Model-based Transformations for Software Architectures: a pervasive application case study

Paula Alexandra Fernandes Monteiro

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Informática elaborada sob a orientação do Doutor João Miguel Lobo Fernandes e do Doutor Ricardo Jorge Silvério de Magalhães Machado

Departamento de Informática
Escola de Engenharia
Universidade do Minho
Braga, Dezembro de 2005

Acknowledgements

I would like to express my gratitude to all people that directly or indirectly contribute to the execution of this work.

Special thanks to my supervisors Doutor João Miguel Fernandes and Doutor Ricardo Machado for creating the environment for doing this research and for the review of this document.

I would like to thank Doutora Helena Rodrigues, of the Information Systems Department for creating the environment needed to this research.

Model-based Transformations for Software Architectures: a pervasive application case study

Abstract

The software architecture is a representation of the software system structure that should capture the system requirements and that comprises the software components and the relations between them. It is of obvious importance that the system requirements can be derived from the user requirements so that software engineers can more easily relate and trace them. If both types of requirements are models this transformation is a model-based technique. User requirements are typically described with informal models and focused to the problem domain. System requirements are described with abstract models of the system with a high-level of detail, and correspond to the first representation of the system's structure or architecture.

To identify the system components the software engineer should start the development by defining the functional model that reflects the system functionalities offered to the users represented through a use case diagram. Object diagrams can be used to show the objects that the system is composed of. This transformation can be made with the usage of Four-Step Rule Set (4SRS) technique [11, 13] which is essentially based on the mapping of use case diagrams into object diagrams.

The main aim of this thesis is to present a proposal on how to transform in a systematic and sustained way user requirements models into software architectures by proposing some improvements on the original version of the 4SRS technique. The application of this technique is illustrated within two different usages on a pervasive case study. In the second usage we also present how to refine the software logical architecture by recursively applying recursively the 4SRS.

Transformações Baseadas em Modelos para Arquitecturas de Software: caso de estudo de uma aplicação intrusiva

Resumo

Uma arquitectura de software é uma representação da estrutura do sistema de software que captura os requisitos do sistema e que compreende os componentes de software e as relações entre eles. É importante que os requisitos do sistema possam ser derivados dos requisitos do utilizador de modo a que os engenheiros de software os possam relacionar e seguir. Se ambos os tipos de requisitos forem modelos esta transformação diz-se baseada em modelos. Os requisitos do utilizador são descritos tipicamente com modelos informais e com foco no domínio do problema. Os requisitos do sistema são descritos com modelos abstractos do sistema com um elevado grau de detalhe, constituindo a primeira representação da estrutura ou arquitectura do sistema.

Para identificar os componentes do sistema o engenheiro de software deve iniciar o desenvolvimento definindo o modelo funcional, que reflecte as funcionalidades do sistema disponibilizadas aos utilizadores, representando-o através de um diagrama de *use cases*. Os diagramas de objectos podem ser usados para mostrar os objectos que compõem o sistema. Esta transformação pode ser feita através da utilização da técnica Four-Step Rule Set (4SRS) [11, 13] que se baseia essencialmente no mapeamento de diagramas de *use cases* em diagramas de objectos.

O principal objectivo desta tese é apresentar uma proposta de como transformar de um modo sistemático e sustentado os modelos de requisitos do utilizador em arquitecturas de software propondo melhoramentos à versão original da técnica 4SRS. A aplicação desta técnica é ilustrada com duas diferentes utilizações de uma aplicação intrusiva. Na segunda utilização é também apresentado como refinar arquitecturas lógicas de software utilizando recursivamente o 4SRS.

Contents

List of Figures

Chapter

1

Introduction

Summary

This chapter presents the motivation of this thesis.
The main motivation is the extension of the Four-Step Rule Set (4SRS), which is a technique to transform user requirements models into software architectures, and in particular for pervasive applications.

1.1. Motivation

Although several definitions were already proposed [27, 33] a pervasive system can be defined in a simple way as a system that lives around us providing services everywhere. In this context a mobile application, i.e. an application that can be on a mobile phone and that enables the access to the system functionalities, is a pervasive system. It is an application that can follow a service-oriented approach.

In the last years, the widespread availability of mobile communications makes the information available at any time and everywhere. In a mobile information environment, the users can interact with a mobile information space with context-aware services. A context-aware service is a service that gives information that is useful to the user in a given context [4, 28].

Mobile applications must be aware of the physical environment that is surrounding the user in a certain moment. It must change automatically its behaviour in order to satisfy the new situation of the user. So a software infrastructure for mobile applications must be capable to dynamically find and deliver the services that best fit the user needs based on his context [7].

The software architecture is the high-level structure of a software system that includes the software components and the relations between those components. It is a representation of the software system structure and it should capture the system requirements.

The identification of the system components of an architecture requires the definition of the functional model that captures the functionalities available to the user [11, 13]. Those functionalities available to the users should represent the user requirements. The system requirements are supposed to be derived from the user requirements. With this derivation it is assured that the design phase is really (or at least strongly) based on the users needs. One of the tools that is typically used in software engineering to capture the system functionalities are use case diagrams, which are simple and easy to read by non-technical stakeholders. This fact implies that they are suitable for discussion between analysts and users, but also between analysts and designers.

Developing mobile applications may include the use of a service-oriented approach. A service is a software entity that runs in one or more machines and provides a specific type of function to clients. Those services have to communicate among them and their combination constitutes a service-oriented architecture. So, in a mobile application, the definition of the service-oriented software architecture must take into account the services as user requirements, as well as the mobile operators and the final clients interfaces, that will be used to characterize the platform; i.e., in use case diagrams, the services must be represented as use cases (functionalities) and the mobile operators and the final clients interfaces should be represented as actors.

An object model can be a design level notation. In this case, it represents a technology-independent architecture, in which the system requirements are considered.

The object model is a design level model, is the first architecture that captures the system requirements. This model shows the objects that the software system is composed of. Objects can be obtained from the use cases that represent the system functionalities. It implies a transformation technique to transform the use case diagram into an object diagram, i.e., to transform the system requirements into the first architecture of the system.

The transformation of a use case diagram into an object diagram is a complex and critical task, since there is not an evident mapping from use cases to objects. To perform this transformation the behaviour specified by a use case must be distributed into several objects.

One technique to guide this transformation is the Four-Step-Rule Set (4SRS) [11, 13]. This technique is divided, as the name indicates, in four steps that guide the transformation of the use cases into objects and its main idea is to ensure a strong continuity of the models.

However, this technique is only a partial solution to the transformation of the use case model into an object model. The interpretation of the steps that compose this technique can be subjective since they are presented in a textual format and the description of each step is made in a somehow abstract way.

The main motivation of this thesis is to contribute to extend and improve the 4SRS technique that helps on transforming use case diagrams into object diagrams. The contribution of this thesis is to present a more detailed and easy-to-apply version of the 4SRS technique. To validate the proposal made here, the "new" 4SRS is used to transform user requirements models into the first logical architecture of the system for a pervasive application case study. However, we believe that this technique is sufficiently generic that it can be used in other application domains. To support the execution of the 4SRS transformation a tabular representation was defined to make the usage of the technique easier for software engineers. This tabular representation was created with the purpose of defining in the future the mechanism to automate at least partially the 4SRS technique.

This work has been developed and applied in the European Sixth Framework project USE-ME.GOV (USability-drivEn open platform for Mobile GOVernment - IST-2002-002294). The aim of USE-ME.GOV project is to develop a new open platform for mobile e-government services. The platform must support interoperability, openness and scalability. It must also facilitate the service deployment and access. In other words, the main goal is to provide an open platform for mobile services provided by the public authorities. USE-ME.GOV offers new types of services for a specific user group. The services must be configurable and personalised [9].

1.2. Research Goals

The main goals of this work are the following:

1. To improve the original version of the 4SRS technique to allow its application in the architectural refinement of software systems, after the first architectural specification;
2. To apply and to validate the new version of the 4SRS technique with the pervasive application taken from the case study of the USE-ME.GOV project.

1.3. Dissertation Roadmap

In chapter 2, a generic description of the model-driven approach and the existent model-based transformation techniques are presented. Chapter 3 presents the original 4SRS technique and discusses how this technique was extended and improved to better support the transformation of use case diagram into object diagrams. This constitutes the main contribution of this thesis. In chapter 4, the 4SRS is applied in a case study for a pervasive application, with the aim of exemplifying its usage and utility. The thesis ends in chapter 5 with conclusions and some pointers for future work.

Chapter

2

Model-Based Transformations

Summary

A model-driven approach is a software development process that uses models during its execution. With model-driven approaches, the development of a software system is made by successively transforming models into other models, until the final system is obtained.

One of the open issues related to model-driven approaches is on the transformation of a model into another one. To tackle this problem, several methodologies were proposed.

In this chapter the model-driven approach as a paradigm to develop software systems and some model-driven methodologies are introduced.

2.1. Introduction

One important factor that must be considered during the development of a system is modelling. The Unified Modeling Language (UML) is a language proposed by the Object Management Group (OMG) that can be used to express the models created during the system development.

A software system development approach could use models during the process and in this case the approach is called *model-driven*.

A problem to the model-based approaches for software development is the transformation from one model to the other. The use of a technique to guide in this transformation is very important in the model-driven approaches, since the transformation techniques should describe which artefacts must be produced, which notation must be used to create the artefacts and which tasks must be executed and in what sequence, to guarantee that the artefacts are created with the correct mapping from one model to another and the correct purpose of this transformation.

The OMG definition of model transformations says that "Model transformation is the process of converting one model to another model of the same system" [23]. In model transformations the (sub-)system defined by the input model must be the same (sub-)system that is defined by the output model that is generated by the application of the transformation rules defined in the selected methodology [18].

In this chapter the model-driven development is introduced as well as some model-based transformation techniques used in model-driven approaches.

2.2. Model-driven Software Development

A system development approach is model-driven if it provides means for use models to guide the course of understanding, design, construction, deployment, operation maintenance and modification [23]. The main idea of a model-driven approach is the usage of well-defined models, which are built following the rules specified by the modelling language, and that represents abstract specifications of the system

The Unified Modeling Language (UML) is a standard language to specify the structure and the behaviour of a system. It is a graphical language for visualizing, specifying, constructing, and documenting the artefacts of distributed object systems [1], which can be used to design the models in a model-driven approach.

Despite actual system development approaches that use code in the software development, the approaches that use models as the main artefact in the software development are significantly increasing. These approaches can be named as model-driven approach [3].

One of these approaches is presented in [15]. In this approach the functional requirements of the system are captured by use cases (the set of all use cases is a use case model). The use case model describes the complete functionality of the system. However, use cases do not only specify the system requirements but also drive the development process. With this development process and based on the use case model, a set of design and implementation models that accomplish the functional requirements are created.

Another approach that can be claimed as model-driven is presented in [16]. In this approach, the system development is made with several kinds of models. The functional requirements are captured by the requirements model, which is composed by a use case model and with the interface descriptions. After the approval of the requirement model, the analysis model is developed, by specifying all the logical objects of the system, the relation between them and how they are grouped. Using the requirements and the analysis model the design model is developed and based on this the implementation model is created. The implementation model aims to implement the system and consists mainly in the source code written by the programmers. Another model can be created, the test model with which the system is verified.

A recent model driven approach, promoted by the OMG, is the Model Driven Architecture (MDA) [23]. MDA is a framework that uses models in the software development process and transforms input models into output models sharing the idea of separate the specification of the functionality of a system from the specification of its implementation on any specific platform. This separation is made by dividing the MDA in two model families: Platform Independent Model (PIM) and Platform Specific Model (PSM). The software is developed by transforming the PIM into a PSM (that is closer to code than the PIM) using a series of transformations called model transformations [18, 19, 25].

The Model-Driven Development (MDD) of software applications tackles the problem of developing a software system by promoting the usage of models as the main artefact to be developed. Like MDA, the MDD is based on the idea of PIMs that can be transformed into PSMs, using several technologies and languages. A MDD methodology has to define which models are to be used by the developers and which modelling tasks have to be performed during a software project.

In MDA, the first model to be defined is typically a PIM which is the input model to the model transformations. It is a model independent from any implementation technology and with a high level of abstraction and it can provide the specification of the system structure as well as the specification of the system functionality (the implementation details are not presented).

The PIM is created in the analysis phase of the software development usually described in a platform independent modelling language, like UML. A PIM can be transformed into one or more PSMs. A PSM can be called output model (it is the output model of the model transformations) that is created to specify the implementation of our system in a specific implementation technology. The output model can provide different levels of details depending on its purpose: if the output model of the model transformation provides all the information required to specify the system implementation this model is a PSM; otherwise, the output model is a PIM that will be refined in order to get a PSM. The output model must describe the same system as the input model. So the model transformation must preserve the meaning of the models that are being transformed.

Another proposed definition of model-driven [25] extends the OMG definition. In this case a system development approach is model-driven if the development is accomplished with the usage of conceptual models in the distinct levels of abstraction; the difference between a PIM and a PSM is clear; models are very important not only in the initial phases of the development process but also in maintenance and reuse; and models provide a precise structure for transformation and refinement.

The Model Driven Engineering (MDE) deals with the problem of a system development promoting the usage of models as a principal artefact to construct [5]. MDE is a wide term comparing with MDA. It includes all models and modelling tasks needed to perform a software project from the beginning to the end. In a MDE development project, the transformations from a source model to a target model are necessary, like in the MDA projects, but a transformation from a PIM to another PIM could be necessary. The importance of this transformation from a PIM to a PIM is, for example, the possibility of transforming a model that represents the user requirements into another model describing the system architecture that supports the user requirements.

The description of a MDE method has to define the artefacts that have to be produced, the language used to create them and the tasks that must be executed (to better the process model) to guarantee the creation of the required artefacts. An MDE method has two main differences comparing with a traditional development method: all the artefacts created are represented using a well-defined modelling language and as a consequence, tools can be created. These tools analyse and transform all the artefacts of the project [5].

The 4SRS technique [11-13], which is the basis of this thesis, starts the development with the definition of the functional model. The functional model is defined with UML use case diagrams. This diagram reflects the system functionalities that are available to the users. After the definition of the use case diagrams, the system objects will be obtained based on the use cases and their respective textual descriptions.

2.3. Related Work

The transformation from one model to another is one of the main ideas of the model-driven approaches. There are several approaches that guide the developer in the model transformation during the software development process.

In the last years UML has become a standard language for modelling software requirements and design and by this the majority of the model transformation approaches are used to transform UML models into other UML models [6, 21]. However, there are also approaches to transform UML models into other type of models like Colored Petri Nets [10, 14], and even approaches to transform UML models into code (like Jamba which is an object-oriented framework that provides a set of classes to represent UML models and a mechanism to generate code [2]).

In the group of the available techniques to transform UML models into other UML models, there are several works that propose a solution for the transformation of the UML use case models into objects or class models [6, 8, 16, 21] by identifying objects or classes through the description of the use cases [11, 13, 17] or through the use case goals [21, 26]. Some of those transformation methodologies identify the objects or classes based on the descriptions of use cases [17, 21], but exists also transformations methods that identify the classes from the use case goals [8, 26].

In the methodology defined in [8] it is needed to find the external actors and objects that interact with the system. The system functionality, like in the 4SRS, is described using the UML notation, in particular, the use case diagrams. However, the next step is different from the 4SRS. The resulting model is mapped in a graph description. The partition of the graph according to different criteria gives origin to the identification of different objects.

The work presented in [21] takes into account the use case goals to identify the classes. This approach transforms the source model into the target model, using the use case goal identification instead of the use cases and their descriptions as in 4SRS. The classes are identified based on the use case goals instead of the information kept in the use case description. For each use case, the respective goal must be found. After finding the goals, it must be identified which entity uses each goal. Each entity will be specified as classes from a class diagram.

The approach presented in [26] uses goals to elicit functional and non-functional requirements and specify the relations between them.

The last two approaches presented [21, 26] give support for the transformation of an analysis model into an architectural design model. But these approaches do not have a framework defined that helps to capture of the semantic of the use cases. This can be solved with the use of a scenario based requirements engineering technique [20, 31].

The current methods used to transform the use cases in objects or classes can be divided in two groups. The first group includes the methods based on the analysis of the textual requirements; the second group contains the tools which automate the transformation from the use cases to the objects/classes models. The 4SRS is a technique of the first group.

Another approach to model transformation uses activity diagrams [17]. The use case model obtained in the analysis phase is complemented with an activity diagram, in other words, the use case description is converted into the activity diagram. The ambiguities and the inconsistencies can be removed during this conversion of the textual description into the activity diagram. Next the objects and the classes are identified.

Using the same idea of complementing the use case model with other artefacts, another work is proposed [6]. In this work the use case diagrams are complemented with a formal document, which is called contracts. It is described in a language with a precise semantic and the main idea of the use of this artefact is to describe the behaviour of the actors. Contracts are used as a step between the use case diagrams and the class diagrams.

Like the last two approaches, the 4SRS can complement the use case diagrams with other software artefacts, like sequence, activity and state diagrams.

