Chapter 11
Transition from Information Systems to Service-Oriented Logical Architectures: Formalizing Steps and Rules with QVT

Nuno Santos, Nuno Ferreira and Ricardo J. Machado

Abstract Specifying functional requirements brings many difficulties namely when regarding the cloud services. During the analysis phase, the alignment between the process-level requirements (information systems) with the product-level requirements (service-based software) may not be properly achieved or even understood. In this chapter, we describe an approach that supports the creation of the intended requirements, beginning in a process-level and evolving to a product-level perspective, to elicit requirements for specifying services that execute in a cloud computing environment. The transition between perspectives are supported by UML model transformations, encompassing a set of transition rules using QVT, from one perspective to the other, in order to assure that process- and product-level requirements are aligned.

Keywords Information systems design · Logical architectures · Requirement analysis · Model transformation · Service-oriented logical architecture · UML use cases · Transition rules

N. Santos (✉) · R.J. Machado
CCG/ZGDV Institute, University of Minho, Guimarães, Portugal
e-mail: nuno.santos@ccg.pt

R.J. Machado
e-mail: rmac@dsi.uminho.pt

N. Santos · R.J. Machado
ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

N. Ferreira
I2S – Insurance Software Systems, Porto, SA, Portugal
e-mail: nuno.ferreira@i2s.pt

© Springer International Publishing AG 2017

M. Ramachandran and Z. Mahmood (eds.), *Requirements Engineering for Service and Cloud Computing*, DOI 10.1007/978-3-319-51310-2_11

11.1 Introduction

The "generalized" adoption of cloud computing paradigm in software industry, together with the industry's high competitiveness, results in a highly demand for new releases that make use of cloud computing platforms with quality but developed in lesser time. It is a common problem in software projects that the final product is misaligned with the stakeholders' needs. The stakeholders are responsible for the business model development, and the development team is responsible for implementing it in software. However, in many cases, there is no stable context for eliciting requirements in Cloud Computing projects, and requirements engineering (RE) for Software-as-a-Service (SaaS) [1] and Service-Oriented Architecture (SOA) [2] are major challenging. A proper alignment is not always easy and bad requirements are one of the main reasons of projects' failure [3]. The elicitation of product-level (service-based software) requirements is achievable by using a process-level perspective to elicit process or (business process) needs and then use the resulting artifacts, like an information system logical architecture, as inputs for modeling of software functional needs. 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 [4]. There are similar approaches that tackle the problem of aligning domain-specific needs with software solutions. For instance, goal-oriented approaches are a way of doing so, but they do not encompass methods for deriving a logical representation of the intended system processes with the purpose of creating context for eliciting product-level requirements.

Our main problem, and the main topic this chapter addresses, is assuring that product-level (IT-related, in the software engineering domain) requirements are perfectly aligned with process-level requirements (in the information systems domain), 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 Sect. 11.2. These requirements may be supported by analysis models, that are implementation agnostic [5]. According to [5], the existing approaches for transforming requirements into an analysis model (i) do not require acceptable user effort to document requirements, (ii) are efficient enough (e.g., one or two transformation steps), (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. For that, requirements are modeled by successive derivation (for more details, please refer to our approach of a V-Model [6, 7]) using UML models, first in process-level perspective, and then in product-level perspective.

Our proposal is to provide context for RE for cloud computing projects, by using a process-level approach for the initial eliciting of business needs, in order to give context to the product-level functionalities elicitation. Our product-level approach

includes the use of models to define functional and nonfunctional requirements for SaaS and SOA solutions, initially in form of UML use cases, and in deriving a service-oriented logical architecture by executing the Four-Step-Rule-Set (4SRS) method [6–8]. This paper intends to detail the steps and rules required to perform the transition between the process- and product-level perspectives within the V+V Model (presented in [9]) using Query/View/Transformation (QVT) [10] in order to achieve that transition between UML models. This way, we formalize the transition between perspectives that is required in order to align the requirements of both V-Models. In comparison with [9], besides the use of QVT transformations, we include additional contributes to support of the rules, like a formalization of a UML metamodel extension. In addition, we strengthen the state-of-the-art section. The result is an integrated approach, beginning in information system architecture and ending in a service-oriented logical architecture.

This chapter will be structured as follows: Sect. 11.2 briefly presents the macro-process for information systems development based on both process- and product-level V-Model approaches; Sect. 11.3 describes the transition steps and detail the model transformations required for applying the transition rules between both perspectives; in Sect. 11.4 we present a real industrial cloud-based demonstration case on the adoption of transition steps between process- and product-level perspectives; in Sect. 11.5 we compare our approach with other related work; and in Sect. 11.6 we present the conclusions.

11.2 The V+V Model

The V+V Model [11] is an approach for information systems development. The entire V+V Model is not presented in this paper, since it is already detailed in [11] and its composing artifacts presented in [6, 7]. Rather, in Fig. 11.1 is depicted the main artifacts, and those artifacts are the ones involved in the transition process. The transition process is presented in [9]. 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 Sect. 11.5) typically apply to a single perspective.

