Refinement of Software Architectures by Recursive Model Transformations*

Ricardo J. Machado[1], João M. Fernandes[2],
Paula Monteiro[1], and Helena Rodrigues[1]

[1] Dept. of Information Systems
[2] Dept. of Informatics
University of Minho, Portugal

Abstract. The main aim of this paper is to present how to refine software logical architectures by application of a recursive model-based transformation approach called 4SRS (four step rule set). It is essentially based on the mapping of UML use case diagrams into UML object diagrams. The technique is based on a sequence of steps that are inscribed in a tabular representation that is used to derive the software architecture for a focused part of the global system.

1 Introduction

The most complex activity during development of software systems is probably the transformation of a requirement specification into an architectural design [1]. The other phases have also their challenges, but they are better understood and a variety of methods, languages and tools are available to support the software engineer.

The process of designing software architectures is, by far, less formalised and often is greatly an intuitive ad-hoc activity, poorly based on engineering principles. Since the architecture of a software system constrains the space solution, the design decisions taken during architectural design must be made with great care, since they typically have a large impact on the quality of the resulting system.

An architectural transformation approach, called 4SRS (four step rule set), is presented that employs successive transformations of the software architecture, in order to satisfy the elicited user requirements. It is essentially based on the mapping of UML use case diagrams into UML object diagrams. The iterative nature of the approach and the usage of graphical models are important issues to guarantee that the final architecture reflects the user requirements.

Fig. 1 illustrates the recursive application of the 4SRS technique. This paper addresses the problem of deriving the logic architecture of a given platform service (called service object diagram), from a functional refinement of the platform architectural model (called platform object diagram), by adopting a recursive version of the 4SRS technique. The first 4SRS execution supports the platform requirements analysis by generating one platform object diagram that corresponds to the logic architecture of the system (this first 4SRS execution is described in detail in [2]). The second

* This work has been supported by projects STACOS (FCT/POSI/CHS/48875/2002) and USE-ME.GOV (IST-2002-002294).

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 422 – 428, 2006.

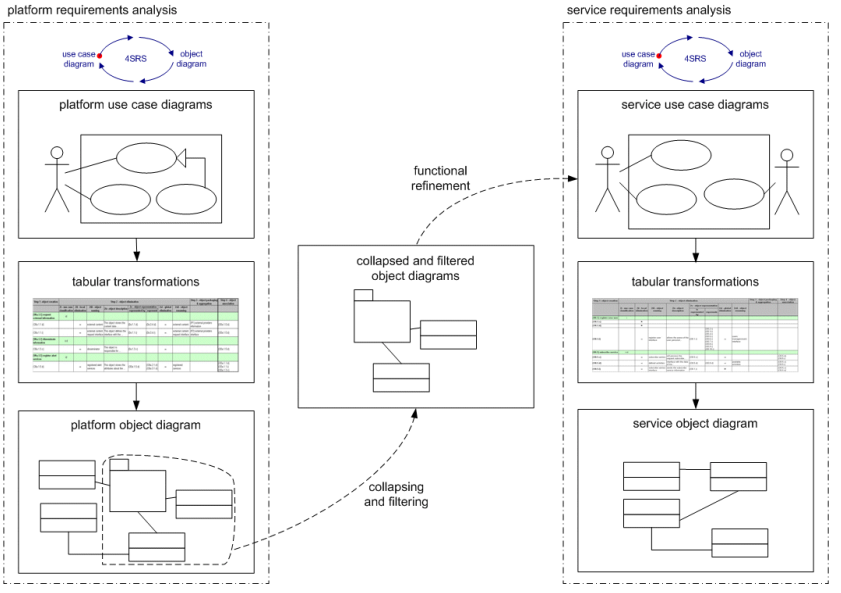

Fig. 1. Service specification with recursive 4SRS execution

4SRS execution supports the service requirements analysis by generating one service object diagram that corresponds to the logic architecture of the service to be specified (this second 4SRS execution is the aim of this paper).

The applicability of this technique is illustrated by presenting some results from a mobile application. For mobile applications, the definition of the underlying service oriented software architecture must consider as user requirements the services themselves, the mobile operators entry points and the final clients interfaces, and use them to characterize the platform. Within the presented demonstration case, the specification of one service of the mobile application was obtained by recursively applying the 4SRS technique.

2 Four Step Rule Set

4SRS is a technique proposed to transform users requirements into architectural models representing system requirements [3, 4]. It associates, to each object found during the analysis phase, a given category: interface, data, control. Each one of these categories is related to one of the three orthogonal dimensions, in which the analysis space can be divided (information, behaviour and presentation) [5].

