed to establish the correct associations to the UCtr's.

TR5 - an inbound UCtr of name *x* with a connection to an outbound AEpc of name *y* (note that this is still an AEpc, since it was not transformed into any other concept in the previous transition rules) gives place to both an UCtr named *x* and to an actor named *y* (see Fig. 7). AEpc's that were not previously transformed are now transformed by the application of this TR5; this means that all AEpc's which exist outside the partition under analysis having connections with inbound UCtr's will be transformed into actors. These actors will support the representation of required external inputs to the inbounds UCtr's created during application of TR1, TR2, or TR3.

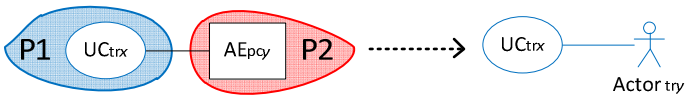

Fig. 7. TR5 - transition rule 5

A special application of TR5 (described as TR5.1) can be found in Fig. 8 where we can see an UCtr with a connection to an outbound AEpc and another connection to an actor. In this case, TR5 is applied and the resulting UCtr is also connected to the original actor. Note that an UCtr belonging to multiple partitions is first and foremost, an inbound UCtr due to being under analysis.

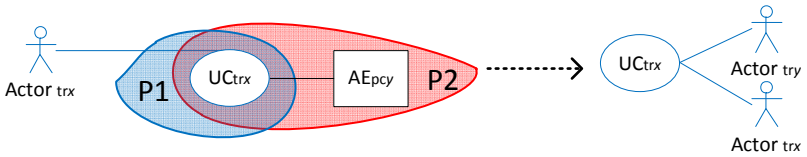

Fig. 8. TR5.1 - transition rule 5.1

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

4 Applicability of the Transition Steps: The ISOFIN Project

The applicability of the transition steps and rules is demonstrated with a real project: the ISOFIN project (Interoperability in Financial Software) [12]. This project aimed to deliver a set of coordinating services in a centralized infrastructure, enacting the coordination of independent services relying on separate infrastructures. The resulting ISOFIN platform allows for the semantic and application interoperability between enrolled financial institutions (Banks, Insurance Companies and others).

The global ISOFIN architecture relies on two main service types: 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 SBS's and/or IBS's exposing functionalities that relate directly to business needs. SBS's are a set of functionalities that are exposed from the ISOFIN Suppliers production infrastructure.

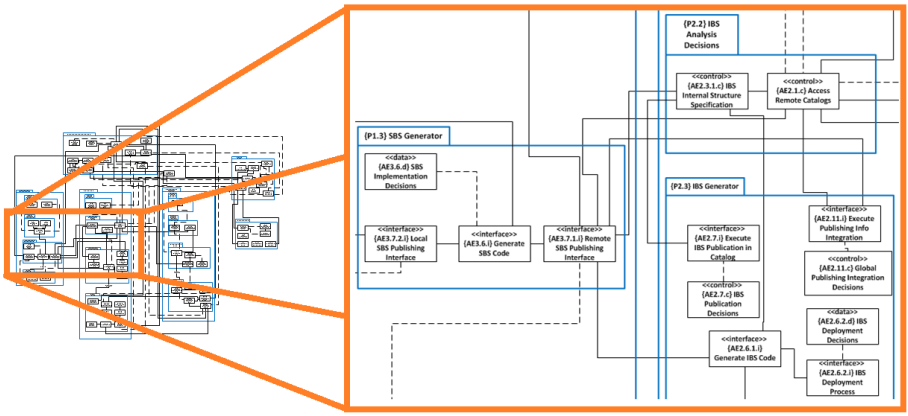

Fig. 9. Subset of the ISOFIN information system logical architecture

From the demonstration case, we first present a subset of the information system logical architecture in Fig. 9, that resulted from the execution of the 4SRS method at a process-level perspective [6]; *i.e.*, the execution of the first (process-level) V-Model. The information system logical architecture is composed by architectural elements that represent processes executed by within the ISOFIN platform.