The first V-Model (in which the most important artifacts are depicted in the left side of Fig. 11.1, the remaining models are out of the scope of this paper) is executed at a process-level perspective performing the identification of business needs and then, by successive artifact derivation, transiting from business-level artifacts (i.e., process-level use case diagrams) to an IT-level artifact (i.e., information system logical architecture) that is assured by the execution of the Four-Step Rule-Set (4SRS) method. For the scope definition of our work, we characterize our

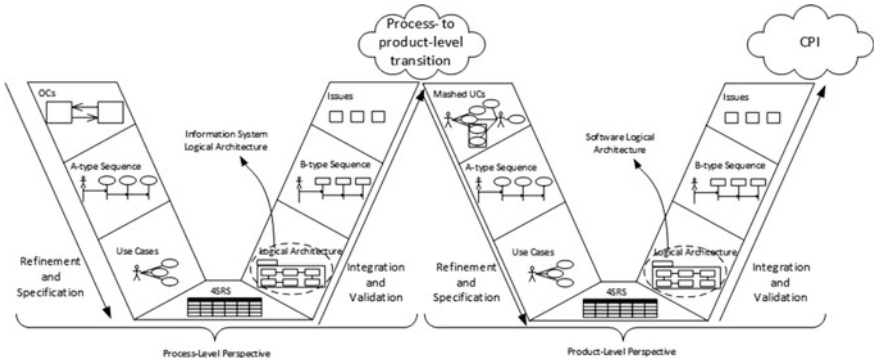

Fig. 11.1 V+V process framed in the development macro-process (from [11])

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 [12]. The process-level 4SRS method execution (see [6–8] 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. Use cases are mandatory to execute the 4SRS method.

The second V-Model (in which the most important artifacts are depicted in the right side of Fig. 11.1) is executed at a product-level perspective. By product-level, we refer as the typical software requirements. The second execution of the V-Model is performed by gathering information from the process-level V-Model in order to create a new model referred as *Mashed UCs* (preliminary product-level use case models). The creation of this model is detailed in the next section of this paper as transition steps and rules. *Mashed UC* model is then used as input for successive artifact derivation until requirements are modeled in product-level use case diagrams that gather typical software user requirements. The remaining models from Fig. 11.1 are out of the scope of this paper. Like in the first V-Model, use cases are input for the 4SRS method (but in its product-level perspective, detailed in [13–15]), which then outputs a service-oriented logical architecture that depict system requirements derived from the original user requirements. 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 nonfunctional requirements representation.

As depicted in Fig. 11.1, 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 UC* models). The result of the second V-Model (product-level) execution is the service-oriented logical architecture. The *Mashed UC* model is the output of the model transformations presented in the next section.

11.3 Steps and QVT Rules for Transition Between V-Models

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 use cases (UCpt's) are aligned with the architectural elements (AEpc's) from the information system logical architecture diagram; 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. To allow the recursive execution of the 4SRS method [13, 15–17], the transition from the first V-Model to the second V-Model must be performed by a set of steps. The output of the first V-Model must be used as input for the second V-Model; i.e., we need to transform the information system logical architecture into product-level use case models. The transition steps to guide this mapping must be able to support a business to technology changing. These transition steps (TS), presented in [9], are depicted in Fig. 11.2 and are structured as follows:

TS1—Architecture Partitioning: By applying collapsing and filtering techniques as detailed in [13], it is possible to identify major groups of elements in the information system logical architecture that must be computationally supported by software. In this transition step, the AEpc's under analysis are classified by their computation execution context with the purpose of defining software boundaries to be transformed into UCpt's. The final software boundary is represented after the execution of filtering and collapsing techniques in the AEpc's. Each of the identified major groups of elements is subject to a separate execution in the following transition steps.

TS2—Use Case Transformation: This transition step is applied to each partition defined in the previous transition step (i.e., to each major groups of elements) with

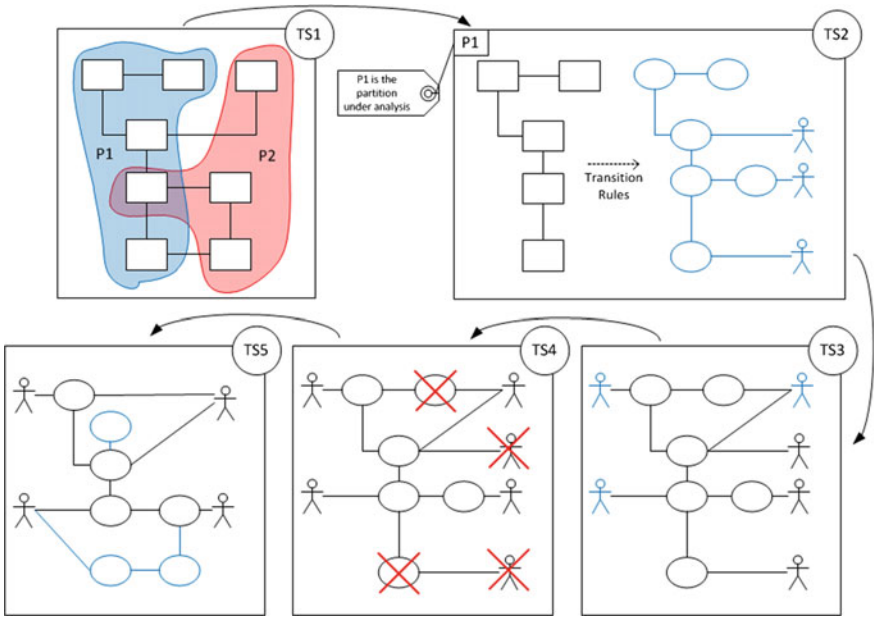

Fig. 11.2 Process- to product-level transition

the purpose of transforming elements of the information system logical architecture (AEpc's) into software use cases and actors. In this transition step, AEpc's are transformed into software use cases and actors that represent the system under analysis. This is the most critical transition step of the transition process and, as such, we have devised a set of transition patterns that must be applied as rules that are later described in this section.

TS3—Original Actors Inclusion: For each defined partition, the original actors that were related to the use cases from which the architectural elements of the process-level perspective are derived (in the first V-Model execution) must be included in the representation. The purpose of this transition step is to introduce into the product-level perspective the necessary information regarding the skills and stakeholders of the originally defined processes. The traceability between the process-level use cases and the AEpc's is assured by the process-level 4SRS execution [6–8].