A proposed model based approach combines the data-flow diagrams (DFD) with the UML object-oriented diagrams [22, 32]. The main idea of this combination is to help the development of complex embedded systems. This approach is based on the 4SRS, the requirements are captured and a use case model is built. The initial object diagram is transformed from the use case diagrams. Finally, after the development of the object diagram, a DFD is constructed following a set of defined steps.

The UML model-to-model transformations can also be made with the usage of XMI (XML Metadata Interchange), which is an XML-based standard for interchange of UML models [24]. The model is converted into XMI and the transformation from one model to another is made with the usage of existing XML tools like XSLT, which transforms the converted XMI model into another XMI model. This type of transformations is not simple because it is required some experience and effort to define the XSLT transformations.

As said before there are some approaches that transform UML diagrams into other type of diagrams, for instance into CPNs [10]. The idea of this approach is to generate a prototype of the user interface and a formal specification of the system. Such as the other approaches discussed before, this starts with the creation of the use case diagram of the system. Beside the creation of the use case model, for each use case it will be created a sequence diagram describing its scenario. The next step is the transformation of the use case and sequence diagrams into CPNs. If there are different CPNs that correspond to the same use case are joined in a unique CPN. After this step the CPNs will be linked into a global one, capturing the system behaviour.

There is another proposal to transform UML models into Colored Petri Net models [14] in which the primary artefacts considered are the statechart and the collaboration diagrams. The statecharts are converted in colored Petri nets and the collaboration diagrams are used to connect the models, in order to get a single Petri net. This approach is used to modelling behaviour. The transformations can be done manually or using a tool. To help the manual transformation an algorithm was devised to support the transformations between scenarios and behavioural models. It guides the transformation between UML collaboration diagrams and state transition diagrams [29].

There are also model transformation approaches, which propose visual languages to make these transformations. One of these works is the Visual Model Transformation (VMT) approach [30]. If offers a transformation language and a tool that allows the UML model transformations. The rules used in this language, like others works, uses the Object Constraint Language (OCL) in the model transformation.

2.4. Conclusions

In this chapter it was presented the concept of “model-based transformation”. To make this brief description, it was generically explained some of the model-driven approaches that are used in software system development.

A brief description of the model-driven techniques that could be used to make the model transformations was also presented.

There are several approaches that use models in the system development process, so they are model-driven. A model-driven approach has to use a model transformation technique to guide the transformation of the models during all the development process.

There are model transformations approaches used to transforms models into models, but there are also approaches to transform models into code (not presented in this thesis). Some of the model-to-model transformation approaches make this transformation by converting the graphical model into text (XML/MOF/XMI) and transform it into another model also represented in XMI. These model–to–model transformations approaches can transform UML models into UML models (for instance, transform a use case model into a class model), but can also transform UML models into other kind of models (like Petri Nets).

Chapter

3

Transforming Use Cases into Objects

Summary

This chapter presents the original 4SRS technique that guides the designers on the seamless transformation of use cases into objects. The extensions and improvements of that technique, proposed in this thesis, are also described. In particular the steps and micro-steps that constitute this technique are described and applied to an example to illustrate the rationale of the method.

In order to give automatic support to the 4SRS transformation steps, a tabular representation is also presented in this chapter.

3.1. Introduction

Transforming an analysis model into an architectural model is not simple and presents a considerable number of difficulties. This transformation has many decisions that cannot be executed by a method or a tool since there is a discontinuity between functional and structural models. The transformation of the use cases into objects is a crucial task in any model-driven development project, because there is not a direct mapping between them.

The object diagram can have different intentions. It can be used to enrich the user requirements, to complement use cases describing requirements. But it can also be used to provide an ideal architecture that captures the system requirements. In the technique defined in this chapter it will be used the last purpose because the object diagrams resulting from the application of the 4SRS are used to present the components of the system under development.

In order to identify the system components it is recommended to start the development process by capturing the functionalities of the system that are offered to the users, in such way that the components can take into consideration the user requirements. The technique used to capture the system functionalities are use case diagrams.

Use case diagrams are readable and simple to understand because it has only three concepts: use cases, actors and relations. This simplicity allows the participation of clients and users in the requirements capture. Despite being used in object-oriented projects, use cases do not hold any intrinsic object-oriented property. Although using use cases for capturing the user requirements is a suitable technique we are faced with the problem of transforming use cases into objects.

In this chapter it will be presented a technique called 4SRS that helps the transformation task from use cases into objects. This technique solves the problem of define the system objects based on the use cases and their respective textual descriptions.

3.2. Four Step Rule Set original version

The 4SRS is a model-driven technique that offers some guidelines and techniques to transform use cases into objects, which consists basically in distributing the behaviour specified by the use cases to a set of objects. The original version of the 4SRS was introduced in [13] and [11].

By applying this technique, the designers are supposed to obtain a holistic set of objects, so that the inter-relations amongst the objectified use cases can be successfully simplified in order to obtain a reduced number of relevant system-level objects.

This original version of the technique is divided in the following 4 steps:

1. Step 1 - object creation: Transform each use case into three objects (one interface, one data, and one control). Each object receives the reference of its respective use case appended with the suffix (i, d, c) that indicates the object's category;
2. Step 2 - object elimination: Based on the textual description for each use case, it must be decided which of the three objects must be maintained to fully represent, in computational terms, the use case, taking into account the whole system and not considering each use case *per si*, as happens in a non-holistic approach;
3. Step 3 - object packaging & aggregation: The survival objects (those that were maintained after the execution of step 2), for which there is an advantage in being treated in an unified way, should give origin to aggregations or packages of semantically consistent objects;
4. Step 4 - object association: The obtained aggregates must be linked to specify the associations among the existing objects.

This technique can be considered a partial solution to the problem of transforming use cases into objects, because the execution of the second step has a subjective interpretation since the object elimination is accomplished based only in the textual description of the use cases which are written in a natural language and thus subject to ambiguities. This problem was tackled and solved in the improved version of the 4SRS, by dividing this step in several micro-steps, which is the main aim of this work.

One weakness of the original version is the huge number of objects that can appear at the time when the objects are created in the first step, a problem that will not be solved in this work. All the objects for each use case are created even if some of them will be eliminated in a later step.

The technique is risk-driven. When the use case model has refinements it can reduce risk. Refining a use case model increases the information since it introduces more detail and gives origin to a decrease of the risk because there is less probabilities of having a lack of information.

3.3. Four Step Rule Set improved version

The 4SRS technique presented in this thesis is based on the general technique presented before. It is divided in some steps and micro-steps. The main steps are:

- Step 1: object creation
- Step 2: object elimination (this step is divided in some micro-steps):
  - Micro-step 2i: use case classification
  - Micro-step 2ii: local elimination
  - Micro-step 2iii: object naming
  - Micro-step 2iv: object description
  - Micro-step 2v: object representation
  - Micro-step 2vi: global elimination
  - Micro-step 2vii: object renaming
- Step 3: object packaging & aggregation
- Step 4: object link

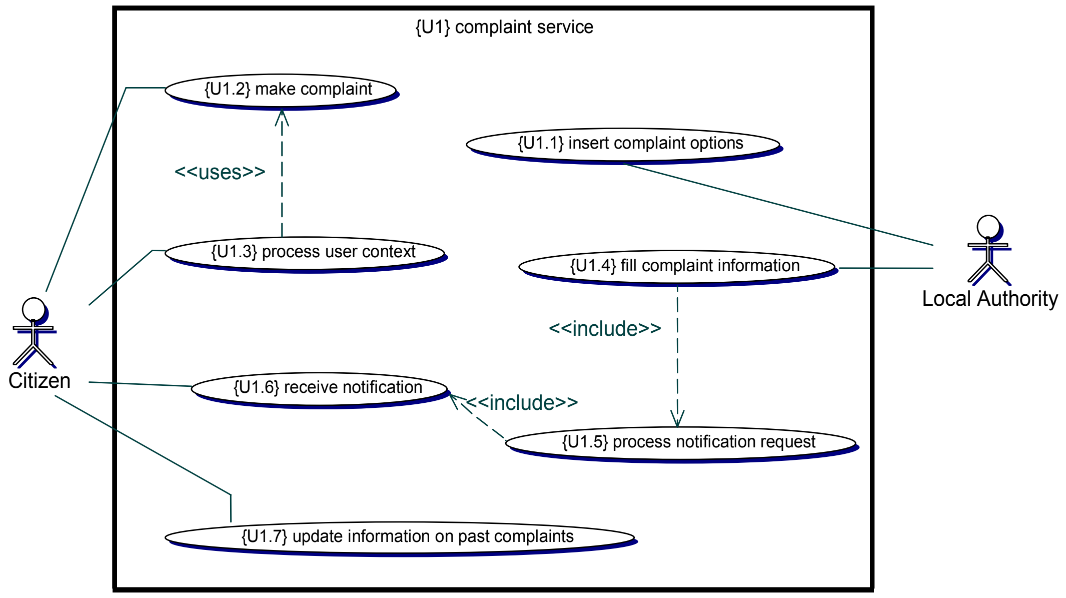

{U1} complaint service
{U1.2} make complaint
{U1.1} insert complaint options
<<uses>>
{U1.3} process user context
{U1.4} fill complaint information
Local Authority
<<include>>
Citizen
{U1.6} receive notification
<<include>>
{U1.5} process notification request
{U1.7} update information on past complaints

Figure 1. Top-level use case diagram

To illustrate the transformation technique, is next presented the step-by-step application of the 4SRS to a case study. The case study used in this section is the development of a service offered by a new open platform for mobile government applications. This platform (that is presented in the next section) is responsible for the deployment of a set of reusable service components, which allows citizens to access government activities at any time and anywhere. One of those activities (services) allows the citizens to report a complaint to the local authority.

The capture of the complaint service requirements allows the identification of all the use cases that compose the service. Figure 1 shows the use case diagram of the service. Each use case has one textual description. As an example, it is presented the textual description of use case {U1.2} and use case {U1.5}

> ***{1.2} make complaint****: the action of the user to issue the functionality of make complaint.*
>
> ***{1.5} process notification request****: This service verifies if users have requested to be notified of a complaint status change. The complaint status is updated via {1.4} fill complaint information. The user information was recorded in the service via {1.2.5} send user information. The user is informed via {1.6} receive notification.*

The use case {U1.2} was refined (figure 2), and next we present examples of the textual descriptions of the sub use cases.

> ***{1.2.2} select complaint option****: The user selects a complaint option. The list of complaints options is received by the user through the use case {1.2.1} get list of complaint options. The complaint option is recorded in the system.*
>
> ***{1.2.5} send user information****: The user sends information on terminal ID (this information is added by the platform to user request); user name (this information is added by the platform, if the user is registered, to user request); user context information (this information is added by the platform to user request). The user context is processed via {1.3} Process user context. The user information is recorded in the service.*

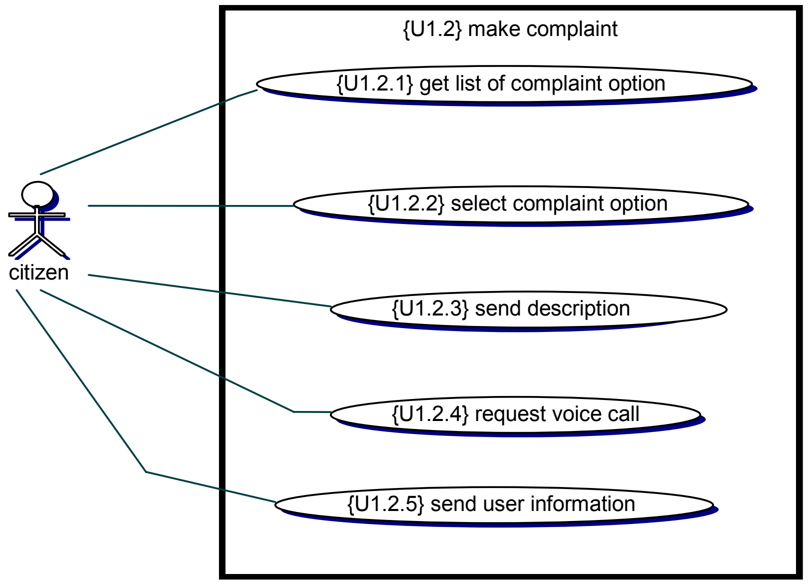

{U1.2} make complaint
{U1.2.1} get list of complaint option
{U1.2.2} select complaint option
citizen
{U1.2.3} send description
{U1.2.4} request voice call
{U1.2.5} send user information

Figure 2. Refinement of use case {U1.2} ***make complaint***

3.3.1. Step 1: object creation

In this step, each use case must be transformed into three objects. Each created object is classified with one of the following categories: ***interface***, ***data*** and ***control***. All objects receive a reference that starts with an "O" followed by the respective use case reference and appended with the suffix ***i***, ***d*** or ***c*****,** according to the type of the object category. In this step there is no need to take or validate any decision.

This creation step is only applied to the use cases that were not refined and to the sub use cases of refined ones. In order words the leafs of the use case hierarchy must be used, instead of the most hierarchical ones.

In the next steps there are only objects as design entities, but the use cases are still used to introduce the requirements in the object model.

Applying this step to the case study and more particularly to the use case **{U1.5}** three objects were created: **{O1.5.i}**, **{O1.5.d}**, **{O1.5.c}**.

3.3.2. Step 2: object elimination

This step is in our opinion the most important one of the 4SRS. The importance of this step to the success of the technique results from the fact that the definitive system-level entities are decided during its execution. We decided decompose the second step in seven micro-steps, due to its complexity.

The aim of this object elimination step is to decide which of the objects created during the first step must be kept in the model. The objects that are not eliminated must fully represent the use case. This decision must take into account the whole system and not each use case alone, and based on the textual descriptions of each use case. Additionally this step allows also the elimination of the redundancy in the user requirements and the discovering of missing requirements.

Micro-step 2i: use case classification

The first micro-step of the second step classifies each use case. This classification is used to help on the transformation of each use case into objects. It gives some hints on the object categories to use and how to connect those objects.

To understand this classification it is important to know how many combinations of objects exist for a use case. Since the 4SRS creates as a maximum three objects (one interface “i”, one control “c” and one data “d” objects) there are 8 different combinations or patterns, Ø, i, c, d, ic, di, cd, icd (ignoring the links between the objects).

As an example, the use case {U1.2.1} was classified as “di”, which means that only the interface and the data object are to be kept and the control object is to be eliminated in micro-step 2ii.

Micro-step 2ii: local elimination

In this micro-step, for each object created in step 1, the developer decides if it makes sense in the problem domain. The idea to check if the objects makes sense comes from the fact that the creation of the objects in the step 1 is executed without take into consideration the system context.

In the case study, object {O1.5.i} (the interface component of the use case {U1.5} *process notification request*) does not make sense in the problem domain. This decision can be justified by analysing its textual description. The use case {U1.5} is responsible to check in the database information (data component) if the user requests a notification (control component). So, for the use case {U1.5}, the components that are kept are the data and control, this components give origin to the inclusion in the object model of the objects {O1.5.c} and {O1.5.d}.

Micro-step 2iii: object naming

The objects that have not been eliminated in micro-step 2ii must be named. The name received by the object must reflect its role taking into account the main component and reflect the use case which it comes from.

As an example, the objects {O1.5.c} and {O1.5.d} receive the name *"notification processor"* and *"request notification data"*, respectively, since they both originate from use case {U1.5} *process notification request.*

Micro-step 2iv: object description

After naming the objects, they need to be described. The idea of this description is to include in the object model the system requirements represented by the objects. The descriptions must be based in the original use case textual descriptions and introduce requirements in this stage is not recommend except if they are classified as non-functional requirements (NFR) or as design decisions (DD). One example of an object description is the description of {O1.5.c}.

> ***{O1.5.c} notification processor:*** *this object is responsible to check if the user has request a notification when a complaint status has been changed.*

Micro-step 2v: object representation

The fifth micro-step is the most critical of the 4SRS technique. In this micro-step the redundancy of the user requirements elicitation is eliminated and the missing requirements are supposed to be detected.

In this step, it must be checked for each object if it can be represented by other one, in order to reduce redundancy. This means that the representative object can represent its own system requirements and the requirements of the represented objects.

In the example, object {O1.2.2.i} *select option interface* that result from use case {U1.2.2} *select complaint option* is represented by object {O1.2.1.i} *user options interface*. This representation means that object {O1.2.1.i} can represent a system-level entity within the object model that is able to accommodate its own system requirements and the requirements of {O1.2.2.i}. This is an example of elimination of functional redundancy, since, from the functional point of view, it is simpler and more correct to support and justify the existence of a unique data repository for all kinds of registered services.

Micro-step 2vi: global elimination

This micro-step is straight forward (and thus is easy to automate), since it only consists on the application of the results of the last micro-step. The objects that are represented by others must be eliminated because their system requirements are now part of the representative object. This micro-step is called "global elimination" due to its global awareness for generating a coherent and cohesive object model, from the system requirements point of view.

In the case study object {O1.2.2.i} must be eliminated because it is represented by object {O1.2.1.i}.