In Fig. 10, we depict the execution of TS1 to a subset of the entire information system logical architecture (composed by the same architectural elements as Fig. 9), *i.e.*, the partitioning of the information system logical architecture, by marking its architectural elements in partition areas, each concerning the context where services are executed, which resulted in two partitions: (i) the ISOFIN platform execution functionalities (in the area marked as P1); (ii) the ISOFIN supplier execution functionalities (in the area marked as P2). The identification of the partitions will enable the application of the transition steps to allow the application of the second V-Model to advance the macro-process execution into the product implementation. Presenting the information that supported the decisions regarding the partitions in the case of the ISOFIN project is out of the scope of this paper.

Fig. 11 shows the filtered and collapsed diagram that resulted from the P2 partition, which (in the demonstration case) is the partition under analysis. P2 includes the architectural elements that belong to both partitions and that must be considered when applying the transition rules. After being filtered and collapsed, the partitioned information system logical architecture is composed, not only by the architectural elements that belong to the partition under analysis, but also by some additional architectural elements belonging to other partitions that has an association (*i.e.*, the dashed and/or straight lines between architectural elements) with architectural elements belonging to the partition under analysis (*e.g.*, *{AE3.6.i} Generate SBS Code* belongs to P1, but possesses an association with *{AE3.7.1.i} Remote SBS Publishing Interface* that belongs to P1 and P2 partitions). The keeping of these outbound AEpc's assures that outbound interfaces information is preserved.

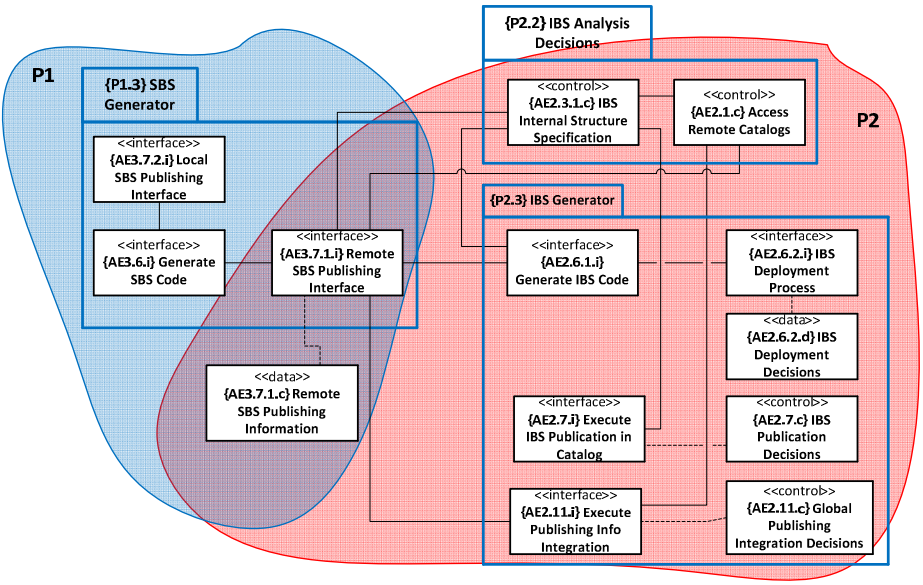

Fig. 10. Partitioning of the information system logical architecture (TS1)

The model is now ready to be transformed. It is during TS2 that the perspective is altered from process- to product-level. We now execute the transition rules presented in the previous section to our demonstration case. We transform from the source model (model from Fig. 11) to the target model (model from Fig. 12), as well as the TR that was applied, are supported in Table 1. Table 1 allows a better understanding of the application of the TR and the result of the transformation executed in TS2.

