A DEMONSTRATION CASE ON THE TRANSFORMATION OF SOFTWARE ARCHITECTURES FOR SERVICE SPECIFICATION *

João M. Fernandes[1], Ricardo J. Machado[2], Paula Monteiro[2], Helena Rodrigues[2]
[1] Dept. Informática & [2] Dept. Sistemas de Informação
Universidade do Minho, Braga - Guimarães, Portugal

Abstract: This paper presents a demonstration case on the successive application of a model-based technique to assist on the refinement of software logical architectures. The technique is essentially based on the transformation of use cases into object diagrams. The applicability of the technique is illustrated by presenting some results from a mobile application. For mobile software, the definition of the underlying service-oriented 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 successively applying the technique.

1. INTRODUCTION

A Model–Driven Development (MDD) approach is a software development technique that uses models during its execution. With MDD approaches, the development of software is made by successively transforming models into other models, until the final system is obtained.

This article presents the 4-Step Rule Set (4SRS) transformation technique that employ successive transformations of the software architecture, to satisfy the elicited user requirements. It is mainly based on the mapping of use cases into object diagrams. The technique's iterative nature and the use of graphical models ensure that architectures reflect user requirements [1, 2].

Since the 4SRS is an MDD method, its description should contain all elements that are usually present in any software method. It should describe

* Research funded by FCT and FEDER under project *STACOS* (POSI/CHS/48875/2002).

which intermediate and final artefacts should be produced, which notations should be used to create those artefacts and which tasks should be performed, and in which sequence, to create the required artefacts.

4SRS associates, to each object found in analysis, a given category: interface, data, control [3]. This categorization originates object models that, in their essence, are similar to the architectures imposed by the Model-View-Controller [4] or by the Entity-Boundary-Controller [5] patterns.

The 4SRS technique is organized as four steps: (1) object creation, (2) object elimination, (3) object packaging & aggregation, and (4) object association. Additionally, the 2nd step is further subdivided in 7 micro-steps. The application of the 4SRS to obtain the first logical architecture of the demonstration case is described in [6]. After executing all 4SRS steps, the logic architecture for the system that captures all its functional requirements and its non-functional intentionalities is obtained. An object model shows the distribution of significant properties of a system across its parts.

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 successively executing the 4SRS technique. The 1st execution, whose details are described in [6] supports the platform requirements analysis by generating one platform object diagram that corresponds to the logic architecture of the system. This paper explains, for the demonstration case considered here, the usage of the 4SRS to derive an object diagram that shows the services the system needs to accomplish its responsibilities.

The 2nd 4SRS execution, which this paper aims to explain, supports service requirements analysis by generating one service object diagram that corresponds to the logic architecture of the service to be specified.

The demonstration case is a platform for mobile applications, supporting usability, openness, interoperability and scalability. It deploys reusable service components to ease the development of context-aware applications that allow citizens to perform a set of government-related activities, and to access the most proper services at any time, anywhere.

2. MODEL-BASED TRANSFORMATIONS

The raw object diagram of the mobile application platform, shown in [6] and obtained after applying the 4SRS, is used in this paper as a starting point for discussing the technique. It identifies the system-level entities, their responsibilities and the relationships among them. Its purpose is to focus at an appropriate decomposition of the system without delving into details.

The components of that object diagram were obtained by reasoning about the characteristics of the service-oriented platform. Applications can be built

on top of this architecture by specifying the right composition of services, building a user interface, and orchestrating the data-flow among the various components. Configuring services and applications so they can be reliably reused and composed into larger applications is a major challenge [7].

The resulting raw object diagram (from the 1st execution) can be used in the subsequent phases to define well-delimited sub-projects, by using collapsing and filtering techniques. These techniques redefine the system boundaries, giving origin, for instance, to the database project, services formalization, or platform pattern analysis. Fig. 1 shows the collapsed object diagram that was obtained from the raw object diagram by hiding the packages details. Therefore, links appear at a higher level of abstraction and the resulting object diagram is easier to be read.