Micro-step 2vii: object renaming

This is the last micro-step and its idea is to rename the objects that were not eliminated in micro-step 2vi and that represent other objects. The new name must reflect all the represented system requirements.

In the case study, object {O1.2.1.i} was renamed, because it was not eliminated in micro-step 2vi and it represents not only itself but also objects {O1.2.2.i}, {O1.2.3.i}, {O1.2.4.i} and {O1.2.5.i}. The new name is *"complaint interface"*.

3.3.3. Step 3: object packaging & aggregation

In this step, the remaining objects (i.e., the objects that survive in step 2), for which there is some advantage in being treated in a unified way, should be aggregated or packaged. These aggregations or packages should include semantically consistent objects. In this step a truly coherent object model is created, since it allows the introduction of another semantic layer in a higher abstraction level which works like "functional glue" for the objects.

The packing technique is still a relatively immature technique (that requires further work). It introduces a very light semantic cohesion among objects that can be easily reversed in the design phase if needed. It can be used to make a more comprehensive and understandable object model.

The aggregation imposes a strong semantic cohesion among objects that is more difficult to reverse in the design phase. Aggregation should only be used when it is explicitly assumed that the set of considered objects is affected by a design decision. It is only used when there is a part of the system that constitutes a legacy sub-system, or when the design has a pre-defined reference architecture that restricts the object model.

As an example, objects {O1.2.i}, {O1.3.2.i} and {O1.7.1.i} were all obtained from different use cases, and are all packaged inside package {P1} *user interface*. In the case study, no aggregation was used.

3.3.4. Step 4: object linkage

The fourth and final step of the 4SRS introduces the links in the object model. These links are strongly based on the textual information of the use cases and the information generated in micro-step 2i.

The textual descriptions of the use cases can give some hints on the links to introduce in the object model namely if there is some care in relating use cases. As an example, the textual description of the use case {U1.5} includes the next sentence "*The complaint status is updated via {1.4} fill complaint information. The user information was recorded in the service via {1.2.5} send user information*". With this information, the object model can include the links from object {O1.5.c} to object {O.1.4.d} and to object {O1.2.5.d}. These links are a pre-design decision related to the sequence of the execution of use cases.

The use case model can include other type of information that gives support to the links. It is the case of UML relations between use cases which can be used in the use case diagrams and express the relationships among them. As an example, use case {U1.3} <<uses>> use case {U1.2}, which means that use case {U1.3} can execute use case {U1.2}which justifies the link between {O1.2.5.d} and {O1.3.1.c}.

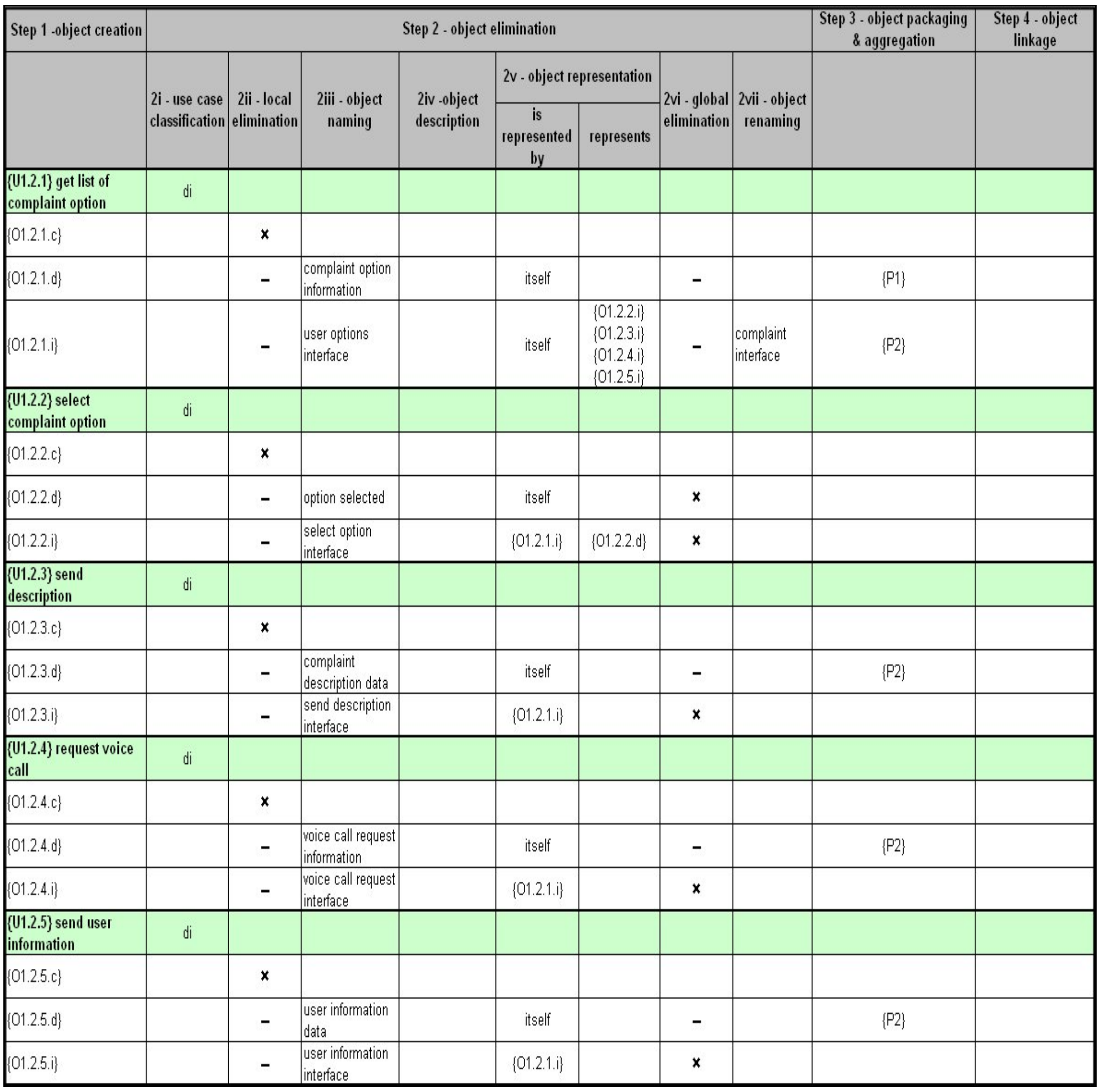
Using the information generated in micro-step 2i, links originally obtained from the same use case must be included in the object model. As an example, in step 2i, use case {U1.2.1} was classified as "di", which implies that objects {O1.2.1.i} and {O1.2.1.d} must be linked.

3.4. Tabular Transformation

To better help the designers with the usage of the 4SRS a tabular representation was conceived to guide on the execution of the transformation steps. The table structure (namely the contents to put in the rows and columns) is explained in this section through 3 instances of tables (figure 3-5). Section 4 presents the full usage of the tables in two examples, that better illustrate the utility of this tabular transformation. During the execution of each step the information and decisions taken will be stored in the table (figure 3). In the table there is one column for each step or micro-step and there is one row for each use case.

The first column corresponds to the execution of Step 1 of the 4SRS. The first row contains the reference and the name of the use case. The next three rows are inserted for recording the information related to one control, one data and one interface object for the corresponding use case. This set of 4 rows is repeated for each use case. Figure 3 depicts 4 different rows for each of the 5 use cases exemplified (in total there are 20 rows).

Micro-Step 2i is represented in the second column. In the first row of each use case, the classification of the use case is inserted. As depicted in the example (figure 3) use case {U1.2.1} *get list of complaint option* was classified as “di”.



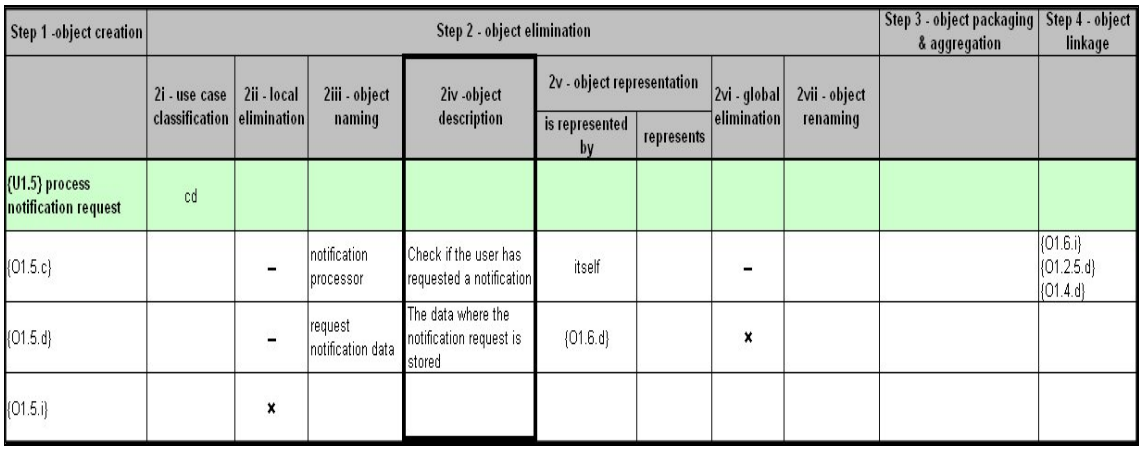
Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation		2vi - global elimination	2vii - object renaming		
					is represented by	represents				
{U1.2.1} get list of complaint option	di									
{O1.2.1.c}		×								
{O1.2.1.d}		–	complaint option information		itself		–		{P1}	
{O1.2.1.i}		–	user options interface		itself	{O1.2.2.i} {O1.2.3.i} {O1.2.4.i} {O1.2.5.i}	–	complaint interface	{P2}	
{U1.2.2} select complaint option	di									
{O1.2.2.c}		×								
{O1.2.2.d}		–	option selected		itself		×			
{O1.2.2.i}		–	select option interface		{O1.2.1.i}	{O1.2.2.d}	×			
{U1.2.3} send description	di									
{O1.2.3.c}		×								
{O1.2.3.d}		–	complaint description data		itself		–		{P2}	
{O1.2.3.i}		–	send description interface		{O1.2.1.i}		×			
{U1.2.4} request voice call	di									
{O1.2.4.c}		×								
{O1.2.4.d}		–	voice call request information		itself		–		{P2}	
{O1.2.4.i}		–	voice call request interface		{O1.2.1.i}		×			
{U1.2.5} send user information	di									
{O1.2.5.c}		×								
{O1.2.5.d}		–	user information data		itself		–		{P2}	
{O1.2.5.i}		–	user information interface		{O1.2.1.i}		×			

Figure 3. 4SRS tabular representation

The third column corresponds to the local elimination step (micro-step 2ii). The objects that do not make sense in the problem domain must be eliminated, which is signalled by putting an “×” in the corresponding cell. The objects that are to be kept (objects that make sense in the problem domain) are marked with a “–”. In figure 3, for {U1.2.1} the control object must be eliminated, because it does not make sense in the problem domain. So objects {O1.2.1.d} and {O1.2.1.i} are marked with “–” and object {O1.2.1.c} is marked with “×”.

The fourth column corresponds to micro-step 2iii. The objects that have not been eliminated in the previous micro-step must receive a proper name. In the case study (figure 3), for instance, object {O1.2.1.d} was named *complaint option information*.

The fifth column corresponds to micro-step 2iv. In this step each named object in the previous micro-step has to be described. The description is inserted in the table as depicted in the figure 4, using typically text written in a natural language.



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				
{U1.5} process notification request	cd									
{O1.5.c}		-	notification processor	Check if the user has requested a notification	itself		-			{O1.6.i} {O1.2.5.d} {O1.4.d}
{O1.5.d}		-	request notification data	The data where the notification request is stored	{O1.6.d}		×			
{O1.5.i}		×								

Figure 4. Example of an object description

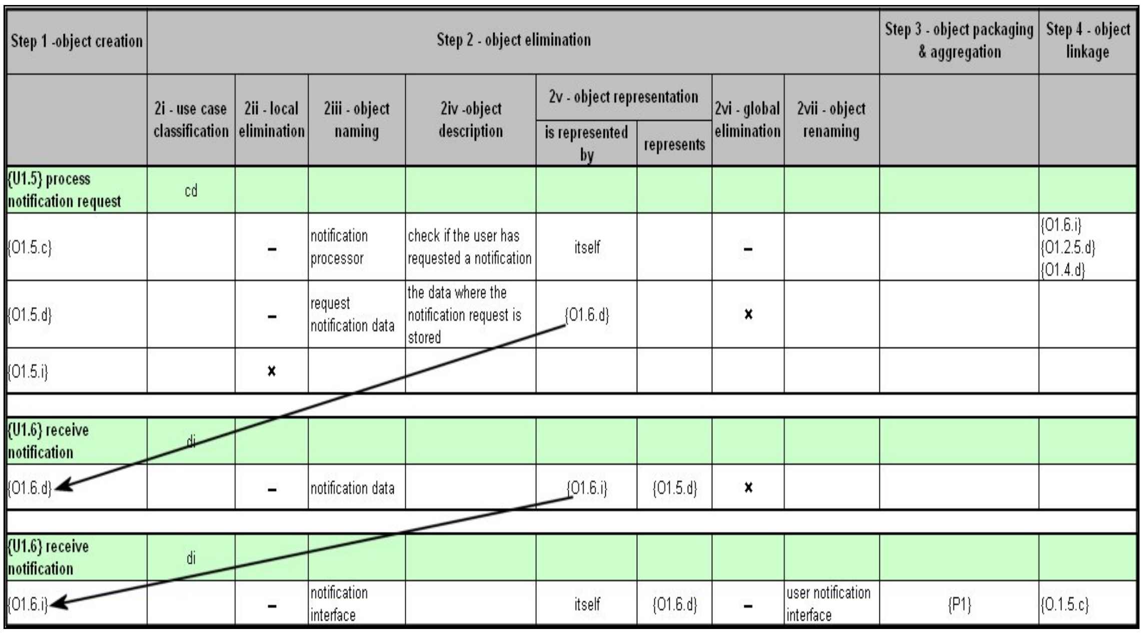
The next step is the most complex step of the technique. This micro-step is divided in two columns: the "***is represented by***" column (sixth column) and the "***represents***" column (seventh column).

In the "***is represented by***" column the reference of the object that represents the object being analysed is stored. If the object that is being analysed is represented by itself, then its "***is represented by***" column must refer to itself (by writing "itself").

In the "***represents***" column, the references of the objects that the object in analysis represents are stored.

As an example, in figure 5, we can see that object {O1.5.c} is represented by itself, and that object {O1.5.d} is represented by object {O1.6.d}. However, object {O1.6.d} is represented by object {O1.6.i} (and represents object {O1.5.d}) and object {O1.6.i} is represented by itself and represents object {O1.6.d}.

In another example (figure 3) we can see that object {O1.2.1.i} is represented by itself and represents objects {O1.2.2.i}, {O1.2.3.i}, {O1.2.4.i} and {O1.2.5.i}.



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				
{U1.5} process notification request	cd									
{O1.5.c}		–	notification processor	check if the user has requested a notification	itself		–			{O1.6.i} {O1.2.5.d} {O1.4.d}
{O1.5.d}		–	request notification data	the data where the notification request is stored	{O1.6.d}		×			
{O1.5.i}		×								
{U1.6} receive notification	di									
{O1.6.d}		–	notification data		{O1.6.i}	{O1.5.d}	×			
{U1.6} receive notification	di									
{O1.6.i}		–	notification interface		itself	{O1.6.d}	–	user notification interface	{P1}	{O.1.5.c}

Figure 5. Micro-step 2v tabular execution

Micro-step 2vi is represented in the eighth column. The objects that are represented by another object must be eliminated. These objects must be marked with an “×”. The objects that are represented by themselves remain in the model and should be marked with an “–”. As an example, figure 5, shows that object {O1.5.c} is kept and is marked with an “–”, while object {O1.5.d} is eliminated and is marked with an “×”.

The ninth column corresponds to micro-step 2vii. The objects that have not been eliminated and represent other objects must be renamed. Figure 5 shows that object {O1.6.i} *notification interface* represents object {O1.6.d} and that it was renamed to *user notification interface*.

The tenth column corresponds to step 3 - object packaging & aggregation. If an object is packaged or aggregated, the reference of the corresponding package or aggregation must be inserted in the corresponding cell of the tenth column. As depicted in figure 5, object {O1.6.i} is included (i.e. packaged) into {P1}.

Step 4 is represented by the eleventh column. If an object has any links with other objects, the references of the associated objects must be inserted in this column.

Figure 5 shows that object {O1.5.c} has links with objects {O1.6.i}, {O1.2.5.d} and {O1.4.d}.

3.5. Conclusions

In this chapter, the new version of the Four-Step Rule Set is presented. This technique is used to build object diagrams which accommodate the user requirements acquired in previous development stages. It is a mechanism that helps software engineers to transform user requirement models into the first logical architecture of the system.

The improved version of the 4SRS technique tries to tackle the problem of the subjective interpretation by dividing the elimination step in several micro-steps. It helps also to solve the complexity of this step with this micro-step division.