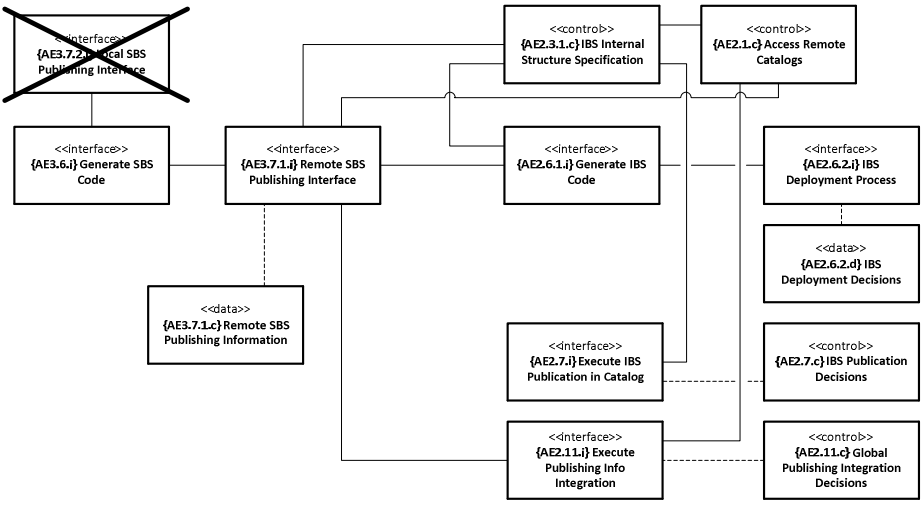

Fig. 11. Filtered and collapsed architectural elements (TS1)

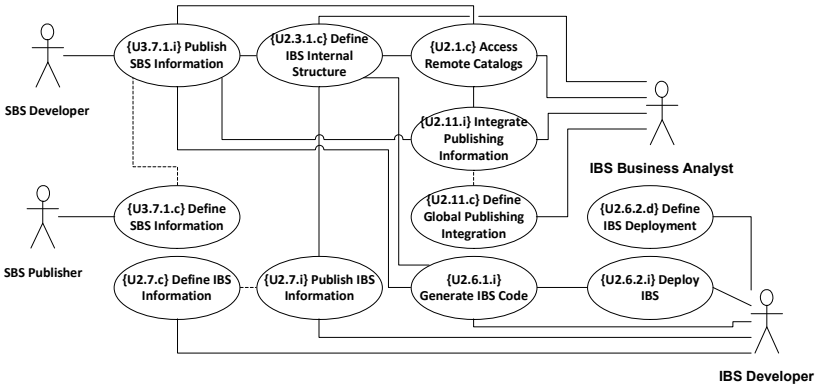

Fig. 12. Mashed *UC* model resulting from the transition from process- to product-level

The resulting model however presents some redundancies and gaps, so it is necessary that the remaining TS are executed. In Fig. 12, we depict the final *Mashed UC* model (the first product-level artifact in the second V-Model), resulting from the execution of TS2-5. Due to space restrictions, we only show the result of the execution of these four transition steps altogether. The resulting mashed use cases are the result of the application of the transition rules in TS2. The resulting model however presents redundancies and gaps, so it is necessary that the remaining TS are executed.

Table 1. Executed transformations to the model

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

It is possible to objectively recognize the effect of the application of some transition rules previously described. TR1 was the most applied transition rule and one example is the transformation of the AEpc named *{AE2.1.c} Access Remote Catalogs* into one UCtr named *{U2.1.c} Access Remote Catalogs*. One example of the application of TR2 is the transformation of the AEpc named *{AE2.6.2.d} IBS Deployment Decisions* into the UCtr named *{U2.6.2.d} Define IBS Deployment* and the actor named *IBS Developer*. TR3 was applied, for instance, in the transformation of the AEpc named *{AE3.7.1.c} Define SBS Information* into the UCtr named *{U3.7.1.c} Define SBS Information* and the actor named *SBS Publisher*. Finally, we can recognize the application of TR5.1 in the transformation of the AEpc named *{AE3.6.i} Generate SBS Code* into the actor named *SBS Developer*. All the other actors result from the execution of TS3. We must refer, for instance, that the actor *SBS Developer* results from the execution of TS4, since the original actor and the actor resulting from an application of TR2 and TR5.1 and also the inclusion of the original actor in TS3, result in the same actor which brings the need to eliminate the generated redundancy. The resulting model allows to identify potential gaps in use cases or actors (in the execution of TS5), but in this case such wasn't required.