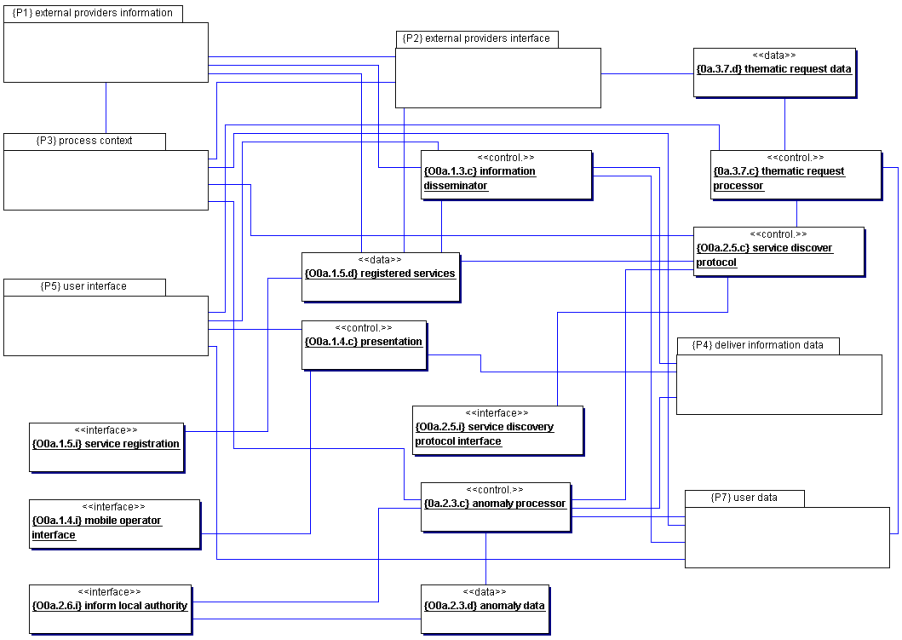

Figure 1. Collapsed object diagram.

Fig. 2 shows the filtered object diagram that was obtained by using collapsing and filtering techniques described in [1] by considering package {P5} as a subsystem 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 subsystem specification and partition of subprojects among various teams.

In this paper, we consider the refinement of package {P5} that has given origin to the AVAccess service. This service is a single point of contact with

the platform and should redirect the user to the appropriate service. In particular, when the user intends to report a complaint, he needs to access the AVAccess service and to select the report complaint functionality.

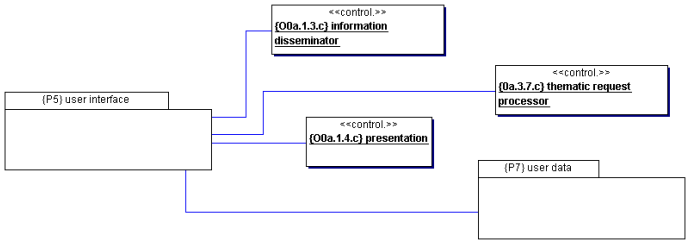

Figure 2. Filtered object diagram for package {P5} service derivation.

Criteria illegible for filtering depend on project management issues, functional implementation domains, etc. Fig. 3 depicts the filtering process executed over Fig. 1 to obtain a {P5}–centric filtered object diagram. During the filtering process, all entities not directly connected to {P5} must be removed from the resulting filtered object diagram.

3. ITERATIVE ARCHITECTURAL REFINEMENT

The development of mobile applications typically follows a service-oriented approach. A service is a software entity running on one or more machines and providing a particular type of function to unknown clients. These services must communicate with each other, to give rise to a service-oriented architecture. The communication can involve either simple data passing or two or more services coordinating some activity. Some means of connecting services to each other is needed, so workflow is a critical part to make services effective. When those services react to changes on user context, applications are said to be context–aware.

For mobile applications, the definition of the underlying service-oriented software architecture must consider the services themselves as user requirements, as well as the mobile operators' entry points and the final clients interfaces, and use them to characterize the platform.