To help the software engineers executing the 4SRS technique it a table where the information of each step is stored was also presented. During the execution of the technique, all decisions and the significant information is stored in the table. Having the information in the table helps the software engineer since it allows the trace of the decisions taken and stored in a structured way. In order words, it allows the software engineers to relate each object with the use case that originates it. The software engineer after executing all steps must build the object diagram and to accomplish that purpose he has only to use information in the table.

Chapter

4

Validation of the 4SRS

Summary

To illustrate the usage of the 4SRS technique, its application to one case study is explained in this chapter. The two different usages that are considered for analysis are both taken from the USE-ME.GOV project (a new open platform for mobile government applications responsible for the deployment of a set of reusable service components that facilitates user context-aware application): (1) the USE-ME.GOV platform specification; and (2) the USE-ME.GOV AVAccess service specification. The second usage (service specification) corresponds to the recursive application of the 4SRS technique to the first one (platform specification).

4.1. Introduction

The USE-ME.GOV is a new open platform for mobile government applications, supporting usability, openness, interoperability and scalability. It is responsible for the deployment of a set of reusable service components that facilitates user context-aware application development that allow citizen to perform a set of high-level government related activities, allowing citizens to access the most appropriate services at any time, anywhere. Those activities include spontaneous community interactions such as reporting anomalies and making suggestions or getting thematic information on local events, city information, healthcare or education.

The USE-ME.GOV's AVAccess service is a refinement of a package obtained in the object diagram of the USE-ME.GOV platform specification. The AVAccess service is the platform component where all user requests are redirected when the request does not specify any Service. This component is a single point of contact and should redirect the user to the appropriate end-service. In most cases user starts the interaction with the system by contacting this component. The AVAccess component should serve as the single point for user data management, user profile management, subscription management, as well as navigation among Services.

4.2. Platform Specification

The first usage of the 4SRS technique analysed in this document is the USE-ME.GOV platform specification.

To capture the functional view of the system, according to the users perspective, several use case diagrams were created. Figures 6 represent the top-level view of the system. After identifying all the use cases of the system, the next step is to describe their behaviour. Next, the description of the top-level use case {U0a.1} with informal text is presented as an example. Similar descriptions were created for the other top-level use cases.

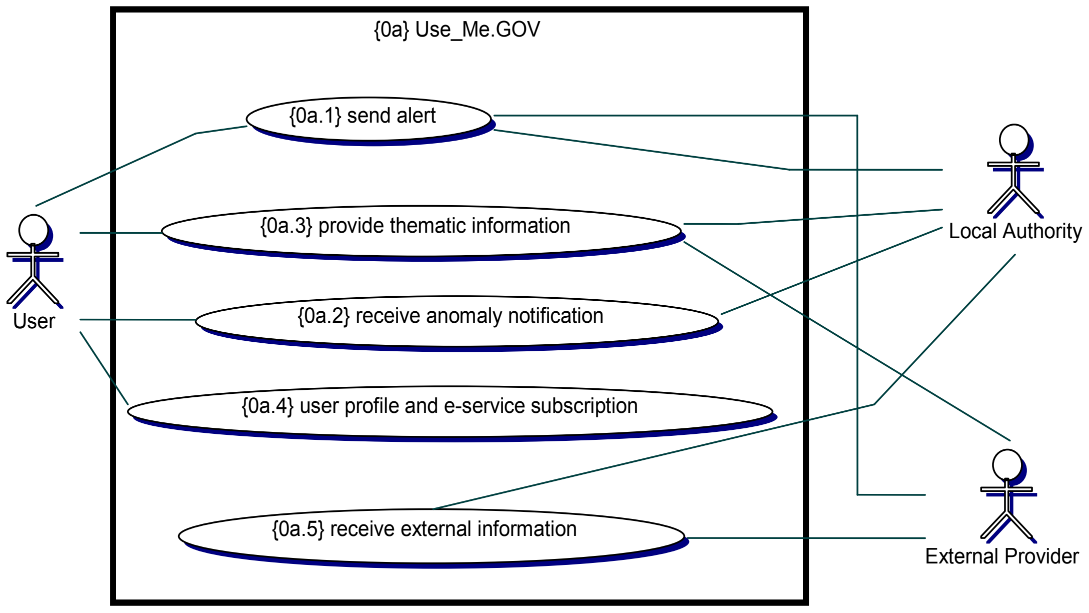

{0a} Use_Me.GOV
{0a.1} send alert
{0a.3} provide thematic information
{0a.2} receive anomaly notification
{0a.4} user profile and e-service subscription
{0a.5} receive external information
User
Local Authority
External Provider

Figure 6. Level zero use case diagram of USE-ME.GOV

> ***{0a.1} send alert****: Send domain alert or disseminating domain information to the users informing of domain related events and situations or unexpected domain situations that are happening in the region. Only users that have previously subscribed this e-service will receive the alert messages (subscription made via {0a.4.} user profile and e-service subscription). This is an asynchronous e-service. If technically possible, the system acquires user context raw information (location, time, activity, preferences, etc) from external context sources. Also, a contextualisation process will assist the system in making the level of granularity of the information adequate to the geographic location of the user context (geographic location context, time context and activity context): For example: ((#1) an alert of a dangerous hole in a street should only be send to the users geographically located in that street; (#2) an alert of a street obstructed should be send to the users geographically located in that street or in any of the incident streets; (#3) an alert of weather storm should be send to all the users geographically located in the region). The information associated to the alert should always be updated and match the user-specific request, excluding any extra information or undesired advertisements. For those users that require personalised information, a subscription must be made via {0a.4.} user profile and e-service subscription.*

Some use cases were refined due to their complexity. The refinements are next presented and discussed.

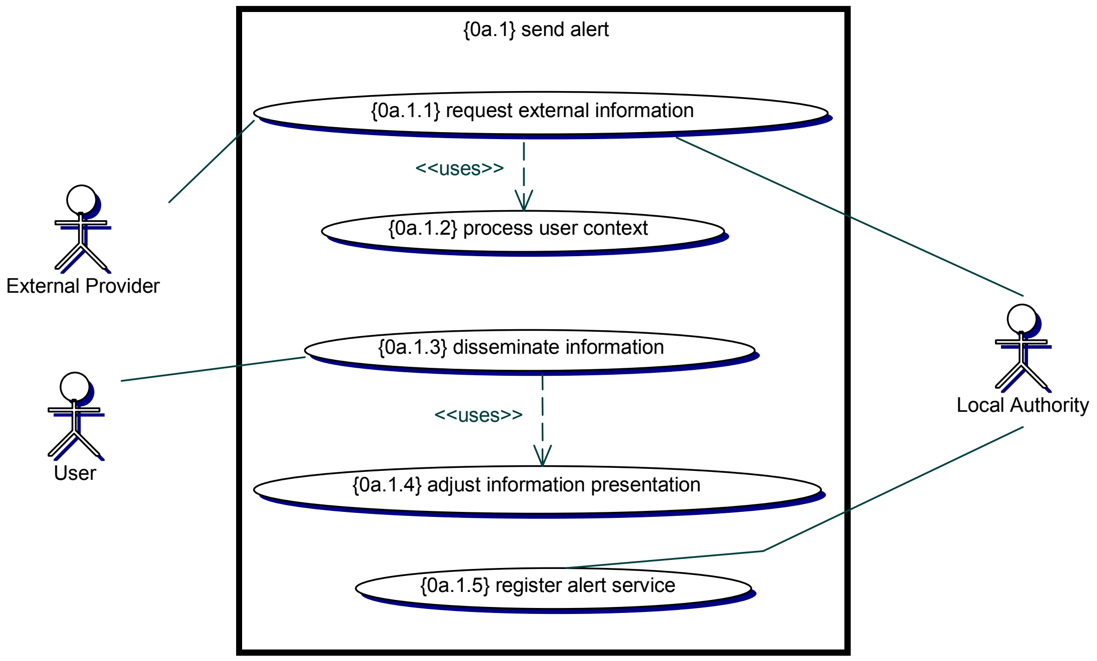

{0a.1} send alert
{0a.1.1} request external information
<<uses>>
{0a.1.2} process user context
{0a.1.3} disseminate information
<<uses>>
{0a.1.4} adjust information presentation
{0a.1.5} register alert service
External Provider
User
Local Authority

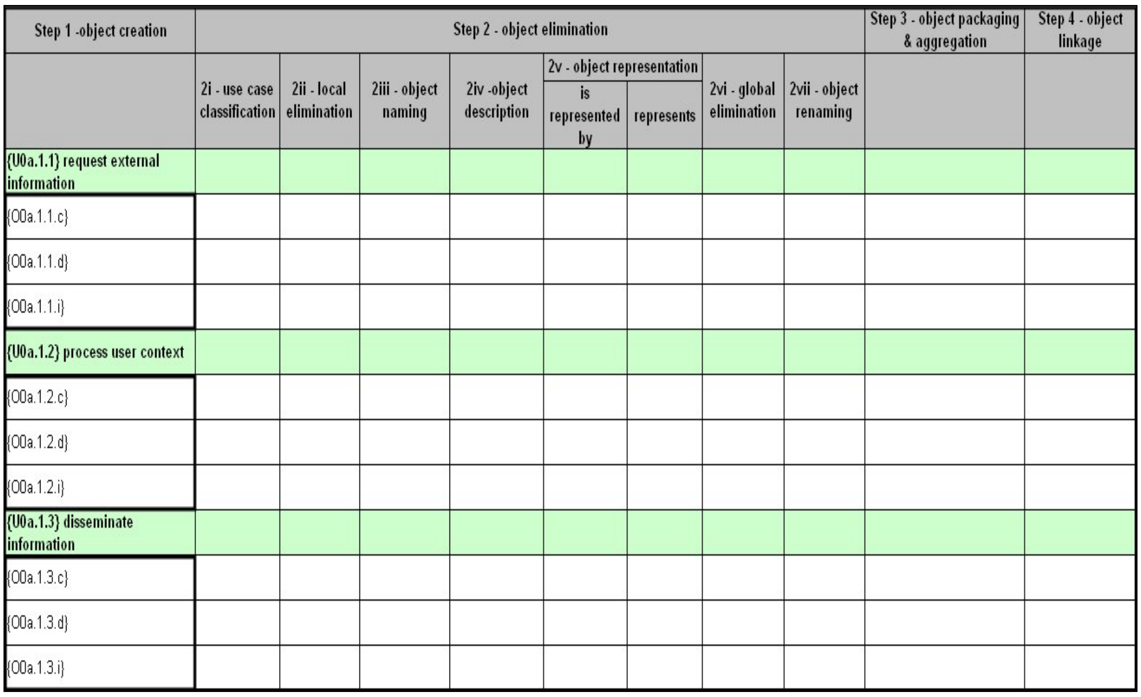
Figure 7. Level one use case diagram of send alert use case

> ***{0a.1.1} request external information****: The system explicitly requests for appropriate domain information, from the known external content*

providers (registered via {0a.1.5} register alert service, for example), based on user context (location, time and activity), acquired and processed via {0a.1.2} process user context, and/or based on user profile, processed via {0a.4} user profile and e-service subscription.

***{0a.1.5} register alert service**: the system registers domain alert services. When a local authority wants to offer an alert service, it registers it in the system providing the alert conditions and source of information. This registration will allow the execution of the following use cases: {0a.1.3} disseminate information and {0a.1.1} request external information.*

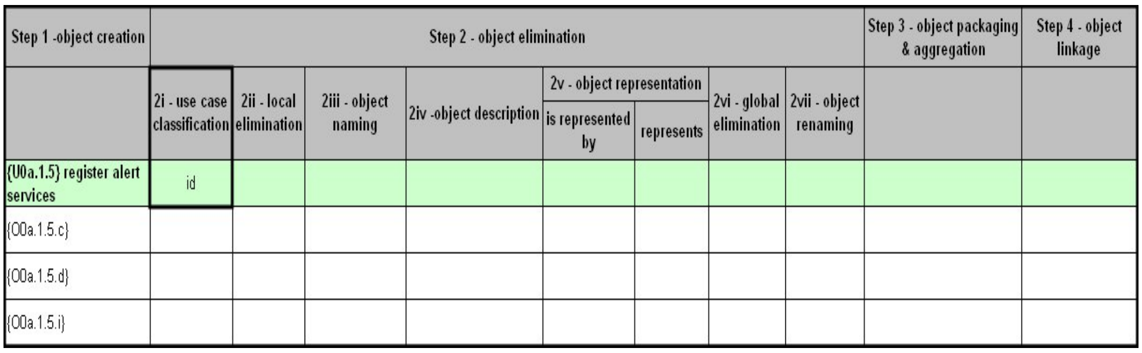
After defining the use cases, it is now time to start the model transformations. In this case study the model transformations of the 4SRS were applied to the 2nd level of the use case refinement. So, the application of the first step is applicable to all the 22 2nd level use cases (5 from {U0a.1}, 7 from {U0a.2}, 8 from {U0a.3} and 2 from {U0a.4}) and to use case for {U0a.5} that has no refinements. This first step gives origin to 69 objects, since the number of use cases to considerer is 23. Part of the tabular information for this step is presented in figure 8.



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				
{U0a.1.1} request external information										
{O0a.1.1.c}										
{O0a.1.1.d}										
{O0a.1.1.i}										
{U0a.1.2} process user context										
{O0a.1.2.c}										
{O0a.1.2.d}										
{O0a.1.2.i}										
{U0a.1.3} disseminate information										
{O0a.1.3.c}										
{O0a.1.3.d}										
{O0a.1.3.i}										

Figure 8. Step 1 execution – object creation for platform specification

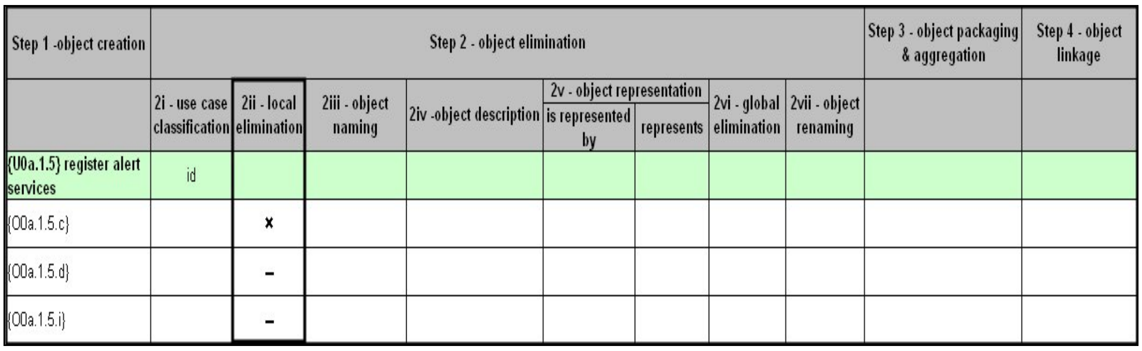
After creating a set of 3 objects the next step is to execute step 2 - object elimination. As explained before, this step is divided into several micro-steps. The first micro-step (micro-step 2i) classifies the use cases. So for each considered use case, we have to classify it with one of the 8 patterns (Ø, i, c, d, ic, di, cd, icd). To illustrate this micro-step, figure 9 shows one example of classification to use case {U0a1.5}. This use case was classified has being of the type “id” because to represent it only the interface and the data object are needed. The same reasoning was taken for the other use cases.



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				
{U0a.1.5} register alert services	id									
{O0a.1.5.c}										
{O0a.1.5.d}										
{O0a.1.5.i}										

Figure 9. Micro-step 2i execution for platform specification

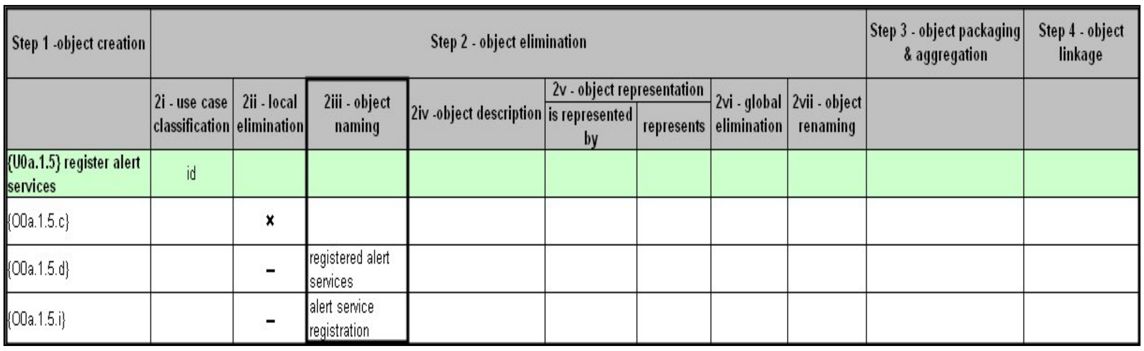
After classifying all the use cases, we must proceed to micro-step 2ii, which eliminates, within the 3 objects for each use case, those that are not considered by the classification of the use case. For the example we have to check, for each of the 69 objects created in step 1, if it makes sense in the problem domain, i.e., check if the object is needed taking into account the problem domain. In this case study we decided to eliminate 13 objects, because they do not make sense. As an example of this elimination let us look to the objects of use case {U0a1.5}. In this use case the control object ({O0a.1.5.c}) does not make sense. This decision was taken analyzing the textual descriptions. From the textual description of use case {U0a.1.5} we can extract the information that this use case is only responsible for allowing the registration of services by the local authority (this corresponds to the interface object) and the maintenance of the data repository of registered services (this corresponds to the data object). So, to this use case only objects {O0a.1.i} and {Oa0.1.5.d} have been kept. In figure 10 we can see that the object that we decided to eliminate is marked with an “×” and the objects that we decide to keep are marked with an “–”. Similar decisions were taken to the remaining objects.