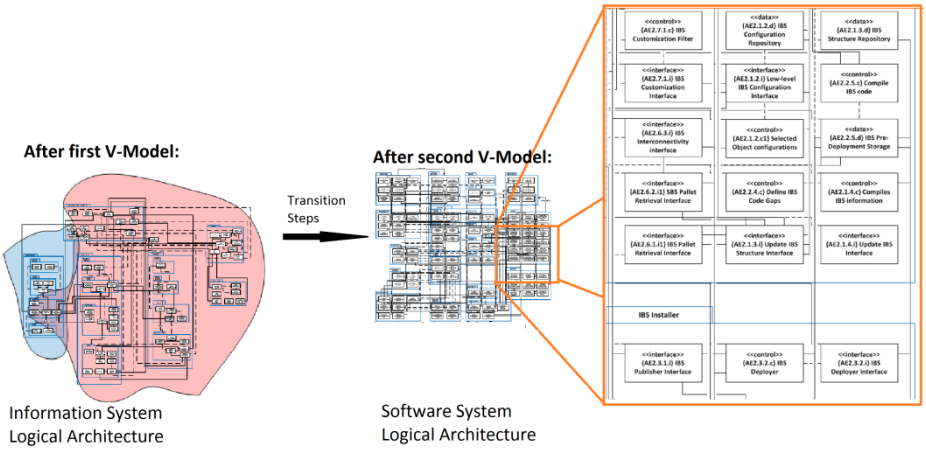

Fig. 13. Subset of the ISOFIN software system logical architecture after V+V process

After the execution of the transition steps, the *Mashed UC* model is used as input for the product-level 4SRS method execution [7] in order to derive the software system logical architecture for the ISOFIN platform. Such architecture is the main output of the second (product-level) V-Model execution. We depict in Fig. 13 the entire software system logical architecture (the second architecture of Fig. 13) obtained after the execution of the V+V process (and, as represented in Fig. 2, derived by transforming product use cases in architectural elements using product-level 4SRS method), having as input the information system logical architecture (the first architecture of Fig. 13) previously presented. The software system logical architecture is composed by architectural elements (depicted in the zoomed area) that represent functionalities that are executed in the platform. The alignment between the architecture elements in

both perspectives is supported by the transition steps. It would be impossible to elicit requirements for a software system logical architecture as complex as the ISOFIN platform (the overall information system logical architecture was composed by near 80 architectural elements, and the resulting software system logical architecture by near 100) by adopting an approach that only considers the product-level perspective.

5 Comparison with Related Work

There are many approaches that allow deriving at a given level a view of the intended system to be developed. Our approach clearly starts at a process-level perspective, and by successive models derivation creates the context for transforming the requirements expressed in an information system logical architecture into product-level context for requirements specification. Other approaches provide similar results at a subset of our specification. For instance, KAOS, a goal-oriented requirement specification method, provides a specification that can be used in order to obtain architecture requirements [13]. This approach uses two step-based methods, which output a formalization of the architecture requirements for each method, each of one providing a different view of the system. The organization's processes can be represented by an enterprise architecture [14], and representation extended by including in the architecture modeling concerns as business goals and requirements [15]. However, such proposals don't intend to provide information for implementation teams during the software development process, but instead to provide to stakeholders with business strategic requirements. The relation between what the stakeholders want and what implementation teams need requires an alignment approach to assure that there are no missing specifications on the transition between phases. An alignment approach also based on architectural models is presented in [16].

In [17] it is specified a mapping technique and an algorithm for mapping business process models, using UML activity diagrams and use cases, so functional requirements specifications support the enterprise's business process. In our approach, we use an information system logical architecture diagram instead of an activity diagram, since an information system logical architecture provides a fundamental organization of the development, creation, and distribution of processes in the relevant enterprise context [18]. Model-driven transformation approaches were already used for developing information systems in [19]. In literature, model transformations are often related to the Model-Driven Architecture (MDA) [20] initiative from OMG. MDA-based transformations are widely used but, as far as the authors know, the supported transformations don't regard a perspective transition, i.e., are perspective agnostic since they concern model transformations within a single perspective (typically the product-level one). For instance, [21] describes MDA-based transformations from use cases and scenarios to components, but only in a product-level perspective. Even in cases when MDA transformations are executed using different source and target modeling languages (there is a plethora in literature regarding these cases, like, for instance, [22], where a source model in Business Process Modeling Notation – BPMN, is transformed into target model in Business Process Execution Language – BPEL),

the transformation only regards a single perspective. The concerns that must be assured by transiting between perspectives are not dealt by any of the previous works.