For readability purposes, a brief description of the 4SRS technique is next presented. There is a complete description of its usage to obtain, in a non-recursive approach, the first logical architecture of the demonstration case used in this paper in [2]. In [6], an alternative version of the 4SRS technique is described for deriving the logical architecture for software product lines. This variant of the 4SRS technique

deals with variability at functional and architectural levels; the IESE GoPhone demonstration case [7] was adopted to experiment the approach.

The 4SRS technique is organized as four steps to transform use cases into objects: (1) object creation, (2) object elimination, (3) object packaging & aggregation, and (4) object association. After the execution of the 4 steps of the 4SRS technique, we obtain the logic architecture for the system that captures all its functional and non-functional requirements. An object model shows how significant properties of a system are distributed across its parts.

Fig. 2 shows the filtered object diagram that was obtained by using collapsing and filtering techniques described in [2] by considering package *{P5}* as one sub-system for design. This diagram was included here as an example of how raw object diagrams can be used during the development process to stress parts of the system and allow sub-system specification and partition of sub-projects among various teams.

In this paper, we consider the refinement of package *{P5}* that has given origin to the *AVAccess* service (the *service object diagram* depicted in fig. 1).

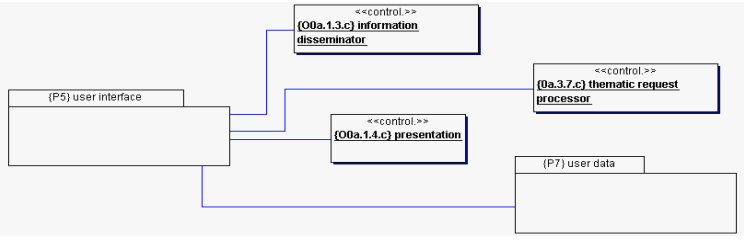

Fig. 2. Filtered object diagram for package {P5} service derivation

3 Recursive Architectural Refinement

{P5} can be considered as the system to be designed and apply, once more, the 4SRS technique to support its architectural refinement (within fig. 2). The recursive approach of the 4SRS technique suggests the construction of a new use case diagram (called *service use case diagram*, in fig. 1) that captures the users requirements of the new (sub-)system to refine. From this use case diagram the corresponding raw object diagram is derived (called *service object diagram*, in fig. 1). This proposed approach contrasts with the dominant one that suggests the application of design patterns to impose into the logical architecture a particular already proven reference architectural model [8, 9]. Our proposal does not reject this pattern-oriented view, only defers it into latter stages of development, allowing a previous functional refinement of requirements at architectural level, taking into account the specific aspects of the particular sub-system to be designed. The use case diagram depicted in fig. 3 was constructed for supporting the architectural refinement of *{P5}* to obtain the raw object diagram of the *AVAccess* service. This service constitutes the example considered in this paper to show the recursive application of the 4SRS technique. All the external entities (UML actors) existent in this diagram correspond to architectural elements connected to package *{P5}* in fig. 2. Object *{O0a.1.3.c}* in fig. 2 did not give origin to

any actor in fig. 3, because the architectural refinement of package *{P5}* did not consider the functionality that is associated with that object. The *user* actor is present in fig. 3, since it was already connected to the use cases that gave origin to the objects inside package *{P5}*, during the development process described in [2]. Actors in fig. 3 must be viewed as external sub-systems (components), from the point of view of the *AVAccess* service. To attain a better actor semantics within the associations with the obtained use cases, the actor *{O0a.3.7.c}* in fig. 3 was specialized into two different actors: *Application System Context Aggregation Service* and *Application System Service Repository*.

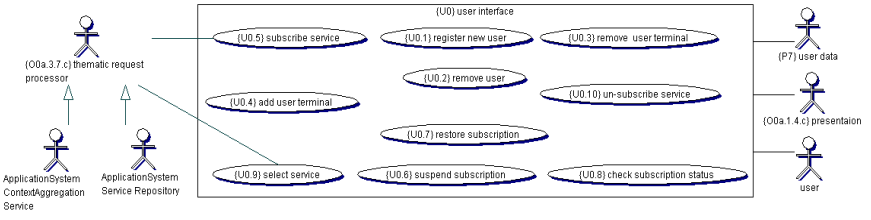

Fig. 3. Use case diagram for AVAccess service

4 Tabular Transformations

The execution of the 4SRS transformation steps can be supported in tabular representations. Moreover, the usage of tables permits a set of tools to be devised and built so that the transformations can be partially automated. These tabular representations constitute the main mechanism to automate a set of decision assisted model transformation steps. The 4SRS has been used both in academia and in industry [3, 4, 10] and has demonstrated to be agile in helping software engineers to find and refine architectural requirements, based on the elicited user requirements.

The table for the transformation steps is organized as follows: (1) each (micro-)step gives origin to one column; (2) each object gives origin to one row.