{P5} can be considered as the system to be designed and apply, once more, the 4SRS technique to support its architectural refinement (in Fig. 2). The iterative application of the 4SRS technique suggests the construction of a new use case diagram (called service use case diagram) that captures the users requirements of the new subsystem to refine. From this use case diagram, the corresponding raw object diagram is derived (called service object diagram). This approach contrasts with the one that suggests the

application of design patterns [4, 8] to impose into the logical architecture a already proven reference architectural model. Our proposal does not reject this pattern-oriented view, only defers it into latter stages of development.

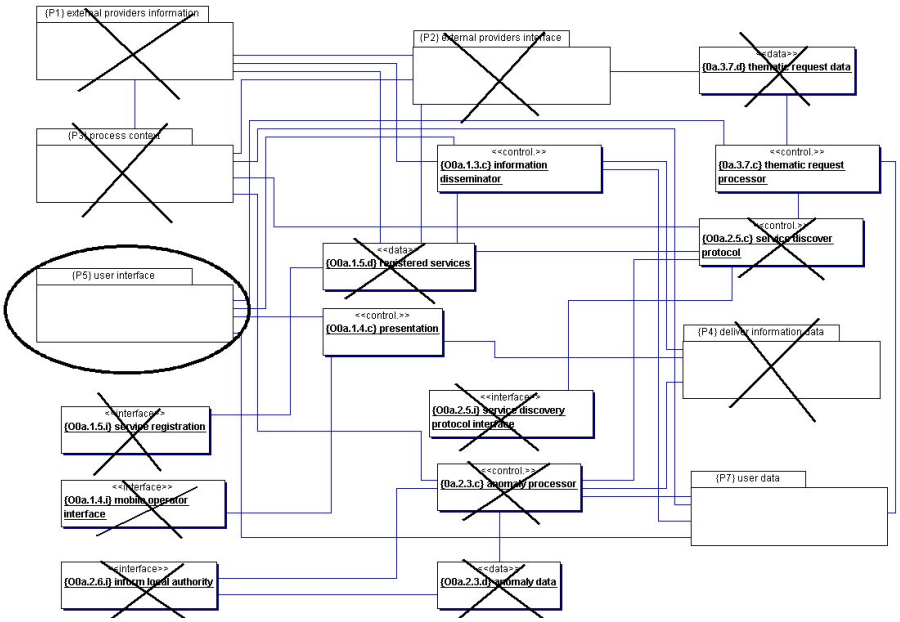

Figure 3. Filtered object diagram for package {P5} service derivation.

The use case diagram in Fig. 4 was created to support 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 iterative application of the 4SRS technique. All the external entities 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 rise to any actor in Fig. 4, 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. 4, since it was already connected to the use cases that gave origin to the objects inside package {P5}, during the development process described in [6]. Actors in Fig. 4 must be viewed as external components, from the point of view of the AVAccess service. To attain better actor semantics within the associations with the obtained use cases, actor {O0a.3.7.c} in Fig. 4 was specialized into two different actors: Application System Context Aggregation Service and Application System Service Repository.

The AVAccess service is the platform component where all user requests are redirected by default. Its service components or end services are architectural components developed and deployed by the local authorities and are the ultimate components to be accessed by the user. They appear as result of

other architectural decisions. The AVAccess service is a single point of contact and should redirect the user to the appropriate end service. Users usually start the interaction with the system by contacting this component.

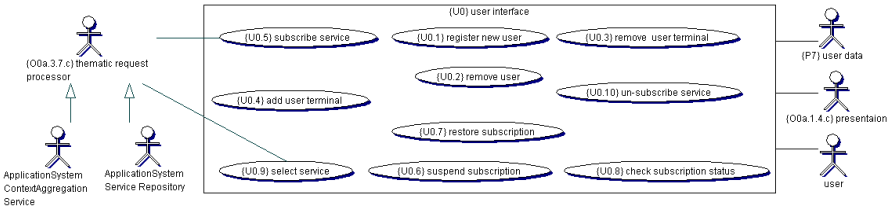

Figure 4. Use case diagram for AVAccess service.