The existing approaches for model transformation attempt to provide an automated or automatic execution. [2] provides a systematic review and evaluation of existing work on transforming requirements into an analysis model and, according to the authors, none of the compared approaches provide a practical automated solution. The transition steps and rules presented in this work intent to provide a certain level of automation into our approach and improve the efficiency, validation, and traceability of the overall V+V process.

6 Conclusions

In this paper, we demonstrated through a real industrial case the transition from previously process-level elicited requirements to requirements in a product-level perspective, included in an elicitation approach based in two V-Models (the V+V process). We illustrated a demonstration case that elicits requirements for developing a complex interoperable platform, by adopting a model-based approach to create context for business software implementation teams in situations where requirements cannot be properly elicited. Our approach is supported on a set of transition steps and transition rules that use as basis an information system logical architecture to output a product-level use case model. By adopting the approach, requirements for specifying software system functionalities are properly aligned with organizational information system requirements in a traceable way.

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

References

1. Maibaum, T.: On specifying systems that connect to the physical world. New Trends in Software Methodologies, Tools and Techniques (2006)
2. Yue, T., Briand, L.C., Labiche, Y.: A Systematic Review of Transformation Approaches between User Requirements and Analysis Models. Requirements Engineering (2011)

3. Ferreira, N., Santos, N., Soares, P., Machado, R.J., Gašević, D.: Transition from Process- to Product-level Perspective for Business Software. In: Poels, G., et al. (eds.) CONFENIS 2012. LNBIP, vol. 139, pp. 268–275. Springer, Heidelberg (2013)
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. Ferreira, N., Santos, N., Machado, R.J., Gašević, D.: Aligning Domain-Related Models for Creating Context for Software Product Design. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 168–190. Springer, Heidelberg (2013)
6. Ferreira, N., Santos, N., Machado, R.J., Gašević, D.: Derivation of Process-Oriented Logical Architectures: An Elicitation Approach for Cloud Design. In: Dieste, O., Jedlitschka, A., Juristo, N. (eds.) PROFES 2012. LNCS, vol. 7343, pp. 44–58. Springer, Heidelberg (2012)
7. Machado, R.J., et al.: Transformation of UML Models for Service-Oriented Software Architectures. In: ECBS 2005, pp. 173–182. IEEE Computer Society (2005)
8. Conradi, R., Jaccheri, M.: Process Modelling Languages. In: Software Process: Principles, Methodology, and Technology, pp. 27–52. Springer US (1999)
9. Browning, T.R., Eppinger, S.D.: Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans. on Engineering Management (2002)
10. Machado, R.J., Fernandes, J.M., Monteiro, P., Daskalakis, C.: 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)
11. 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)
12. ISOFIN Consortium. ISOFIN Research Project; ISOFIN Research Project (2010), http://isofincloud.i2s.pt
13. Jani, D., Vanderveken, D., Perry, D.: Experience Report: Deriving architecture specifications from KAOS specifications (2003)
14. The Open Group. TOGAF - The Open Group Architecture Framework, http://www.opengroup.org/togaf/
15. Engelsman, W., et al.: Extending enterprise architecture modelling with business goals and requirements. Enterprise Information Systems 5(1), 9–36 (2010)
16. Strnadl, C.F.: Aligning Business and It: The Process-Driven Architecture Model. Information Systems Management 23(4), 67–77 (2006)
17. Dijkman, R.M., Joosten, S.M.M.: An algorithm to derive use cases from business processes. In: SEA 2002. Acta Press, Cambridge (2002)
18. Winter, R., Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. In: EDOCW 2006, p. 30 (2006)
19. Iribarne, L., et al.: A Model Transformation Approach for Automatic Composition of COTS User Interfaces in Web-Based Information Systems. Information Systems Management 27(3), 207–216 (2010)
20. OMG, MDA Guide Version 1.0.1, OMG Std
21. Kaindl, H., Falb, J.: Can We Transform Requirements into Architecture? In: ICSEA 2008 (2008)
22. Bauer, B., Müller, J.P., Roser, S.: A Model-driven Approach to Designing Cross-Enterprise Business Processes. In: Meersman, R., Tari, Z., Corsaro, A. (eds.) OTM-WS 2004. LNCS, vol. 3292, pp. 544–555. Springer, Heidelberg (2004)