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

TS5—Gap Filling: This final transition step intents to create, in the form of use cases to be added to the model, the necessary information of any requirement that is intended to be part of the design and that is not yet present. Typical missing use

cases are connections between existing use cases that were automatically created by the transition rules.

During the execution of these transition steps, a specific stereotype of use cases, called 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 first 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, 7]). Nevertheless, this information is not enough to properly identify the partitions. Information gathered in scenarios that were elicited in early models in the first V-Model must also be accounted. A partition is created by identifying all the relevant architectural elements that belong to a given organizational configuration scenario. The rules to support the execution of the TS2 are applied in the form of transition rules and must be applied in accordance to the stereotype of the envisaged architectural element. There are three stereotyped architectural elements: *d-type*, which refer to generic decision repositories (data), representing decisions not supported computationally by the system under design; *c-type*, which encompass all the processes focusing on decision-making that must be supported computationally by the system (control); and *i-type*, which refer to process' interfaces with users, software, or other processes. The full descriptions and specifications of the three stereotypes are available in [6].

The proposed process not only includes activities for perspective transition (as it is performed by the application of transition rules in TS2) but it also concerns to obtain a stable model (by performing TS3-5). By analyzing the perspectives on which the steps from the transition process are performed, the steps are easily classified.

The transition process naturally starts in the process-level perspective with AEpc's. In Table 11.1 it is possible to realize that after TS1 the transition is still dealing with AEpc's as input; the execution of TS2 results in the perspective transition, since it is in this TS that UCtr's are introduced and they relate to product-level; in the remaining transition steps, naturally they relate to product-level perspective. The purpose of the remaining transition steps 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 TS we consider the resulting model as containing product-level (software) use cases (UCpt's), which will compose the *Mashed UC* model. In summary, in TS1 the artifact regards AEpc's, in TS2-5 the focus is in UCtr's and only when the Mashed UC model is finished UCtr's become UCpt's.

Table 11.1 Transition steps overview

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

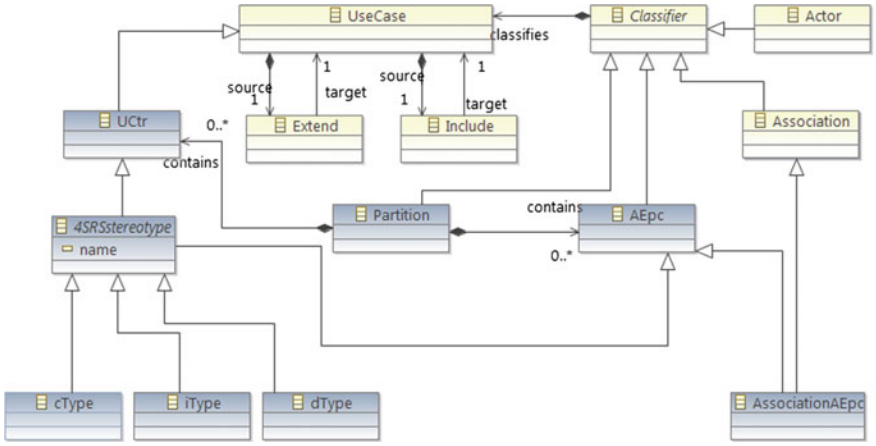

Fig. 11.3 Excerpt of AEpc and UCtr extension

For the sake of understandability we present in Fig. 11.3 an excerpt of the UML extension that supports the creation of AEpc's, UCtr's and partitions (please note that UCpt's regard the traditional use cases). We consider that a partition is a container of AEpc's or UCtr's and acts as a border delimiter for the combinations of possible systems candidates to be analyzed.

After delimiting all the partitions, it is necessary to focus on a particular one (called inbound partition) and execute the required transformations considering all the remaining neighbor partitions (outbound partitions).

A proper way of defining the transformations between models is by means of using OMG's QVT [10]. QVT is a set of languages (QVT-Operational, QVT-Relations, and QVT-Core) that enables models transformations. QVT-Operational enables unidirectional transformations of a given model into another. QVT-Relations allow bi-directional transformations. QVT-Core can be considered a subset of QVT-Relations. All the QVT set of languages are associated with model-driven approaches. These model-driven approaches are usually associated with design and implementation models and lack support to requirements and analysis models. The requirements specification (in any perspective) is a crucial task in any software development process. As such, models that support requirements specification should be integrated into model-driven methods.

In our proposed approach we have chosen QVT as a mean to transform AEpc's models into UCtr's models, or being more specific, transforming information system logical architectural models into *Mashed UC* models. This relates to integrating models that support requirements specifications into a model-driven approach. In [9], the steps and rules were already described, but without technological formalization. Associated with the transition rules, we present a subset of the QVT-Operational (-like) code that supports the transformation intended by a given rule. The defined transition rules, from the logical architectural diagram to the *Mashed UC* diagram, are presented in [9] and are as follows:

TR1—an inbound *c-type* or *i-type* AEpc is transformed into an UCtr of the same type (see Fig. 11.4). By inbound we mean that the element belongs to the partition under analysis.

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

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

TR2—an inbound d-type AEpc is transformed into an UCtr and an associated actor (see Fig. 11.5). This is due to the fact that d-type AEpc's correspond to decisions not computationally supported by the system under design and, as such, it requires an actor to activate the depicted process.

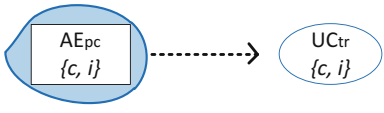

Fig. 11.4 TR1—transition rule 1 (from [9])

Fig. 11.5 TR2—transition rule 2 (from [9])

TR2 is supported by the following:

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

Rules TR1 and TR2 are the most basic ones and the patterns they express are the most used in the transition step 2. The remaining rules regard more specific situations, however require equal attention from the analyst. The remaining rules are as follows:

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 responsible for outbound actions associated with *UCtrx* (Fig. 11.6).