One example of a description for the top-level use cases is next presented. Similar descriptions were created for the other top-level use cases.

{U0.1} register new user: the user provides user personal information to the AVAccess system. Its personal information consists of userName, password, and, optionally, user profile information. The AVAccess service parses user personal information and sends it to subsystem User. The AVAccess sys-tem sends back the information on success/no success of this operation. The information sent to the user is format-ted by the subsystem Presentation. The system must know terminal model information.

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 model transformation steps.

The table for supporting the transformation steps uses one row for each object and one column for each step. 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 that use case. For the demonstration case, there is no use case refinement, so step 1 is applicable to all (10) use cases in Fig. 4, which gave origin to 30 objects. Fig. 5 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, each use case is classified as one of the 8 different combinations or patterns (Ø, i, c, d, ic, di, cd, icd). This classification helps on the transformation of each use case into objects, and provides hints on which objects to use and how to connect them. For the demonstration case, {U0.1}

was classified as “i”, meaning that only the interface object is kept (the control and data objects will be eliminated in micro–step 2ii). {U0.5} was classified as “icd”, which means that all objects are kept.

Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	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}		×								
{O0.1.d}		×								
{O0.1.i}		–	register user interface	allows the parse of the user personal...	itself	{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...	itself		–			{O0.5.d} {O0.5.i}
{O0.5.d}		–	defined activities	interface with the data of the...	itself	{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}		×			{O0.5.c} {O0.5.d}

Figure 5. Table for supporting the 4SRS technique.

The 3rd column supports the execution of micro–step 2ii. In this micro-step one decides if each object created in step 1 makes sense in the problem domain, since the creation of objects in step 1 was blindly executed. Objects 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 subsystems and vice versa, which means that data and control dimensions are not within the scope of this use case.

The 4th column is dedicated 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, considering its main component. {O0.1.i}, for instance, was named register user interface.

The 5th column is related 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. For the demonstration case, the following descriptions were obtained:

{O0.1.i} register user interface: allows the parse of the user personal information and sends it to the destination subsystem, and sends back the information on success/no success of the request.
{O0.5.c} subscribe service: will process the request Subscribe service. Will request to the user all the additional information needed to perform the request of the user.
{O0.5.d} defined activities: interface with the data of the available activities in the system (could be a XML file).

{O0.5.i} subscribe service interface: sends the subscribe service information to the destination subsystem, and sends back the information on success/no success of the request.

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, and the discovering of missing requirements. The "is represented by" column stores the reference of the object that represents the object being analyzed. If the analyzed object is 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 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 is related to micro–step 2vi. This is a fully "automatic" micro–step, since it is based on the results of the previous one. The objects that are represented by other ones must be eliminated, since its system requirements no longer belong to them.

The 9th column is used for micro–step 2vii. Its purpose is to rename the objects not eliminated in the previous micro–step and that represent additional objects. For the demonstration case, object {O0.1.i} was renamed "users management interface" to reflect the list of objects it represents.

The 10th column supports the execution of step 3. Since aggregations and packages were not used in the demonstration case, column 10 is not filled.

The 11th column supports step 4. The associations in the demonstration case were solely derived from the use case classification in step 1. The classification of {U0.5} as type "icd" suggests the existence of three internal links relative to the objects generated from the same use case. However, "id" link (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. 6 depicts the raw object diagram for the AVAccess service, obtained after a new application of the 4SRS technique over the global logical architecture of the application represented in Fig. 2. Object {O0a.4.1.i} in

Fig. 2 is mapped into object {O0.1.i} in Fig. 6. This object receives user requests for user management and service subscription. In the case of use case {U0.5}, {O0.1.i} uses the functionalities of {O0.5.d} and {O0.5.c}.