The 1st column corresponds to the execution of step 1. The first row allows the insertion of both the reference and the name of the use case. The next three rows allow the insertion of one interface, one data, and one control objects for the corresponding use case. For the demonstration case, there is no use case refinement, so step 1 is applicable to all (10) use cases in fig. 3, which gave origin to 30 objects. Fig. 4 depicts 4 different rows for each of the two previously exemplified use cases.

The 2nd column corresponds to the execution of micro-step 2i. In this micro-step, the software engineer classifies each use case as one of the 8 different combinations or patterns (Ø, i, c, d, ic, di, cd, icd). The idea behind this classification is to help on the transformation of each use case into objects. This classification would provide hints on which object categories to use and how to connect those objects. For the demonstration case, *{U0.1}* was classified as type "i", which means that only the interface object is kept (the control and data objects will be eliminated in micro-step 2ii), and *{U0.5}* was classified as type "icd", which means that all objects are kept.

The 3rd column corresponds to the execution of micro-step 2ii. The aim of this micro-step is to answer if each object created in step 1 makes sense in the problem domain, since the creation of objects in step 1 was blindly executed, not considering the system context for the object creation. Object that are to be eliminated are marked with "x" and objects that are to be kept are marked with "-". For the demonstration case, *{U0.1}* got two of its originated objects eliminated, since they do not make sense in the problem domain. *{U0.1}* is only responsible to send the new user information from the user to other sub-systems and vice-versa, which means that data and control dimensions are not within the scope of this use case.

Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object association
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation		2vi - global elimination	2vii - object renaming		
					is represented by	represents				
{U0.1} register new user	i									
{O0.1.c}		x								
{O0.1.d}		x								
{O0.1.i}		-	register user interface	allows the parse of the user personal...	{O0.1.i}	{O0.2.i} {O0.3.i} {O0.4.i} {O0.5.i} {O0.6.i} {O0.7.i} {O0.8.i} {O0.9.i} {O0.10.i}	-	users managemment interface		
{U0.5} subscribe service	icd									
{O0.5.c}		-	subscribe service	will process the request subscribe...	{O0.5.c}		-			{O0.5.d} {O0.5.i}
{O0.5.d}		-	defined activities	interface with the data of the...	{O0.5.d}	{O0.9.d}	-	available activities		{O0.5.c} {O0.5.i}
{O0.5.i}		-	subscribe service interface	sends the subscribe service information...	{O0.1.i}		x			{O0.5.c} {O0.5.d}

Fig. 4. Table for 4SRS transformations

The 4th column corresponds to the execution of micro-step 2iii. In this micro-step, objects that have not been eliminated from the previous micro-step must receive a proper name that reflects both the use case from which it is originated and the specific role of the object, taking into account its main component. For the demonstration case, object *{O0.1.i}*, for instance, was named *register user interface*.

The 5th column corresponds to the execution of micro-step 2iv. Each named object resulting from the previous micro-step must be described, so that the system requirements they represent become included in the object model. These descriptions must be based on the original use case descriptions.

The 6th and 7th columns correspond to the execution of micro-step 2v. This is the most critical micro-step of the 4SRS technique, since it supports the elimination of redundancy in the user requirements elicitation, as well as the discovering of missing requirements. The "*is represented by*" column stores the reference of the object that will represent the object being analyzed. If the analyzed object will be represented by itself, the corresponding "*is represented by*" column must refer to itself. The "*represents*" column stores the references of the objects that the object analyzed will represent. *{O0.1.i}* does not delegate in other objects its representation (i.e. it is represented by itself) and it additionally represents a considerable list of other objects (each one of these objects must refer to *{O0.1.i}* in their columns "*is represented by*").

The 8th column corresponds to the execution of micro-step 2vi. This is a fully "automatic" micro-step, since it is based on the results of the previous one. The ob-

jects that are represented by other ones must be eliminated, since its system requirements no longer belong to them.

The 9th column corresponds to the execution of micro-step 2vii. Its purpose is to rename the objects that have not been eliminated in the previous micro-step and that represent additional objects. The new names must reflect the plenitude of system requirements. For the demonstration case, object *{O0.1.i}* was renamed *users management interface* to reflect the list of other objects that it additionally represents.

The 10th column corresponds to the execution of step 3. For the demonstration case, neither aggregations, nor packages were used, so column 10 remains unfilled.

The 11th column corresponds to the execution of step 4. For the demonstration case, the associations were solely derived from the use case classification executed in step 1. The classification of *{U0.5}* as type "icd" suggests the existence of three internal associations relative to the objects generated from the same use case. However, "id" association (between the interface and the data objects) was not allowed. Additionally, the following two tabular transformations imposed some constrictions to the object connectivity exercise: (1) in step 2v, it was decided that *{O0.5.i}* is represented by *{O0.1.i}*; (2) in step 2vi, {O0.5.i} was eliminated. These two decisions imply the existence of the following associations: (1) between *{O0.5.c}* and *{O0.5.d}*, suggested by the "icd" classification; (2) between *{O0.5.c}* and *{O0.1.i}*, due to the transitivity of the suggested association between *{O0.5.c}* and *{O0.5.i}* through the delegation executed by *{O0.5.i}* in *{O0.1.i}*.