The specification for TR3 is:

```
if (AEpc.Partition=multiple) and
(AEpc.4SRSstereotype=cType) then {
UCtr.name:=AEpc.name;
UCtr.4SRSstereotype:=AEpc.4SRSstereotype;
Actor.name:=self.name;
Actor.association:=UCtr }
endif;
```

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

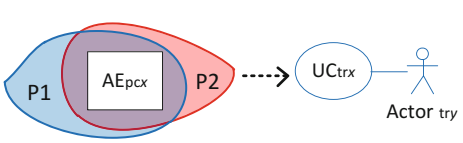

Fig. 11.6 TR3—transition rule 3 (from [9])

Fig. 11.7 TR4—transition rule 4 (from [9])

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. 11.7). 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 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.

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.

Regarding TR4, the necessary specification is

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

TR5—an inbound UCtr of name *x* with a connection to an outbound AEpc of name *y* (note that this is still an AEpc, since it was not transformed into any other concept in the previous transition rules) gives place to both an UCtr named *x* and to an actor named *y* (see Fig. 11.8). 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

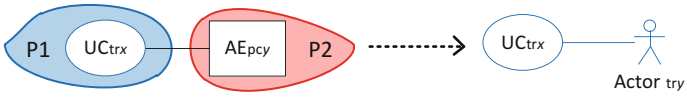

Fig. 11.8 TR5—transition rule 5 (from [9])

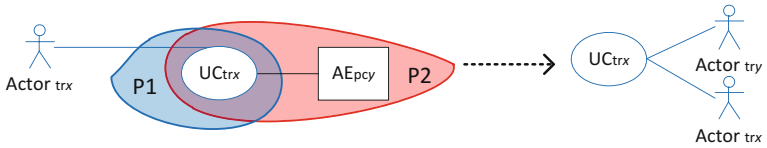

Fig. 11.9 TR5.1—transition rule 5.1 (from [9])

required external inputs to the inbounds UCtr's created during application of TR1, TR2, or TR3.

For TR5, the supporting specification is

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

A special application of TR5 (described as TR5.1) can be found in Fig. 11.9 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.

The application of these transition steps and rules to all the partitions of the information system logical architecture gives origin to a set of Mashed UC models. In the next section, we present a demonstration case study an information system logical architecture is transformed into a product-level *Mashed UC* model by executing the transition steps.

11.4 Demonstration Case on the Transition Process

The applicability of the proposed approach was assessed with a real project that is analyzed in this manuscript as a case study: the ISOFIN project (Interoperability in Financial Software) [18]. 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, e.g., Banks, Insurances, and others.

The global ISOFIN architecture relies on two main service types: Interconnected Business Service (IBS) and Supplier Business Service (SBS). In this context, there are two external business domain entities with access to the ISOFIN Platform:

ISOFIN Customers and ISOFIN Suppliers. An ISOFIN Customer is an entity whose domain of interactions resides in the scope of consuming, for economic reasons, the functionalities exposed by IBSs. An ISOFIN Supplier is a company that interacts with the ISOFIN SaaS Platform by supplying the platform with functionalities (SBSs) that reside in their private clouds. IBS's 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. SBSs are made available in the ISOFIN Supplier private cloud by the use of generators and are composed, in the public cloud where the ISOFIN SaaS Platform resides implement an IBS. Composition of basic SBSs into IBSs give origin to more powerful functionalities that are exposed by the platform.

The requirements elicitation of activities in the ISOFIN project resulted in a model composed by 39 use cases (i.e., the process-level use cases from Fig. 11.1). From the demonstration case, we first present a subset of the information system logical architecture in Fig. 11.10, that resulted from the execution of the 4SRS method at a process-level perspective [6–8]; 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 within the ISOFIN platform. The first V-Model execution ended with 74 documented architectural elements (not counting associations). This means that we added more details to the problem description. All of these architecture elements from the logical architecture were input for the transition process.

The logical process-level architecture of the ISOFIN project has embedded design decisions that are initially injected in the processes descriptions. The design decisions concern the deployment of the system in a public cloud environment and

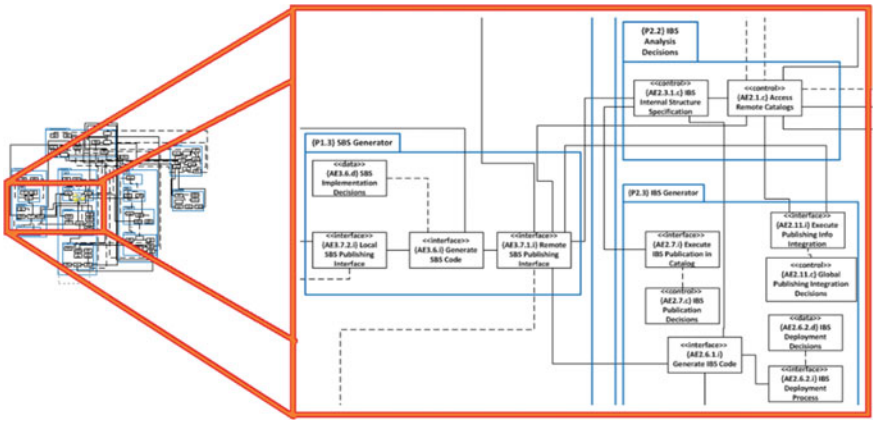

Fig. 11.10 Subset of the ISOFIN information system logical architecture (from [9])

its interoperability with several other private clouds as defined in the project objectives.

The resulting logical model of the system architecture, based on the processes that are intended to be executed, shows a software solution able to be deployed in IaaS layer. That layer will support the execution of a set of services that will allow suppliers to specify the behavior of the services they intend on supplying, in a PaaS layer. This will allow customers, or third-parties, to use the platform's services, in a SaaS layer and be billed accordingly. Additionally, processes regarding the provider perspective (e.g., infrastructure management) were also considered.