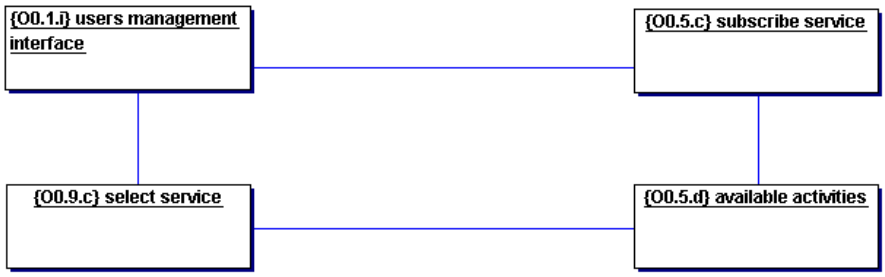

Figure 6. Raw object diagram of the AVAccess service.

Object {O0a.2.2.i} in Fig. 2 maps into object {O0.1.i} in Fig. 6. This object receives user requests in case of execution of use case {U0.9} and object {O0.1.i} uses the functionalities associated with objects {O0.9.c} and {O0.5.d}. The obtained raw object model (Fig. 6) constitutes the semantic reference for the service to be designed, since it has emerged from the software logical architecture (Fig. 2) of the platform by adopting a complementary functional refinement at architectural level.

After obtaining this new architectural refined raw object model, the underlying service can be described by a set of diagrams to specify the corresponding architectural component, namely, 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 views of the same service are not generated from the application of 4SRS technique, even though they are easier constructed after obtaining the raw object diagram of the service (Fig. 6).

6. CONCLUSIONS AND FUTURE WORK

A software infrastructure for running mobile applications must find, adapt, and deliver the right services to the user computing environment based on his context. The current trend in software industry is for service providers to supply reusable functions via components called services. Building applications involves specifying the right composition of services, building a user interface, and defining the data flow among the components.

For mobile applications, the definition of the underlying service–oriented architecture must consider the services themselves as user requirements, as

well as 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 a mobile application was obtained by recursively applying the 4SRS technique. The technique has shown its usefulness by assuring the generation of a seamless specification of the service–oriented architecture requirements.

The proposed iterative usage of the 4SRS technique allows designers to build a new use case diagram that captures the users requirements of the new system to refine a service. From this use case diagram, a raw object diagram can be derived. This approach is complementary to the use of design patterns by allowing a functional refinement of requirements at architectural level, considering the specific aspects of the system under design. This transformational approach shows that model continuity is an important topic and highlights the importance of defining a well-defined process to relate, map and transform requirement models [9]. In the presented case, the 4SRS has allowed the specification of one particular service, considering all the architectural decisions previously taken to specify the platform where the service is supposed to run, by assuring a continuous mapping between the platform and the service models.

As future work, the 4SRS technique will be extended to consider the transformation of objects diagrams into class diagrams, which seem a crucial step for software-intensive systems.

REFERENCES

1. J.M. Fernandes, R.J. Machado. From Use Cases to Objects: An Industrial Information Systems. OOIS 2001, Calgary, Canada, pp. 319-328, Springer, August, 2001.
2. J.M. Fernandes, R.J. Machado, H.D. Santos. Modeling Industrial Embedded Systems with UML. CODES 2000, San Diego, California, U.S.A., pp. 18-22, 2000.
3. I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.
4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, 1996.
5. I. Jacobson, G. Booch, J. Rumbaugh. The Unified Software Development Process. Object Technology. Addison-Wesley, 1999.
6. R.J. Machado, J.M. Fernandes, P. Monteiro, H. Rodrigues. Transformation of UML Models for Service–Oriented Software Architectures. ECBS 2005, Greenbelt, Maryland, U.S.A., pp. 173 182, 2005.
7. G. Banavar, A. Bernstein. Software Infrastructure and Design Challenges for Ubiquitous Computing Applications. Communications of the ACM, vol. 45, no. 12, pp. 92-96, 2002.
8. R. Ahlgren, J. Markkula. Design Patterns and Organisational Memory in Mobile Application Development. PROFES 2005, Oulu, Finland, pp. 143-156, 2005.
9. R.J. Machado, I. Ramos, J.M. Fernandes. Specification of Requirements Models. In A. Aurum and C. Wohlim (Eds.), Engineering and Managing Software Requirements, pp. 47-68, 2005.