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				
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		-								
{O0a.1.5.i}		-								

Figure 10. Micro-step 2ii execution for platform specification

Now we only have 56 objects, which are the ones that apparently make sense in the problem domain. For all of these objects we have to give a name (execution of the micro-step 2iii). Taking the example of objects {O0a.1.5.i} and {O0a.1.5.d} we named these objects as *"registered alert services"* and *"alert service registration"*, respectively. Figure 11 shows the table filled with this information.



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				
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		-	registered alert services							
{O0a.1.5.i}		-	alert service registration							

Figure 11. Micro-step 2iii execution for platform specification

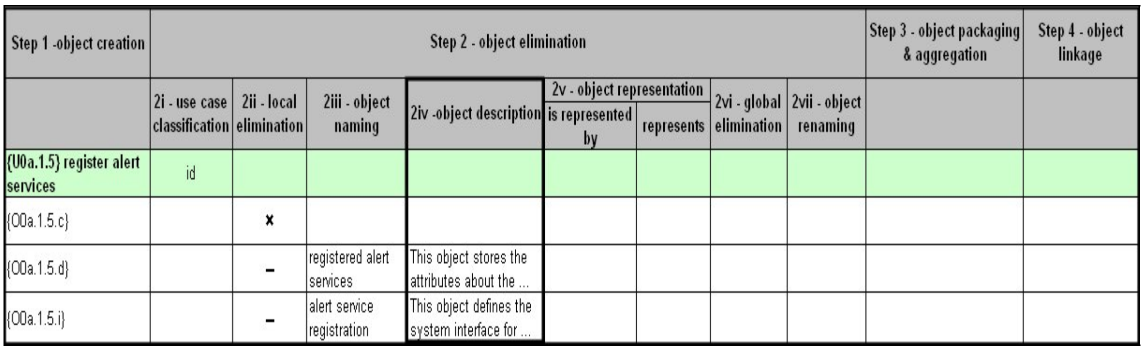
The same objects that were named must now be described (micro-step 2iv). Using again objects {O0a.1.5.d} and {O0a.1.5.i} as an example, we must present their descriptions.

> ***{O0a.1.5d} registered alert services:*** *This object stores the attributes about the services registered in the system (by "service" in this context we refer to the application functionalities that the system will offer to its users). This object must store the attributes that will enable service discovery, service aggregation, service automatic activation (when applied) and service access. A proper service description mechanism should be introduced. NFR1: This mechanism should be extensible, manageable, self explaining and semantically uniform. For service automatic activation purposes, this object must store information about internal and external alert conditions. This object must store information*

about the dependencies between registered services and external content providers. For context-aware service discovery, this object must store attributes defining the service context of use. NFR2: For scalability constraints, this must be a distributed object, mainly in a regional, national or international setting.

{O0a.1.5i} alert service registration: *This object defines the system interface for service registration, deregistration, dynamic configuration and service attributes query. NFR1: For security constraints this interface must only be provided to the system components (not to the users).*

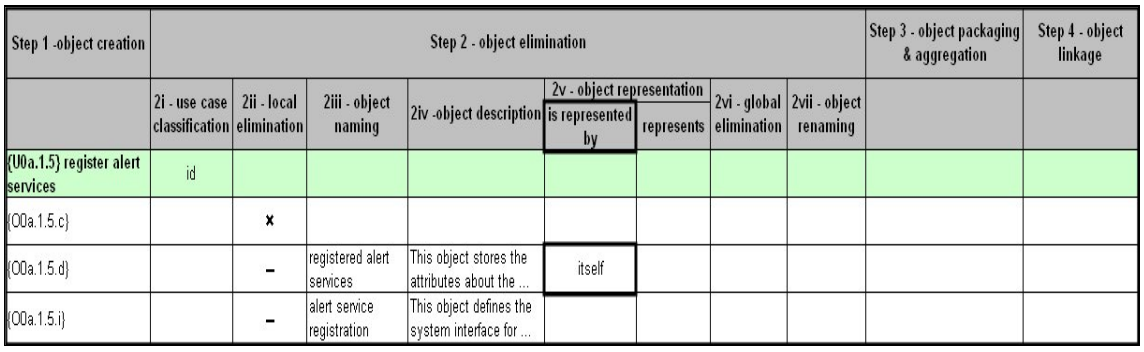
Figure 12 shows the description for objects {O0a.1.5.d} and {O0a.1.5.i}. For readability purposes only a small fraction of the text is shown.



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: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		-	registered alert services	This object stores the attributes about the ...						
{O0a.1.5.i}		-	alert service registration	This object defines the system interface for ...						

Figure 12. Micro-step 2iv execution for platform specification

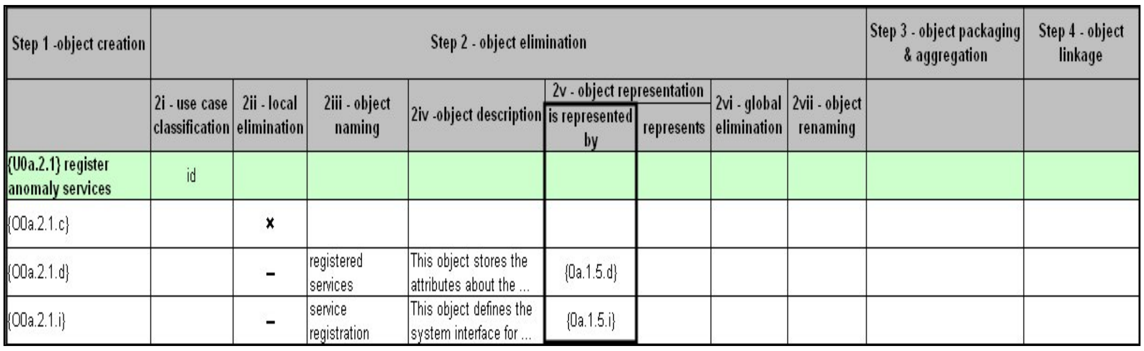
The next micro-step to execute is 2v (object representation) which is considered as the most critical one. In this case study after analysing the requirements it was decided that several objects can be represented by others. For example object {O0a.1.5.d} *registered alert services* can represent a system-level entity in the object model that is able to accommodate its own requirements, and it was decided that this object must be represented by itself. So, in the table (figure 13) the cell "*is represented by*" for object {O0a.1.5.d} is marked with "itself".



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: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		-	registered alert services	This object stores the attributes about the ...	itself					
{O0a.1.5.i}		-	alert service registration	This object defines the system interface for ...						

Figure 13. Micro-step 2v execution for platform specification

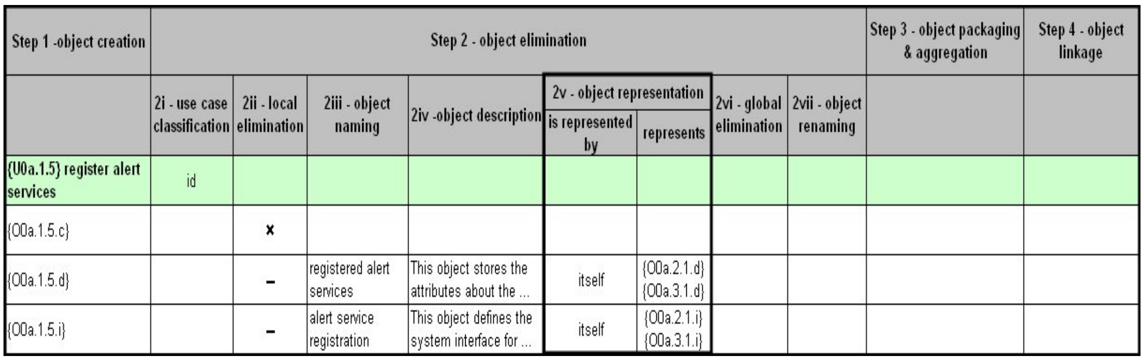
Carrying on with the object representation step, it was decided that object {O0a.2.1.d} *registered anomaly services* was represented by object {O0a.1.5.d}. This means that object {O0a.1.5.d} includes not only its own system requirements, but also the requirements of object {O0a.2.1.d}.In this case, we need to fill the cell "*is represented by"* for object {O0a.2.1.d} with "{O0a.1.5.d}" (figure 14) and to make the table complete and traceable fill also the cell *"represents"* for object {O0a.1.5.d} with "{O0a.2.1.d}" (figure 15).



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: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U0a.2.1} register anomaly services	id									
{O0a.2.1.c}		×								
{O0a.2.1.d}		–	registered services	This object stores the attributes about the ...	{0a.1.5.d}					
{O0a.2.1.i}		–	service registration	This object defines the system interface for ...	{0a.1.5.i}					

Figure 14. Micro-step 2v execution for platform specification

In figure 15 we can also see in the *"represents"* cell for object {O0a.1.5.d} the reference to object {O0a.3.1.d}, since this object, like {O0a.2.1.d}, is represented by {O0a1.5.d}.

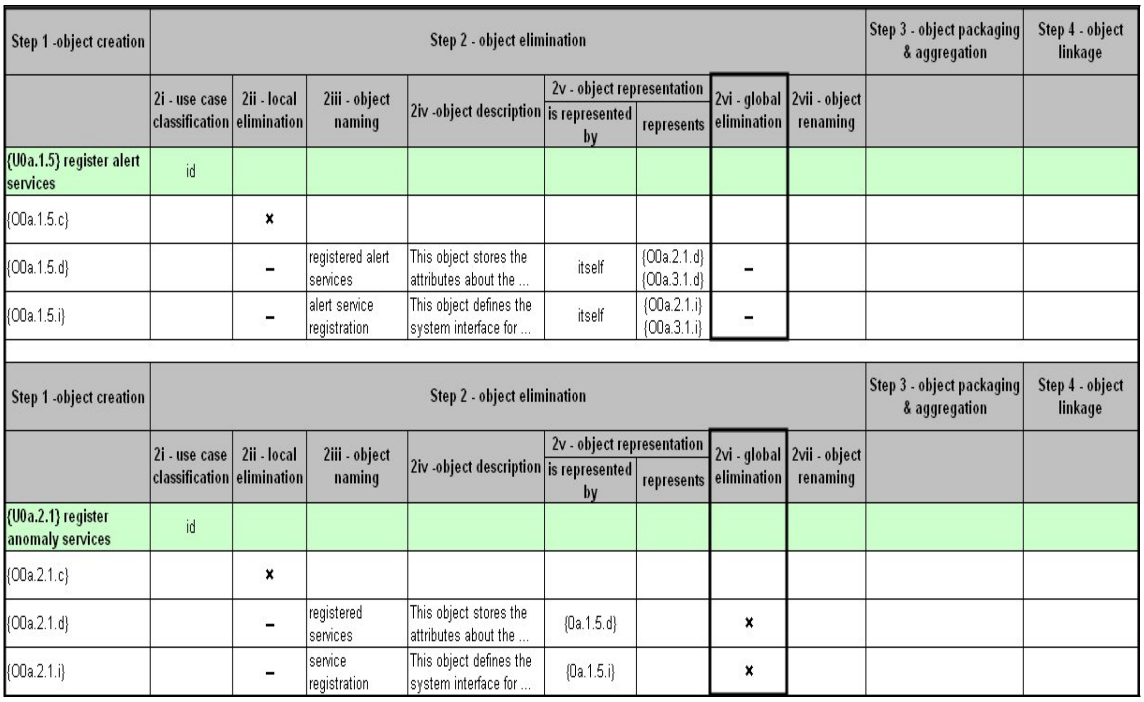


Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		–	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}				
{O0a.1.5.i}		–	alert service registration	This object defines the system interface for ...	itself	{O0a.2.1.i} {O0a.3.1.i}				

Figure 15. Micro-step 2v execution for platform specification

We have here an example of elimination of functional redundancy, since it seems more correct and simpler from an architecture point of view to justify and support the existence of a unique database for all kinds of registered services.

After finding out all the object representations we must execute the global eliminations. This global elimination is executed based on the results of the last step. The objects that are represented by other object can be easily eliminated. The objects to be eliminated are marked in the table with an "×". The objects that are represented by themselves must be kept and are marked with an "–". In figure 16, the execution of this step to the objects presented in the last step({O0a.1.5.d}, {O0a.2.1.d}) is shown.

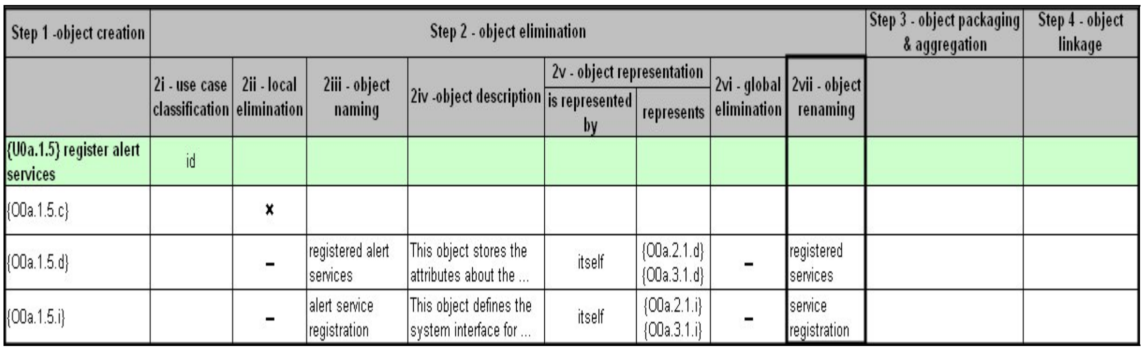


Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation		2vi - global elimination	2vii - object renaming		
					is represented by	represents				
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		–	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}	–			
{O0a.1.5.i}		–	alert service registration	This object defines the system interface for ...	itself	{O0a.2.1.i} {O0a.3.1.i}	–			

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				
{U0a.2.1} register anomaly services	id									
{O0a.2.1.c}		×								
{O0a.2.1.d}		–	registered services	This object stores the attributes about the ...	{0a.1.5.d}		×			
{O0a.2.1.i}		–	service registration	This object defines the system interface for ...	{0a.1.5.i}		×			

Figure 16. Micro-step 2vi execution for platform specification

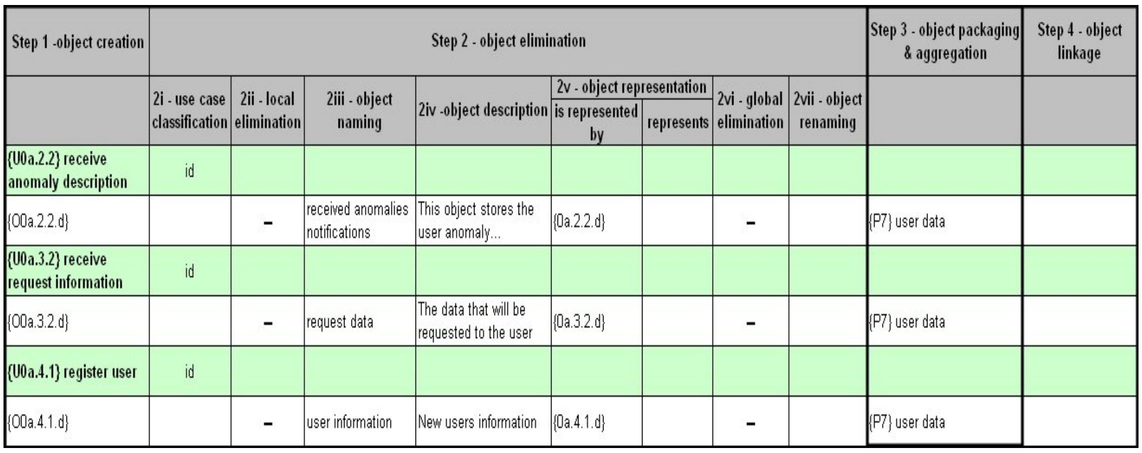
The next step to execute is the object renaming. This step is only applicable to the objects that represent other objects. In the examples that we have used to illustrate this technique the object that can be considered for renaming is object {O0a.1.5.d}. The chosen name was *registered services*, since this object now represents the objects *"registered alert services"*, *"register anomaly services"* and *"registered thematic services"*. In the table, this step inserts into the respective cell the new name (figure 17).