In Fig. 11.11, we depict the execution of TS1 to a subset of the entire information system logical architecture, 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.

TS1 ends with the collapsing of AEpc's from outside the boundaries and without any associations to inbound AEpc's. In the subset of Fig. 11.11, such only applied to *{AE3.7.2.i} Local SBS Publishing Interface*. Thus, this AEpc is immediately excluded from the remaining steps.

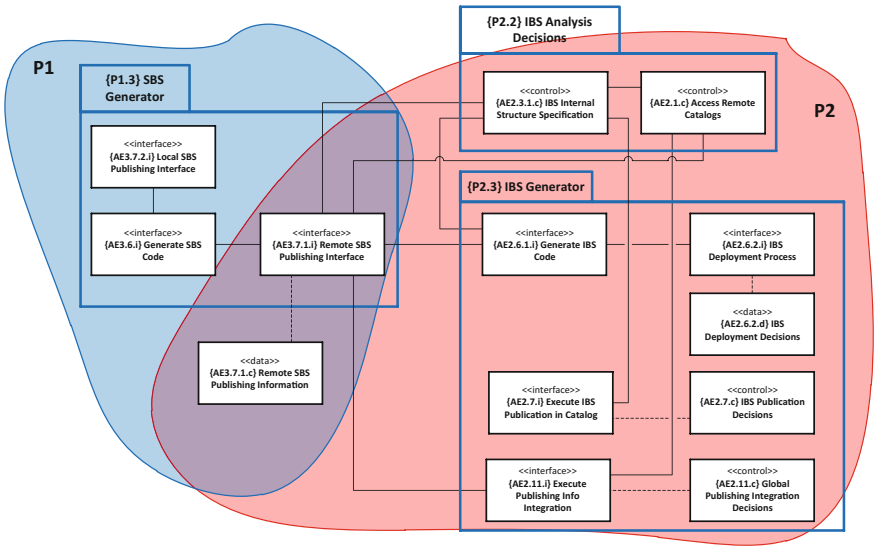

Fig. 11.11 Partitioning of the information system logical architecture (TS1) (from [9])

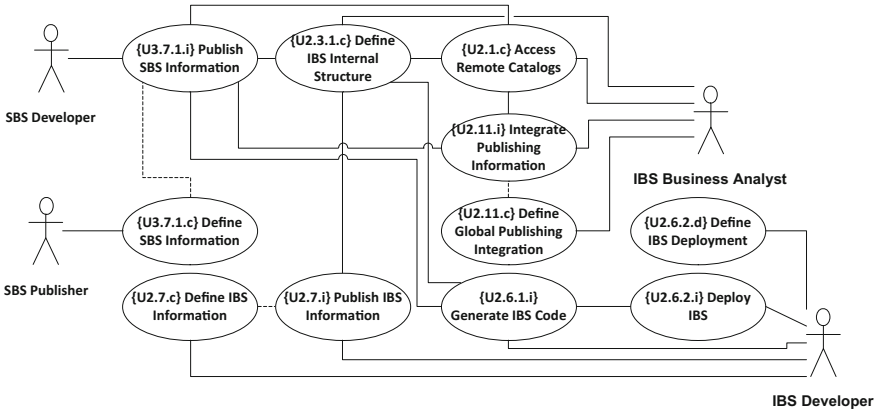

Fig. 11.12 *Mashed UC* model resulting from the transition from process- to product-level (from [9])

In Fig. 11.12, 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.

Table 11.2 summarizes the application of QVT transformations to the AEp'cs from Fig. 11.11. 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.

After the execution of the transition steps, the resulting *Mashed UC* model is the first artifact that composes the product-level V-Model and that is then used as starting point for the rest of the V-Model execution. Thus, after performing the

Table 11.2 Executed QVT transformations to the model

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

transition process, the *Mashed UC* model was used for deriving a new set of artifacts, this time regarding a product-level perspective, in a new V-Model execution. Like in the process-level perspective, the process ends with the 4SRS method execution, where a service-oriented logical architecture was derived.

The derivation of this architecture is out of the scope of this paper, so this demonstration case skips directly from the *Mashed UC* model to the software system logical architecture. We depict in the second architecture of Fig. 11.13 the entire software system logical architecture obtained after the execution of the V+V process, 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. 11.13) previously presented.

The software system logical architecture is composed by architectural elements (depicted in the zoomed area) that represent services 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 service-oriented logical architecture as complex as the ISOFIN platform (the overall