5 Service Specification

Fig. 5 depicts the raw object diagram for the *AVAccess* service, obtained from the recursive application of the 4SRS technique.

The obtained raw object model (fig. 5) constitutes the canonical semantic reference for the service to be designed, since it has emerged from the software logical architecture of the platform by adopting a complementary functional refinement at architectural level. This architectural refinement has been explicitly executed within a component-based service development.

After obtaining this new architectural refined raw object model, the underlying service can be described through a set of diagrams as a means to specify the corresponding architectural component, namely, by designing a class diagram for the static characterization of the service component, a statechart for the life cycle characterization of the service, a set of activity diagrams for methods specification and a set of sequence diagrams for interface and protocol specification. These additional perspectives of the same service are not directly generated from the application of 4SRS technique, even though they are easier constructed after obtaining the raw object diagram of the service (fig. 5).

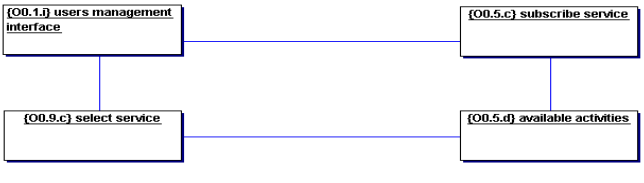

Fig. 5. Raw object diagram of the AVAccess service

6 Conclusions

The proposed recursive approach of the 4SRS technique suggests the construction of a new use case diagram that captures the users requirements of the new (sub-)system to refine a service. From this use case diagram the corresponding raw object diagram can be derive. This approach complements the usage of design patterns by allowing a previous functional refinement of requirements at architectural level, taking into account the specific aspects of the particular sub-system to be designed. This transformational approach shows that model continuity is a key issue and highlights the importance of having a well defined process to relate, map and transform requirements models. In the demonstration case, the 4SRS has allowed the specification of one particular service, taking into account all the architectural decisions previously taken to specify the platform where the service is intended to run, by assuring a continuous mapping between the platform and the service models.

References

1. J. Bosch, P. Molin. Software Architecture Design: Evaluation and Transformation. 7th IEEE Int. Conf. on the Engineering of Computer-Based Systems (ECBS'99), Nashville, Tennessee, U.S.A., pp. 4-10, IEEE CS Press, March, 1999.
2. R.J. Machado, J.M. Fernandes, P. Monteiro, H. Rodrigues. Transformation of UML Models for Service-Oriented Software Architectures. 12th IEEE Int. Conf. on the Engineering of Computer-Based Systems (ECBS 2005), Greenbelt, Maryland, U.S.A., pp. 173-182, IEEE CS Press, April, 2005.
3. J.M. Fernandes, R.J. Machado, H.D. Santos. Modeling Industrial Embedded Systems with UML. 8th IEEE/IFIP/ACM Int. Workshop on Hardware/Software Co-Design (CODES 2000), San Diego, California, U.S.A., pp. 18-22, ACM Press, May, 2000.
4. J.M. Fernandes, R.J. Machado. From Use Cases to Objects: An Industrial Information Systems. 7th Int. Conf. on Object-Oriented Information Systems (OOIS 2001), Calgary, Canada, pp. 319-328, Springer-Verlag, August, 2001.
5. I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.
6. A. Bragança, R.J. Machado. Deriving Software Product Line's Architectural Requirements from Use Cases: An Experimental Approach. 2nd Int. Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES 2005), Rennes, France, pp. 77-91, June, 2005.
7. D. Muthig, I. John, M. Anastasopoulos, T. Forster, J. Dörr, K. Schmid. GoPhone: A Software Product Line in the Mobile Phone Domain. IESE Technical Report no. 025.04/E, 2004.
8. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, 1996.
9. 9 R. Ahlgren, J. Markkula. Design Patterns and Organisational Memory in Mobile Application Development. 6th Int. Conf. on Product-Focused Software Process Improvement (PROFES2005),Oulu,Finland,pp.143-156,Springer-Verlag,June, 2005.
10. J.M. Fernandes, R.J. Machado. System-Level Object-Orientation in the Specification and Validation of Embedded Systems. 14th SBC/IFIP/ACM Symposium on Integrated Circuits and System Design (SBCCI 2001), Pirenópolis, Brazil, pp. 8-13, IEEE Computer Society Press, August, 2001.