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				
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		×								
{O0a.1.5.d}		–	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}	–	registered services		
{O0a.1.5.i}		–	alert service registration	This object defines the system interface for ...	itself	{O0a.2.1.i} {O0a.3.1.i}	–	service registration		

Figure 17. Micro-step 2vii execution for platform specification

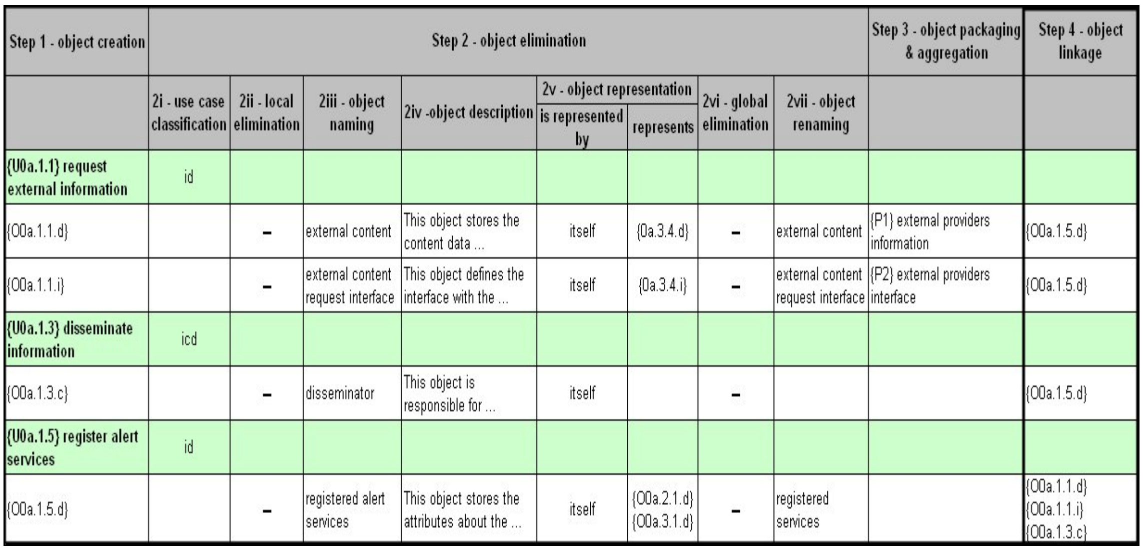
All the micro-steps of the object elimination were executed, so we can proceed to step 3 (object packaging & aggregation). In this case study no aggregation was used, because there was no reference architecture, nor any kind of legacy sub-system. As an example of packaging, objects {O0a.4.1.d}, {O0a.2.2.d} and {O0a.3.2.d}, all obtained from different use cases, are kept together in package {P7} *user data*. Transposing this information to the tabular transformation we have to insert in the line of the object the reference of the correspondent package (figure 18, to turn the image readable only the objects of the package are in the table image).



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: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U0a.2.2} receive anomaly description	id									
{O0a.2.2.d}		-	received anomalies notifications	This object stores the user anomaly...	{0a.2.2.d}		-		{P7} user data	
{U0a.3.2} receive request information	id									
{O0a.3.2.d}		-	request data	The data that will be requested to the user	{0a.3.2.d}		-		{P7} user data	
{U0a.4.1} register user	id									
{O0a.4.1.d}		-	user information	New users information	{0a.4.1.d}		-		{P7} user data	

Figure 18. Step 3 execution for platform specification

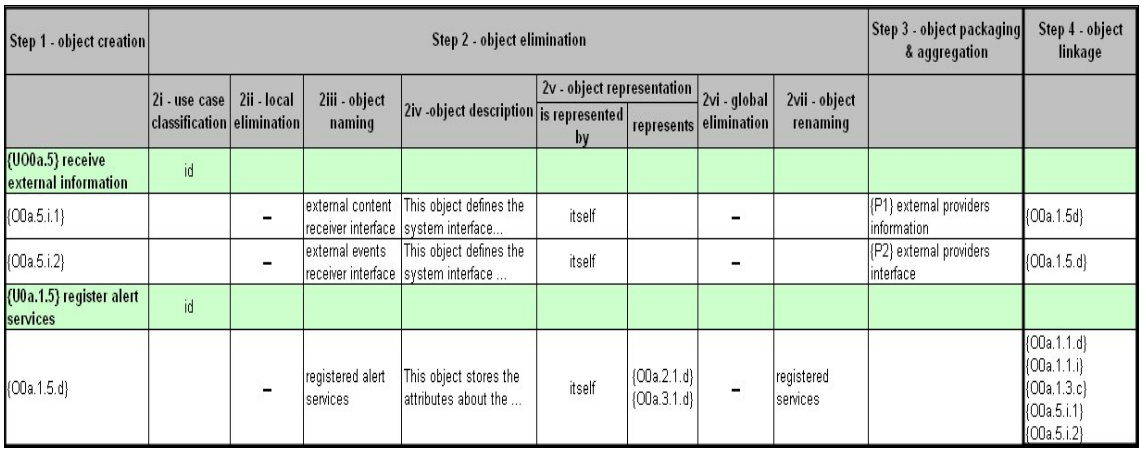
The final step of the 4SRS is the object linkage. In the case study, use case {U0a.1.5} includes in its textual description the next information "*This registration will allow the execution of the following use cases: {U0a.1.3} disseminate information and {U0a.1.1} request external information*". This sentence gives us enough information to include links from object {O0a.1.5.d} to objects {O0a.1.1.d}, {O0a.1.1.i} and {O0a.1.3.c}. So, in the table, the column for step 4, includes references to the linked objects, as figure 19 illustrates.



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: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U0a.1.1} request external information	id									
{O0a.1.1.d}		–	external content	This object stores the content data ...	itself	{0a.3.4.d}	–	external content	{P1} external providers information	{O0a.1.5.d}
{O0a.1.1.i}		–	external content request interface	This object defines the interface with the ...	itself	{0a.3.4.i}	–	external content request interface	{P2} external providers interface	{O0a.1.5.d}
{U0a.1.3} disseminate information	icd									
{O0a.1.3.c}		–	disseminator	This object is responsible for ...	itself		–			{O0a.1.5.d}
{U0a.1.5} register alert services	id									
{O0a.1.5.d}		–	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}	–	registered services		{O0a.1.1.d} {O0a.1.1.i} {O0a.1.3.c}

Figure 19. Step 4 execution for platform specification

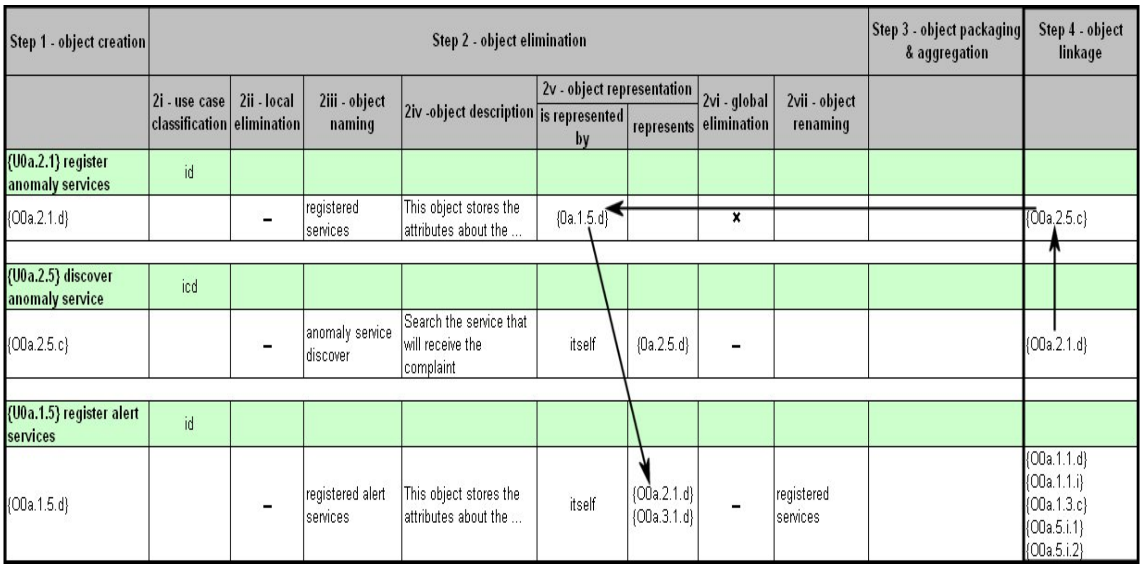
Similarly, in the textual description of use case {Ua0.5} the sentence *"These external content providers must exist in some service register (service register is made in {U0a.1.5} register alert service…"* gives hints to include a link between objects {O0a.1.5} and objects {O0a.5.i.1} and {O0a.5.i.2}.



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: is represented by	2v - object representation: represents	2vi - global elimination	2vii - object renaming		
{U00a.5} receive external information	id									
{O0a.5.i.1}		–	external content receiver interface	This object defines the system interface...	itself		–		{P1} external providers information	{O0a.1.5d}
{O0a.5.i.2}		–	external events receiver interface	This object defines the system interface ...	itself		–		{P2} external providers interface	{O0a.1.5.d}
{U0a.1.5} register alert services	id									
{O0a.1.5.d}		–	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}	–	registered services		{O0a.1.1.d} {O0a.1.1.i} {O0a.1.3.c} {O0a.5.i.1} {O0a.5.i.2}

Figure 20. Step 4 execution for platform specification

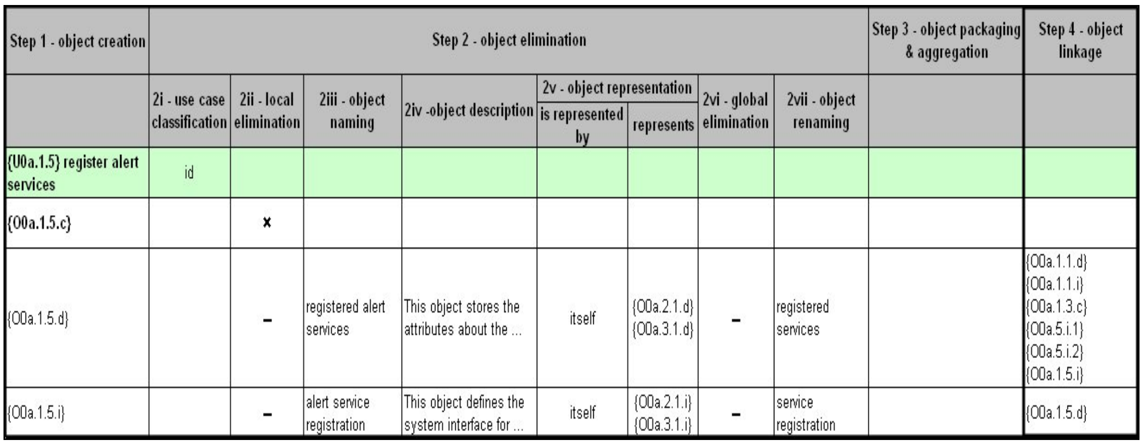
The textual description of use case {U0a.2.5} contains the next sentence: *"the system searches, from the known set of anomaly services registered (via {U0a.2.1} register anomaly service…"*. This sentence provides information that allows the inclusion of the links between objects {Oa.2.5.c} and {O0a.2.1.d}. However object {O0a.2.1.d} was eliminated, because it is represented by object {O0a.1.5.d}, so the link must be between the objects {Oa.2.5.c} and {O0a.1.5.d}.



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				
{U0a.2.1} register anomaly services	id									
{O0a.2.1.d}		-	registered services	This object stores the attributes about the ...	{0a.1.5.d}		x			{O0a.2.5.c}
{U0a.2.5} discover anomaly service	icd									
{O0a.2.5.c}		-	anomaly service discover	Search the service that will receive the complaint	itself	{0a.2.5.d}	-			{O0a.2.1.d}
{U0a.1.5} register alert services	id									
{O0a.1.5.d}		-	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}	-	registered services		{O0a.1.1.d} {O0a.1.1.i} {O0a.1.3.c} {O0a.5.i.1} {O0a.5.i.2}

Figure 21. Step 4 execution for platform specification

Concerning the information of step 2i, links between objects obtained form the same use case must be added. As an example use case {U0a.1.5} was classified as being of type "id", this means that objects {O0a.1.5.d} and {O0a.1.5.i} must be linked.



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				
{U0a.1.5} register alert services	id									
{O0a.1.5.c}		x								
{O0a.1.5.d}		-	registered alert services	This object stores the attributes about the ...	itself	{O0a.2.1.d} {O0a.3.1.d}	-	registered services		{O0a.1.1.d} {O0a.1.1.i} {O0a.1.3.c} {O0a.5.i.1} {O0a.5.i.2} {O0a.1.5.i}
{O0a.1.5.i}		-	alert service registration	This object defines the system interface for ...	itself	{O0a.2.1.i} {O0a.3.1.i}	-	service registration		{O0a.1.5.d}

Figure 22. Step 4 execution for platform specification

After completing the 4SRS execution for this case study, we can create the object diagram using the information in the table.

The diagram in figure 23 represents the USE-ME.GOV object diagram, and it identifies the high-level objects/components of the system and the relationships among them. Its purpose is to direct attention at an appropriate decomposition of the system without delving into details.

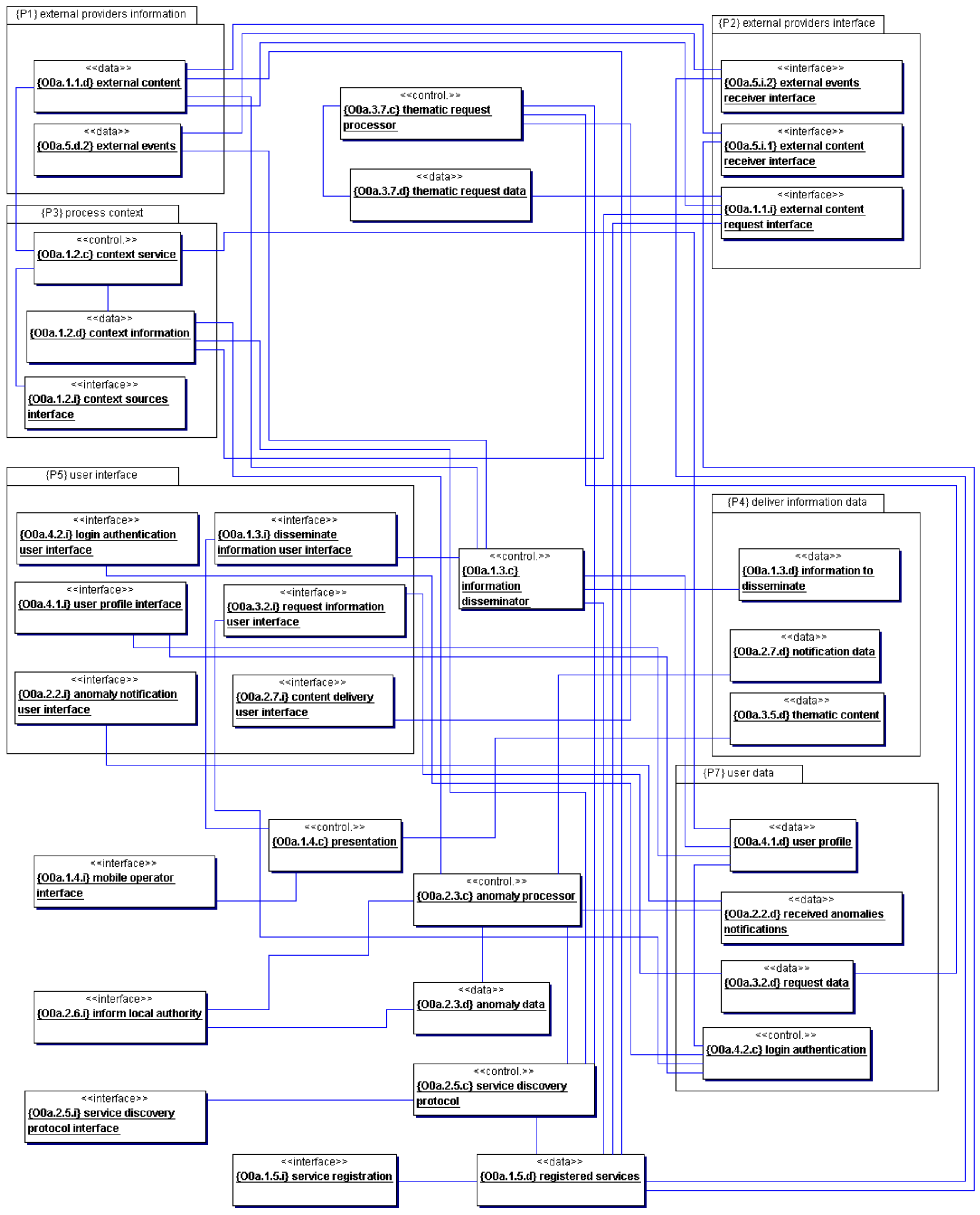

{P1} external providers information
<<data>>
{O0a.1.1.d} external content
<<data>>
{O0a.5.d.2} external events
{P3} process context
<<control>>
{O0a.1.2.c} context service
<<data>>
{O0a.1.2.d} context information
<<interface>>
{O0a.1.2.i} context sources interface
<<control>>
{O0a.3.7.c} thematic request processor
<<data>>
{O0a.3.7.d} thematic request data
{P2} external providers interface
<<interface>>
{O0a.5.i.2} external events receiver interface
<<interface>>
{O0a.5.i.1} external content receiver interface
<<interface>>
{O0a.1.1.i} external content request interface
{P5} user interface
<<interface>>
{O0a.4.2.i} login authentication user interface
<<interface>>
{O0a.1.3.i} disseminate information user interface
<<interface>>
{O0a.4.1.i} user profile interface
<<interface>>
{O0a.3.2.i} request information user interface
<<interface>>
{O0a.2.2.i} anomaly notification user interface
<<interface>>
{O0a.2.7.i} content delivery user interface
<<control>>
{O0a.1.3.c} information disseminator
{P4} deliver information data
<<data>>
{O0a.1.3.d} information to disseminate
<<data>>
{O0a.2.7.d} notification data
<<data>>
{O0a.3.5.d} thematic content
{P7} user data
<<data>>
{O0a.4.1.d} user profile
<<data>>
{O0a.2.2.d} received anomalies notifications
<<data>>
{O0a.3.2.d} request data
<<control>>
{O0a.4.2.c} login authentication
<<control>>
{O0a.1.4.c} presentation
<<interface>>
{O0a.1.4.i} mobile operator interface
<<control>>
{O0a.2.3.c} anomaly processor
<<interface>>
{O0a.2.6.i} inform local authority
<<data>>
{O0a.2.3.d} anomaly data
<<control>>
{O0a.2.5.c} service discovery protocol
<<interface>>
{O0a.2.5.i} service discovery protocol interface
<<interface>>
{O0a.1.5.i} service registration
<<data>>
{O0a.1.5.d} registered services

Figure 23. USE-ME.GOV Raw Object Diagram

4.3. Recursive Refinement

An object diagram can be collapsed, i.e., some details of the object diagram can be hidden, to transform the diagram readable. The collapsed object diagram (figure 24) is obtained from the raw object diagram (figure 23) by hiding the details of the packages. Therefore, the links appear at a higher level of abstraction and the resulting object diagram is easier to read (although it has less detail).