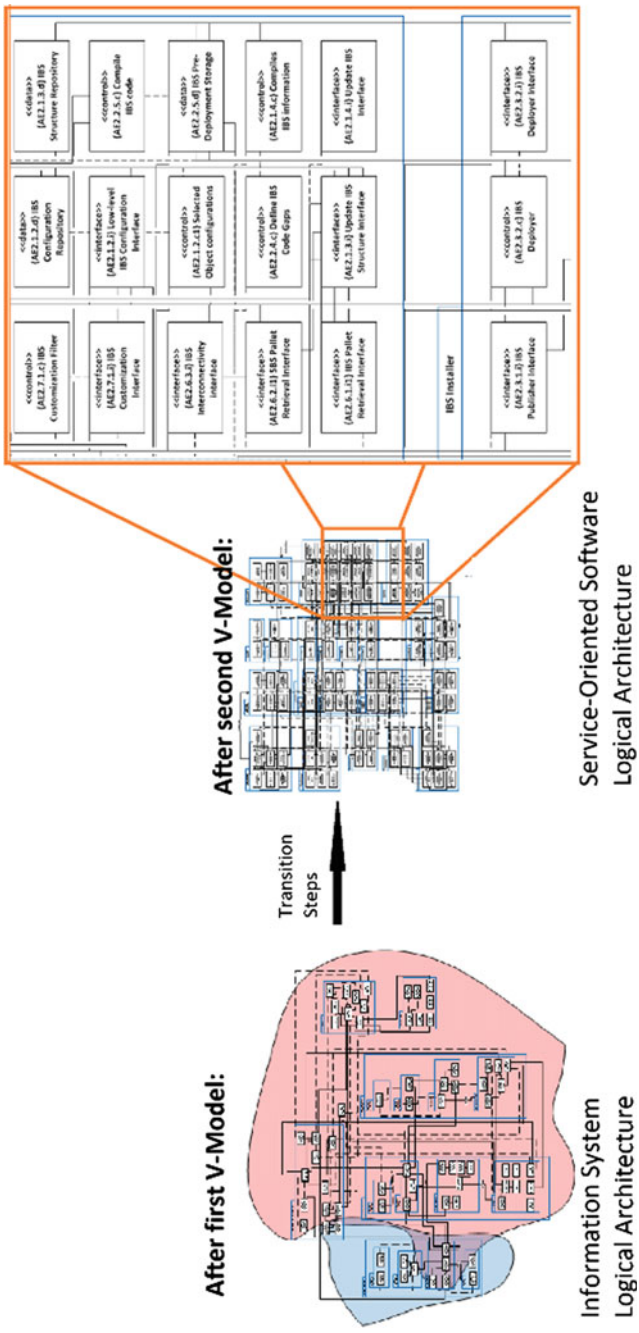

Fig. 11.13 Subset of the ISOFIN service-oriented logical architecture based on the information system logical architecture (from [9])

information system logical architecture was composed by near 80 architectural elements, and the resulting service-oriented logical architecture by near 100) by adopting an approach that only considers the product-level perspective.

Our V+V process allows to perform RE activities in an integrated approach, and the ISOFIN project demonstration case (namely the service-oriented logical architecture) demonstrated that this approach is suitable for Cloud Computing projects. The services that compose this SOA-based platform were identified, by performing sequential RE-related tasks and requirements modeling. It is also possible to depict in Fig. 11.13 the alignment (supported by the transition steps and QVT rules) between the architecture elements in both perspectives.

11.5 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 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 [19]. This approach uses two step-based methods, which output a formalization of the architecture requirements for each method. Since it uses two methods, each of the derived architectures provides a different view of the system. It is acknowledged in software engineering that a complete system architecture cannot be represented using a single perspective [20]. 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. An important view considered in our approach regards the architecture. The organization's processes can be represented by an enterprise architecture, as proposed in [21], and representation extended by including in the architecture modeling concerns as business goals and requirements [22]. However, such proposals do not intend to provide information for implementation teams during the software development process, but instead to provide to stakeholders with business strategic requirements. Most agree that an architecture concerns both structure and behavior, with a level of abstraction that only regards significant decisions, is influenced by its stakeholders and the environment where it is intended to be instantiated and also encompasses decisions based on some rationale or method. Some architecture views can be seen in [20, 23–25]. Krutchen's work [20] 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. Our stereotyped usage of sequence diagrams adds more representativeness value to the specific model. 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 describes. Ullah and Lai [26] models business goals and derives system requirements, but it outputs a UML state chart. Our approach outputs system requirements in an architectural diagram and stereotyped sequence diagrams.

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. Tarafdar and Qrunfleh [27] argues that an alignment between business and IT can be "strategic" and "tactical," and [28] presents an alignment approach also based on architectural models. An approach that enacts the alignment between domain-specific needs and software solutions, is the GQM + Strategies (Goal/Question/Metric+Strategies) [29]. This approach uses measurement to explicitly link goals and strategies from business objectives to project operations. Another goal-oriented approach is the Balanced Scorecard (BSC) [30]. 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, and can be properly aligned with four key elements of IT-business alignment (integrated planning, effective communication, active relationship management, and institutionalized culture of alignment) [31], as well as for information security management [32]. Another approach, COBIT [33], 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. 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.

The 4SRS method is used for transforming functional user requirements into logical architectural models representing system requirements. It can be executed either in a process-level [7–9] and in a product-level perspective [13–15] but the method executed alone does not allow to transit between perspectives. Tan et al. [34] presents an approach to transform a functional analysis model (in a data flow diagram) into object-oriented design and implementation. This approach is executed in a product-level perspective and, like 4SRS, the transformation only regards a single perspective. In a product-level perspective, there are several approaches that support model transformations to software architectures based on requirements, like the work in [34], the Component-Oriented Platform Architecting Method for product family engineering (COPA) [35], the Reuse-driven Software Engineering Business (RSEB) [36], the Family-Oriented Abstraction, Specification and Translation (FAST) [37], the Feature-Oriented Reuse Method (FORM) [38], the

Komponentenbasierte Anwendungsentwicklung (german for component-based product line engineering—KobrA) [39], or the Quality-driven Architecture Design and Analysis (QADA) [40]. In a process-level perspective, Tropos [41] 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; afterwards, the SIRA approach describes a software requirements and architectural models from the perspective of an organization in the context of Tropos, using i* models (a goal-oriented approach to describe both the system and its environment in terms of strategic actors and social dependencies among them) [42], in [43] is presented a process to generate Acme ADL [44] architectural models from i* models; and in [45] is described a method for obtaining architectural models based in KAOS requirements models. None of these presented approaches support process- to product-level transition.

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.

In [46] 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 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 [47].

In literature, model transformations are often related to the Model-Driven Architecture (MDA) [48] initiative from OMG. An MDA-based approach uses model transformations in order to transform a high-level model (Platform-Independent Model—PIM) to a lower level model (Platform-Specific Model—PSM). MDA-based model transformations are widely used but, as far as the authors know, the supported transformations do not regard perspective transition, i.e., are perspective agnostic since they concern model transformations within a single perspective (typically the product-level one). Model-driven transformation approaches were already used for developing information systems in [49]. In [50] business process models are derived from object-oriented models.

The existing approaches for model transformation attempt to provide an automated or automatic execution. Yue et al. [5] provides a systematic review and evaluation of existing work on automating of transforming requirements into an analysis model and, according to the authors, none of the compared approaches provide a practical automated solution. The transition steps and rules presented in this work intent to provide a certain level of automation into our approach and improve the efficiency, validation, and traceability of the overall V+V process. The transitions depicted in the present work are able to be fully implemented in development tools that support QVT transformations, like the well-known Eclipse IDE.

11.6 Conclusions

We have described the transition steps and rules for assuring an alignment between process- and product-level requirements within the execution of the V+V process. This approach is adopted to create context for software implementation teams in Cloud Computing projects where requirements cannot be properly elicited. The V+V process is based on successive models construction and recursive derivation of logical architectures (first an information system one and then for a service-oriented one), and makes use of model derivation for creating use cases, based on high-level representations of desired system interactions.

We presented a real industry demonstration case in order to elicit requirements for developing a platform that provides interoperability between financial institutions by providing services in a cloud environment. Our approach is supported on a set of transition steps and QVT-based transition rules in order to execute the transition from process- to product-level perspective. These transition steps use as basis an information system logical architecture to output a product-level use case model. The product-level requirements are specified in a software system logical architecture, having as basis the information system logical architecture.

It is a common fact that domain-specific needs, namely business needs, are fast changing. Information system architectures must be in a way that potentially changing domain-specific needs are local in the architecture representation. Our approach enables requirements traceability within three stages of its process, namely within the derivation of both process- (information system) and product-level (service-oriented) logical architectures and during the transition between perspectives. Each V-Model uses software engineering techniques, such as operational model transformations to assure the execution of a process that begins with business needs and ends with a logical architectural representation of a system. Each V-Model from our proposed V+V process encompasses the derivation of a logical architecture representation that is aligned with domain-specific needs by executing the 4SRS method and any change made to those domain-specific needs is reflected in the logical architectural model, and the transformation and traceability is properly assured by the 4SRS method. Since the *Mashed UC* model (and, consequently, the perspective transition) is derived from a model transformation based on QVT mappings (from AEpc's to UCtr's), traceability between AEpc's and UCpt's is guaranteed, thus any necessary change on product-level requirements due to a change on a given business needs is easily identified and propagated alongside the models that comprise the V+V process.

Since we are designing SOA and Cloud Computing solutions, using SOA Modeling Language (SoaML) diagrams, an instantiation of UML models for SOA contexts, instead of UML diagrams, may be more adequate. Within SoaML diagrams, 4SRS was already used to derive participants, requests, services, and properties by using UML use cases as input. As future work, we intend to include in our approach the derivation of other SoaML diagrams, like Service Contracts, Service Architectures, Interfaces, Service Choreographies, amongst others, in order

to improve our process, since we believe that using additional diagrams will improve the specification of the SOA and cloud services. However, SoaML notation should be used throughout the entire V-Model, in order to obtain information regarding services at the end of the requirements elicitation phase. We intend to develop this V-Model in future work, as we believe that the rationale regarding the RE-tasks present in the V-Model as described in this paper will be similar for developing a SoaML variation of the V-Model.

References

1. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. (2009).
2. Bianco, P., Kotermanski, R., Merson, P.: Evaluating a service-oriented architecture. (2007).
3. Standish Group: CHAOS Report 2014. (2014).
4. Maibaum, T.: On specifying systems that connect to the physical world. New Trends Softw. Methodol. Tools Tech. (2006).
5. Yue, T., Briand, L.C., Labiche, Y.: A Systematic Review of Transformation Approaches between User Requirements and Analysis Models. Requir. Eng. Vol. 16, (2011).
6. Ferreira, N., Santos, N., Machado, R., Fernandes, J.E., Gasević, D.: A V-Model Approach for Business Process Requirements Elicitation in Cloud Design. In: Bouguettaya, A., Sheng, Q. Z., and Daniel, F. (eds.) Advanced Web Services. pp. 551–578. Springer New York (2014).
7. Santos, N., Teixeira, J., Pereira, A., Ferreira, N., Lima, A., Simões, R., Machado, R.J.: A demonstration case on the derivation of process-level logical architectures for ambient assisted living ecosystems. In: Garcia, N.M. and Rodrigues, J.J.P.C. (eds.) Ambient Assisted Living Book. pp. 103–139. CRC Press (2015).
8. Salgado, C., Teixeira, J., Santos, N.: A SoaML Approach for Derivation of a Process-Oriented Logical Architecture from Use Cases. Explor. Serv. …. (2015).
9. Ferreira, N., Santos, N., Soares, P., Machado, R., Gašević, D.: A Demonstration Case on Steps and Rules for the Transition from Process-Level to Software Logical Architectures in Enterprise Models. In: Grabis, J., Kirikova, M., Zdravkovic, J., and Stirna, J. (eds.) The Practice of Enterprise Modeling. pp. 277–291. Springer Berlin Heidelberg (2013).
10. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), http://www.omg.org/spec/QVT/1.1.
11. Ferreira, N., Santos, N., Soares, P., Machado, R.J., Gasevic, D.: Transition from Process- to Product-level Perspective for Business Software, (2012).
12. Browning, T.R., Eppinger, S.D.: Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng. Manag. 49, 428–442 (2002).
13. Machado, R.J., Fernandes, J., Monteiro, P., Rodrigues, H.: Refinement of Software Architectures by Recursive Model Transformations, http://dx.doi.org/10.1007/11767718_38, (2006).
14. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of UML Models for Service-Oriented Software Architectures, (2005).
15. Fernandes, J., Machado, R., Monteiro, P., Rodrigues, H.: A Demonstration Case on the Transformation of Software Architectures for Service Specification. In: Kleinjohann, B., Kleinjohann, L., Machado, R., Pereira, C., and Thiagarajan, P.S. (eds.) From Model-Driven Design to Resource Management for Distributed Embedded Systems. pp. 235–244. Springer US (2006).
16. Azevedo, S., Machado, R.J., Muthig, D., Ribeiro, H.: Refinement of Software Product Line Architectures through Recursive Modeling Techniques, http://dx.doi.org/10.1007/978-3-642-05290-3_53, (2009).