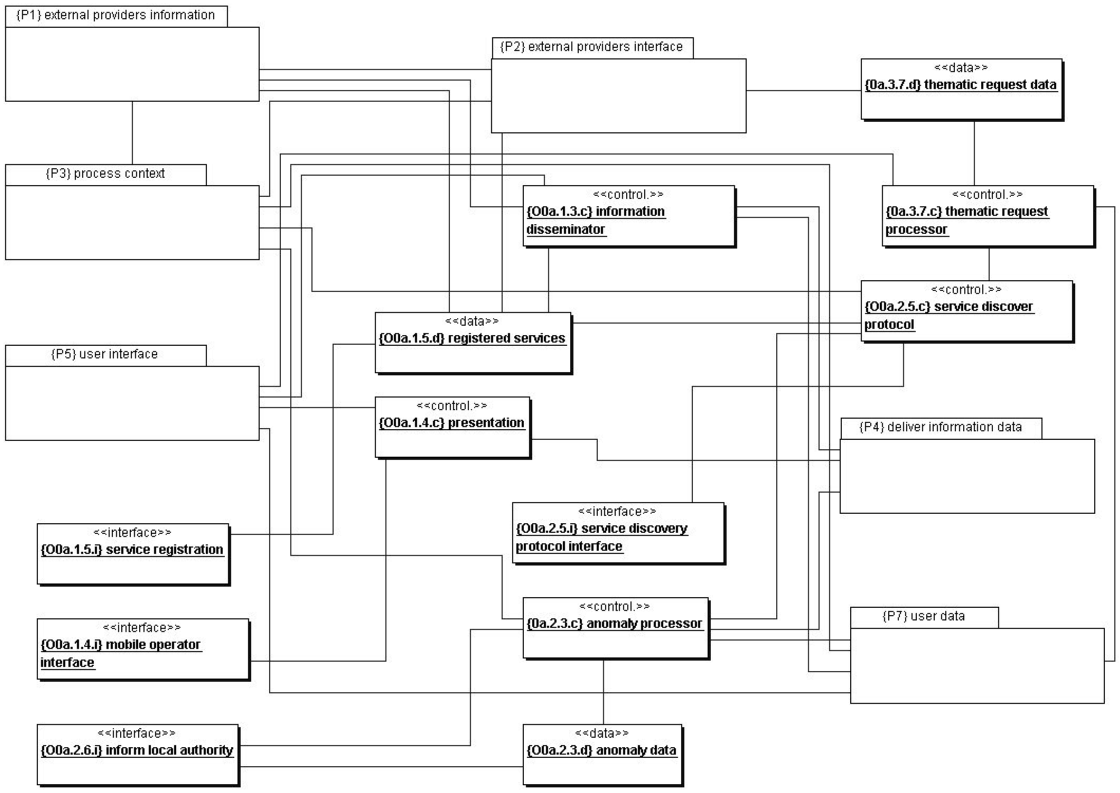

{P1} external providers information
{P2} external providers interface
<<data>>
{0a.3.7.d} thematic request data
{P3} process context
<<control.>>
{O0a.1.3.c} information disseminator
<<control.>>
{0a.3.7.c} thematic request processor
<<control.>>
{O0a.2.5.c} service discover protocol
<<data>>
{O0a.1.5.d} registered services
{P5} user interface
<<control.>>
{O0a.1.4.c} presentation
{P4} deliver information data
<<interface>>
{O0a.2.5.i} service discovery protocol interface
<<interface>>
{O0a.1.5.i} service registration
<<control.>>
{0a.2.3.c} anomaly processor
{P7} user data
<<interface>>
{O0a.1.4.i} mobile operator interface
<<interface>>
{O0a.2.6.i} inform local authority
<<data>>
{O0a.2.3.d} anomaly data

Figure 24. USE-ME.GOV Collapsed Object Diagram

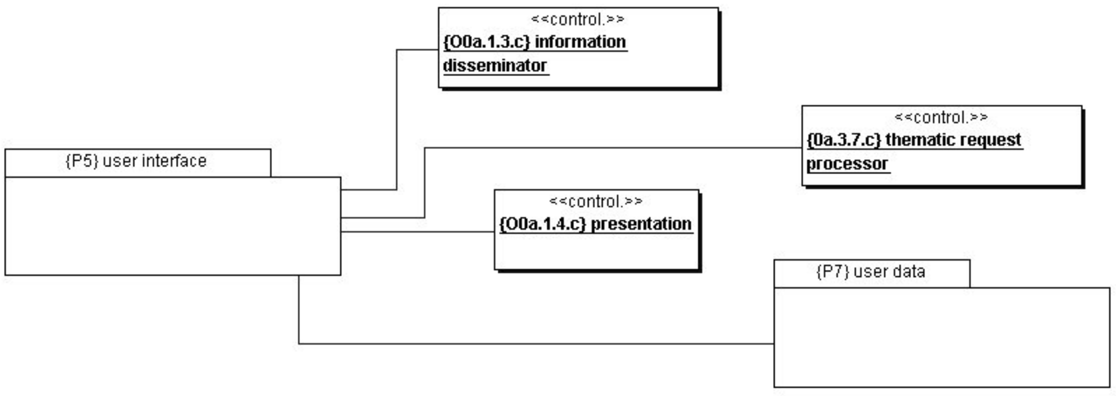


Figure 25. USE-ME.GOV Filtered Object Diagram

The diagram presented in figure 25 is a filtered object diagram. The filtered diagram was obtained from the previous one by considering the package {P5} *user interface* as one sub-system to be design. Figure 26 describes the filtering process executed over the collapsed diagram (figure 24) to obtain a *{P5}*-centric filtered object diagram. During this filtering process, all the entities that are not directly connected to package {P5} must be eliminated from the result filtered object diagram.

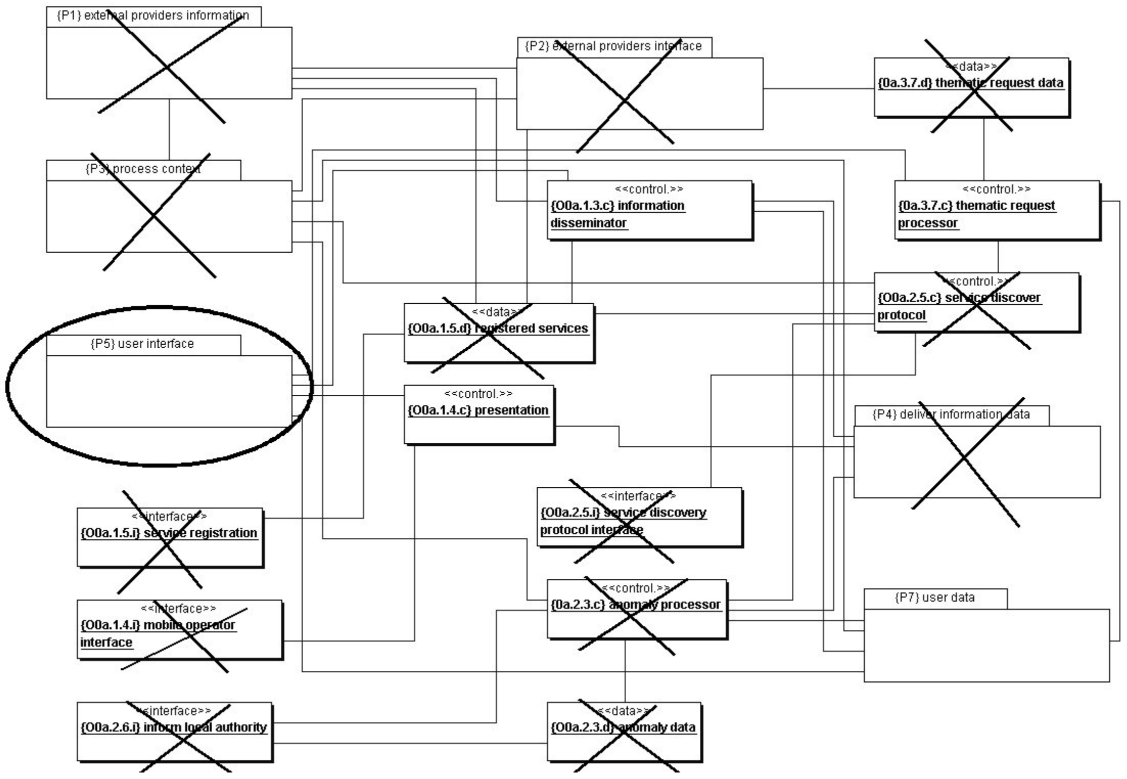

{P1} external providers information
{P2} external providers interface
<<data>> {0a.3.7.d} thematic request data
{P3} process context
<<control.>> {O0a.1.3.c} information disseminator
<<control.>> {0a.3.7.c} thematic request processor
<<control.>> {O0a.2.5.c} service discover protocol
<<data>> {O0a.1.5.d} registered services
{P5} user interface
<<control.>> {O0a.1.4.c} presentation
{P4} deliver information data
<<interface>> {O0a.2.5.i} service discovery protocol interface
<<interface>> {O0a.1.5.i} service registration
<<control.>> {O0a.2.3.c} anomaly processor
{P7} user data
<<interface>> {O0a.1.4.i} mobile operator interface
<<interface>> {O0a.2.6.i} inform local authority
<<data>> {O0a.2.3.d} anomaly data

Figure 26. Execution of the filtering process

Package {P5} can be regarded as the system to be designed. The recursive use of the 4SRS suggests the construction of a new use case diagram which captures the user requirements of the sub-system to refine.

The use case diagram from figure 27 was built to support the refinement of package {P5} to get the raw object diagram of the USE-ME.GOV's AVAccess service. All the actors in this use case diagram are all elements connected to package {P5} in figure 25. The object {Oa0.1.3.c} in the USE-ME.GOV Filtered Object Diagram did not give origin to any actor, because in the package {P5} refinement the functionality associated to this object was not considered. The *user* actor is present in this use case because it was already connected to the use cases that gave origin to the objects inside package {P5} in the process explained in section 4.2. The actors in figure 27 must be seen as external components (sub-system) from the point of view of the *AVAccess service*. The actor {O0a.3.7.c} was specialized into two different actors: *Application System Context Aggregation Service* and *Application System Service Repository*.

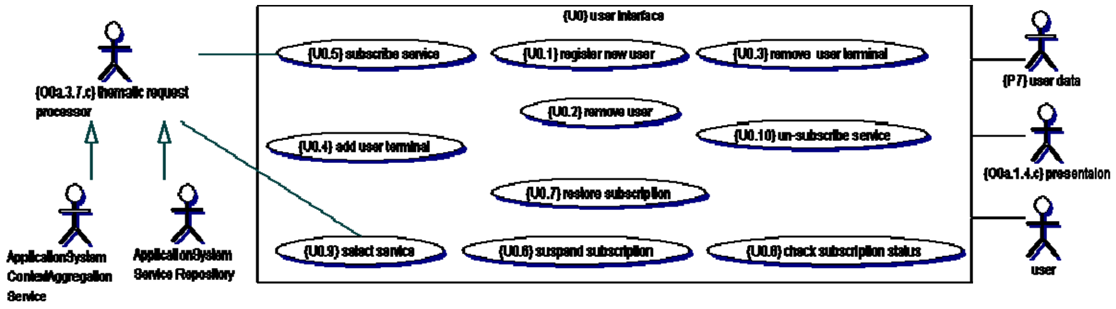

{U0} user interface
{U0.5} subscribe service
{U0.1} register new user
{U0.3} remove user terminal
{U0.2} remove user
{U0.10} un-subscribe service
{U0.4} add user terminal
{U0.7} restore subscription
{U0.9} select service
{U0.6} suspend subscription
{U0.8} check subscription status
{O0a.3.7.c} thematic request processor
ApplicationSystem ContextAggregation Service
ApplicationSystem Service Repository
{P7} user data
{O0a.1.4.c} presentation
user

Figure 27. Use case Diagram: USE-ME.GOV's AVAccess service

Next as examples, the textual descriptions of two use cases are presented.

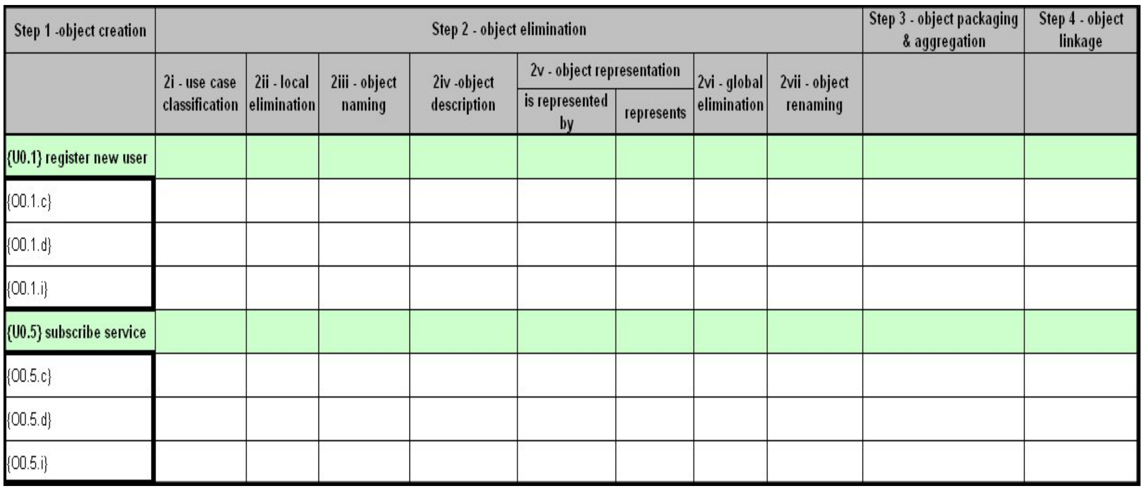
{0.1} Register new user: *the user provides (through communication subsystem) user personal information to the AVAccess service. Its personal information consists of userName, password, and, optionally, user profile information. The AVAccess service parses user personal information and sends it to subsystem User. The AVAccess service sends back the information on success/no success of this operation. The information sent to the user is formatted by the subsystem Presentation. The system must know terminal model information.*

{0.5} Subscribe to service: *the user provides (through communication subsystem) service subscription information to the AVAccess service: the user gets (through communication system) activities defined in AVAccess service; The user gets activityID, activityName and activityDescription; The information on activities sent to the user (through communication subsystem) is formatted by the subsystem Presentation; The user provides (through communication subsystem) the activityID to the AVAccess service; The AVAccess service computes the Service type serviceType that match the chosen activity; The AVAccess service sends back the complete list of Services (information on name, description, cost and ServiceID (URI)) registered in the service repository The user must be authenticated. Service subscription information provided to AVAccess service consists of a list of serviceID. The AVAccess service sends back the information on success/no success of this operation. The information sent to the user (through communication subsystem) is formatted by the subsystem Presentation. The system must know terminal model information.*

4.4. Service Specification

The requirements of the USE-ME.GOV's AVAccess service are now completely defined and the next stage is to apply the 4SRS to this system.