17. Azevedo, S., Machado, R., Maciel, R.: On the Use of Model Transformations for the Automation of the 4SRS Transition Method. In: Bajec, M. and Eder, J. (eds.) Advanced Information Systems Engineering Workshops. pp. 249–264. Springer Berlin Heidelberg (2012).
18. ISOFIN: ISOFIN Research Project. http://isofincloud.i2s.pt/, (2010).
19. Jani, D., Vanderveken, D., Perry, D.: Experience Report: Deriving architecture specifications from KAOS specifications. (2003).
20. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Softw. 12, 42–50 (1995).
21. The Open Group: TOGAF—The Open Group Architecture Framework, http://www.opengroup.org/togaf/.
22. Engelsman, W., Quartel, D., Jonkers, H., van Sinderen, M.: Extending enterprise architecture modelling with business goals and requirements. Enterp. Inf. Syst. 5, 9–36 (2010).
23. Clements, P., Garlan, D., Little, R., Nord, R., Stafford, J.: Documenting software architectures: views and beyond, (2003).
24. Hofmeister, C., Nord, R., Soni, D.: Applied software architecture. Addison-Wesley Professional (2000).
25. Chen, D., Doumeingts, G., Vernadat, F.: Architectures for enterprise integration and interoperability: Past, present and future. Comput. Ind. 59, 647–659 (2008).
26. Ullah, A., Lai, R.: Modeling business goal for business/it alignment using requirements engineering. J. Comput. Inf. Syst. 51, 21 (2011).
27. Tarafdar, M., Qrunfleh, S.: IT-Business Alignment: A Two-Level Analysis. Inf. Syst. Manag. 26, 338–349 (2009).
28. Strnadl, C.F.: Aligning Business and It: The Process-Driven Architecture Model. Inf. Syst. Manag. 23, 67–77 (2006).
29. 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 (Long. Beach. Calif). 43, 57–65 (2010).
30. Kaplan, R.S., Norton, D.P.: The balanced scorecard–measures that drive performance. Harv. Bus. Rev. 70, 71–79 (1992).
31. Huang, C.D., Hu, Q.: Achieving IT-business strategic alignment via enterprise-wide implementation of balanced scorecards. Inf. Syst. Manag. 24, 173–184 (2007).
32. Herath, T., Herath, H., Bremser, W.G.: Balanced Scorecard Implementation of Security Strategies: A Framework for IT Security Performance Management. Inf. Syst. Manag. 27, 72–81 (2010).
33. Information Technology Governance Institute (ITGI): COBIT v5—A Business Framework for the Governance and Management of Enterprise IT. ISACA (2012).
34. Tan, H.B.K., Yang, Y., Bian, L.: Systematic Transformation of Functional Analysis Model into OO Design and Implementation. IEEE Trans. Softw. Eng. Vol. 32, (2006).
35. Obbink, H., Müller, J., America, P., van Ommering, R., Muller, G., van der Sterren, W., Wijnstra, J.G.: COPA: A component-oriented platform architecting method for families of software-intensive electronic products. In: Tutorial for the First Software Product Line Conference., Denver, Colorado. (2000).
36. Jacobson, I., Griss, M., Jonsson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison Wesley Longman (1997).
37. Weiss, D.M.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional (1999).
38. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Sw Eng. (1998).
39. Bayer, J., Muthig, D., Göpfert, B.: The library system product line. A KobrA case study. Fraunhofer IESE. (2001).
40. Matinlassi, M., Niemelä, E., Dobrica, L.: Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture. VTT Technical Research Centre of Finland (2002).

41. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Inf. Syst. (2002).
42. Yu, E.: Modelling strategic relationships for process reengineering. In: Yu, E., Giorgini, P., Maiden, N., and Mylopoulos, J. (eds.) Social Modeling for Requirements Engineering. pp. 11–152. The MIT Press (2011).
43. Lucena, M., Castro, J., Silva, C., Alencar, F., Santos, E., Pimentel, J.: A model transformation approach to derive architectural models from goal-oriented requirements models. In: OTM Confederated International Conferences“ On the Move to Meaningful Internet Systems.” pp. 370–380. Springer Berlin Heidelberg. (2009).
44. Garlan, D., Monroe, R., Wile, D.: Acme: an architecture description interchange language. CASCON First Decad. High Impact Pap. (2010).
45. Lamsweerde, A. Van: From system goals to software architecture. Form. Methods Softw. Archit. (2003).
46. Dijkman, R.M.: Deriving use case diagrams from business process models. Tech. report, CTIT Tecnhical Rep. (2002).
47. Winter, R., Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture, (2006).
48. OMG: MDA Guide Version 1.0.1, (2003).
49. Iribarne, L., Padilla, N., Criado, J., Asensio, J.-A., Ayala, R.: A Model Transformation Approach for Automatic Composition of COTS User Interfaces in Web-Based Information Systems. Inf. Syst. Manag. 27, 207–216 (2010).
50. Redding, G., Dumas, M., Hofstede, A.H.M. ter, Iordachescu, A.: Generating Business Process Models from Object Behavior Models. Inf. Syst. Manag. 25, 319–331 (2008).