The first step, the object creation, gives origin to 30 objects. In this case study, there is no refinement of the use cases, so this step is applicable to all 10 use cases. Figure 28 presents 4 different rows for use cases {U0.1} and {U0.5}.

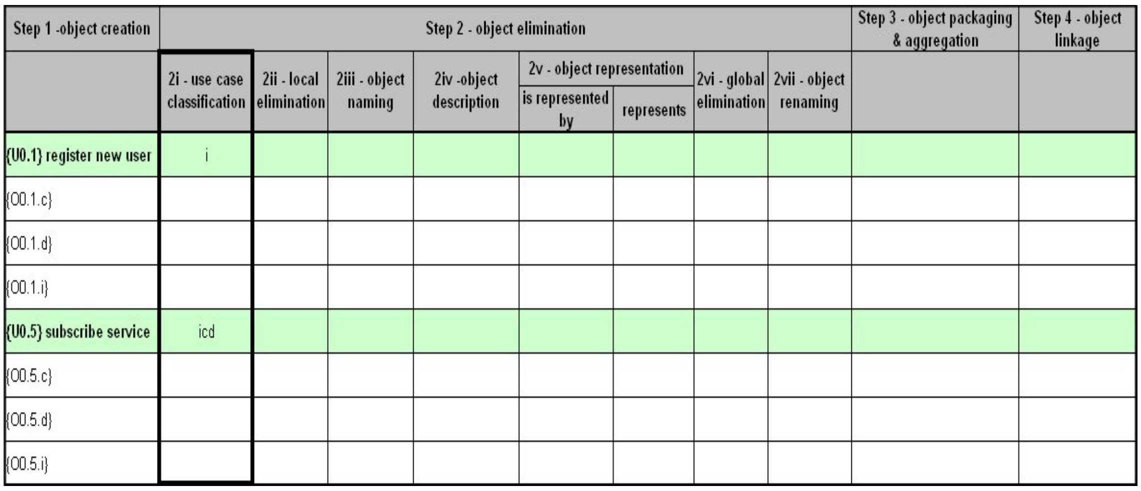


Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation		2vi - global elimination	2vii - object renaming		
					is represented by	represents				
{U0.1} register new user										
{O0.1.c}										
{O0.1.d}										
{O0.1.i}										
{U0.5} subscribe service										
{O0.5.c}										
{O0.5.d}										
{O0.5.i}										

Figure 28. Execution of step 1- object creation for Service specification

The 2nd step of the 4SRS deals with the object elimination. The first micro-step is the use case classification.

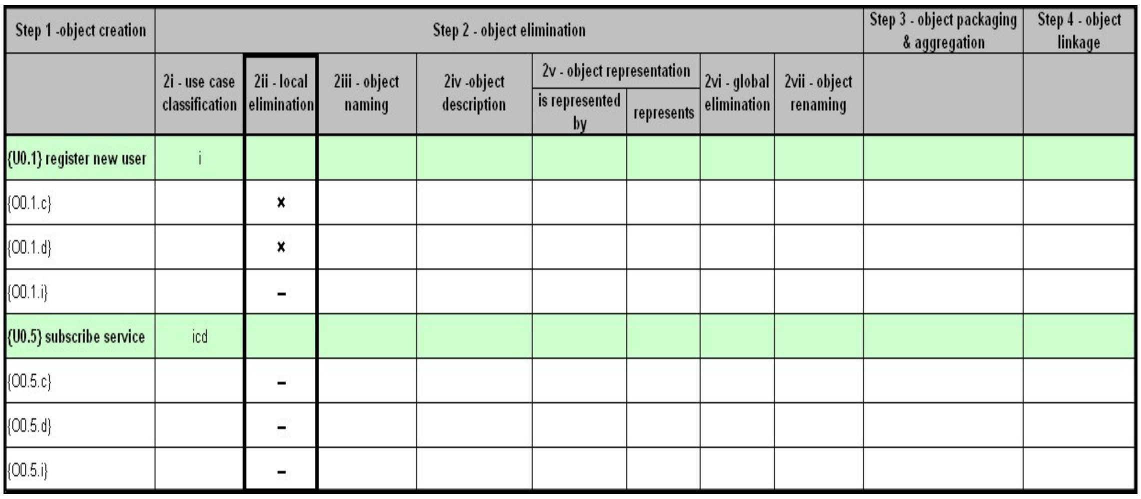
In micro-step 2i, the designer classifies each use case. Use case {U0.1} was classified as type "i" which means that only the interface object will be kept, and {U0.5} was classified as being of type "icd" which means that all objects remain in the model (figure 29).



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}										
{U0.5} subscribe service	icd									
{O0.5.c}										
{O0.5.d}										
{O0.5.i}										

Figure 29. Execution of micro-step 2i – use case classification for Service specification

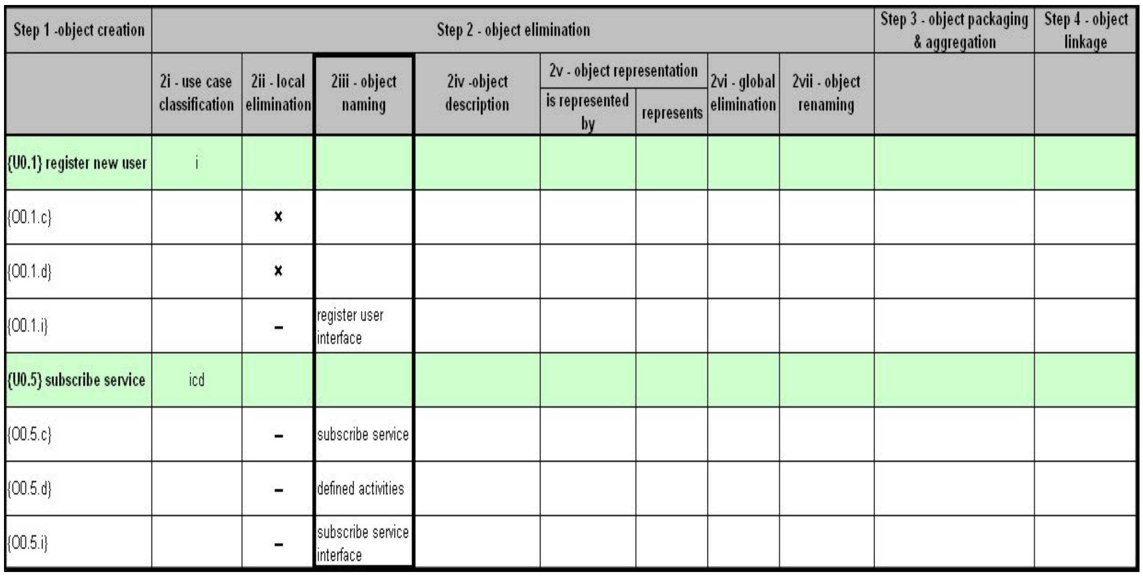
The next step is local elimination. Objects {O0.1.c} and {O0.1.d} do not make sense in the problem domain, as can be seen in the textual descriptions. In figure 30, {U0.1} got two of its originated objects eliminated (only object {O0.1.i} remains in the model), because they do not make sense in the problem domain. {U0.1} is only responsible fot sending 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. The objects originated from the use case {U0.5} were all kept, since they all make sense in the problem domain.



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}		-								
{U0.5} subscribe service	icd									
{O0.5.c}		-								
{O0.5.d}		-								
{O0.5.i}		-								

Figure 30. Execution of micro-step 2ii – local elimination for Service specification

In micro-step 2iii, the objects that have not been eliminated from the previous 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 the main component. Object {O0.1.i} was named *"register user interface"*, and objects {O0.5.c}, {O0.5.d} and {O0.5.i} were named, respectively, *"subscribe service"*, *"defined activities"* and *"subscribe service interface"* (figure 31).

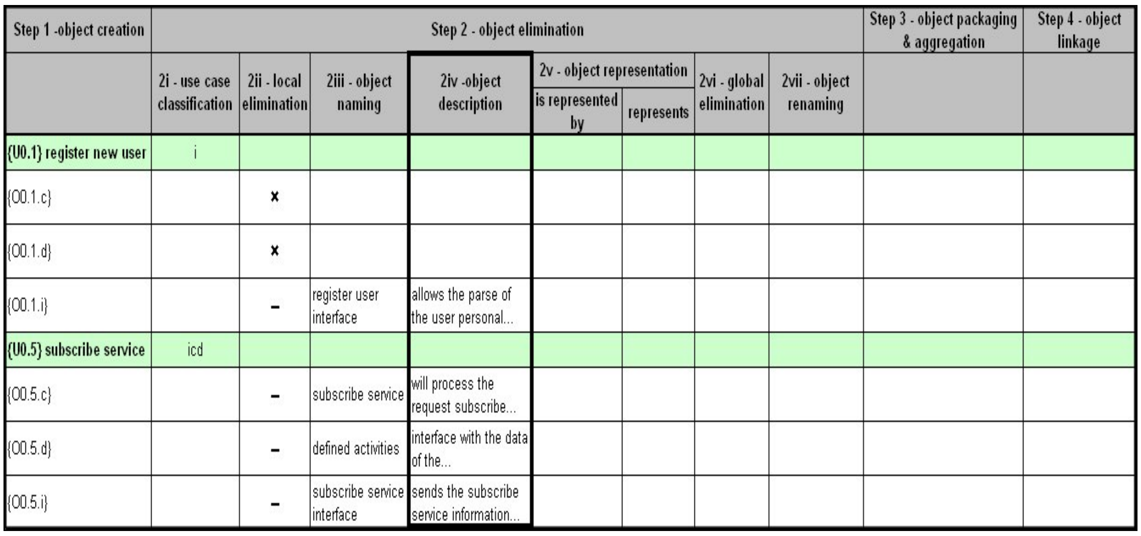


Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local elimination	2iii - object naming	2iv -object description	2v - object representation		2vi - global elimination	2vii - object renaming		
					is represented by	represents				
{U0.1} register new user	i									
{O0.1.c}		×								
{O0.1.d}		×								
{O0.1.i}		–	register user interface							
{U0.5} subscribe service	icd									
{O0.5.c}		–	subscribe service							
{O0.5.d}		–	defined activities							
{O0.5.i}		–	subscribe service interface							

Figure 31. Execution of micro-step 2iii – object naming for Service specification

Each named object resulting from the previous micro-step must be also described, so taht the system requirements they represent become included in the table. The following objects descriptions were obtained and correspond to objects {O0.1.i}, {O0.5.c}, {O0.5.d} and {O0.5.i} (figure32).

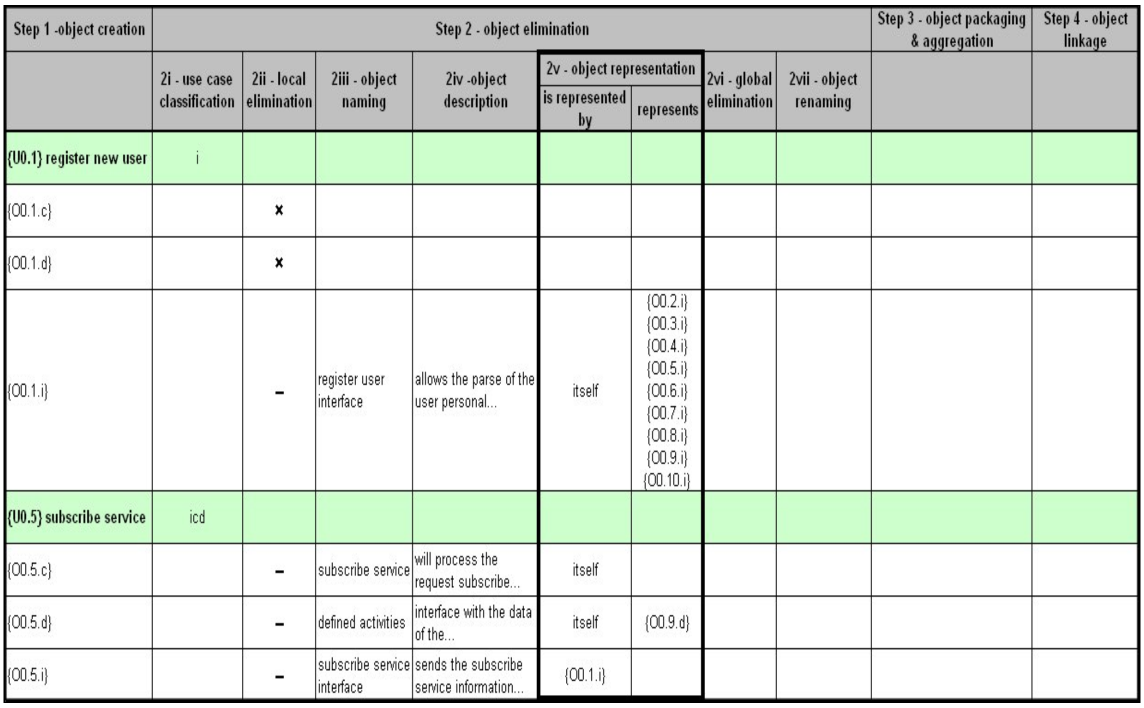
> ***{O0.1.i} register user interface:*** *allows the parse of the user personal information and sends it to the destination subsystem, and sends back the information on success/no success of the request.*
>
> ***{O0.5.c} subscribe service:*** *will process the request Subscribe service. Will request 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*



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...						
{U0.5} subscribe service	icd									
{O0.5.c}		–	subscribe service	will process the request subscribe...						
{O0.5.d}		–	defined activities	interface with the data of the...						
{O0.5.i}		–	subscribe service interface	sends the subscribe service information...						

Figure 32. Execution of micro-step 2iv – object description for Service specification

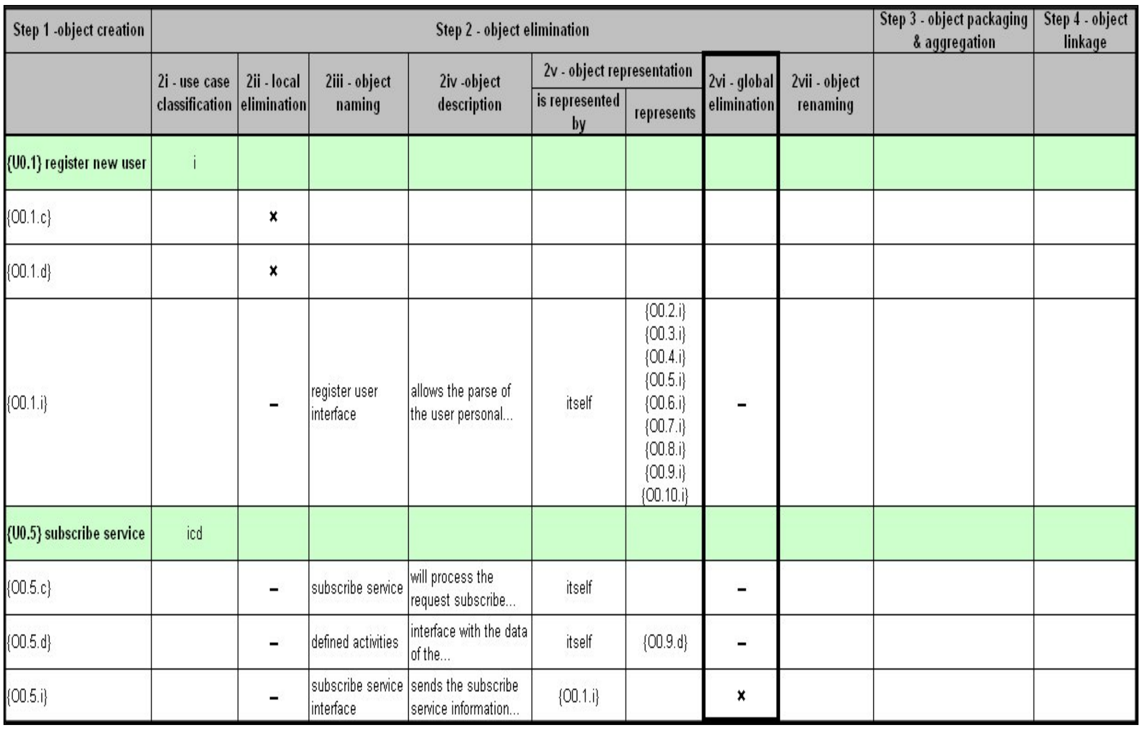
The next micro-step is 2v object representation. This is the most critical micro-step of the 4SRS. In figure 33, {O0.1.i} does not delegate in other object its representation (it will be represented by itself) and it, additionally, represents a considerable list of other objects (each one of these objects refers to {O0.1.i} in its "*is represented by*" cell).



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}				
{U0.5} subscribe service	icd									
{O0.5.c}		–	subscribe service	will process the request subscribe...	itself					
{O0.5.d}		–	defined activities	interface with the data of the...	itself	{O0.9.d}				
{O0.5.i}		–	subscribe service interface	sends the subscribe service information...	{O0.1.i}					

Figure 33. Execution of micro-step 2v – object representation for Service specification

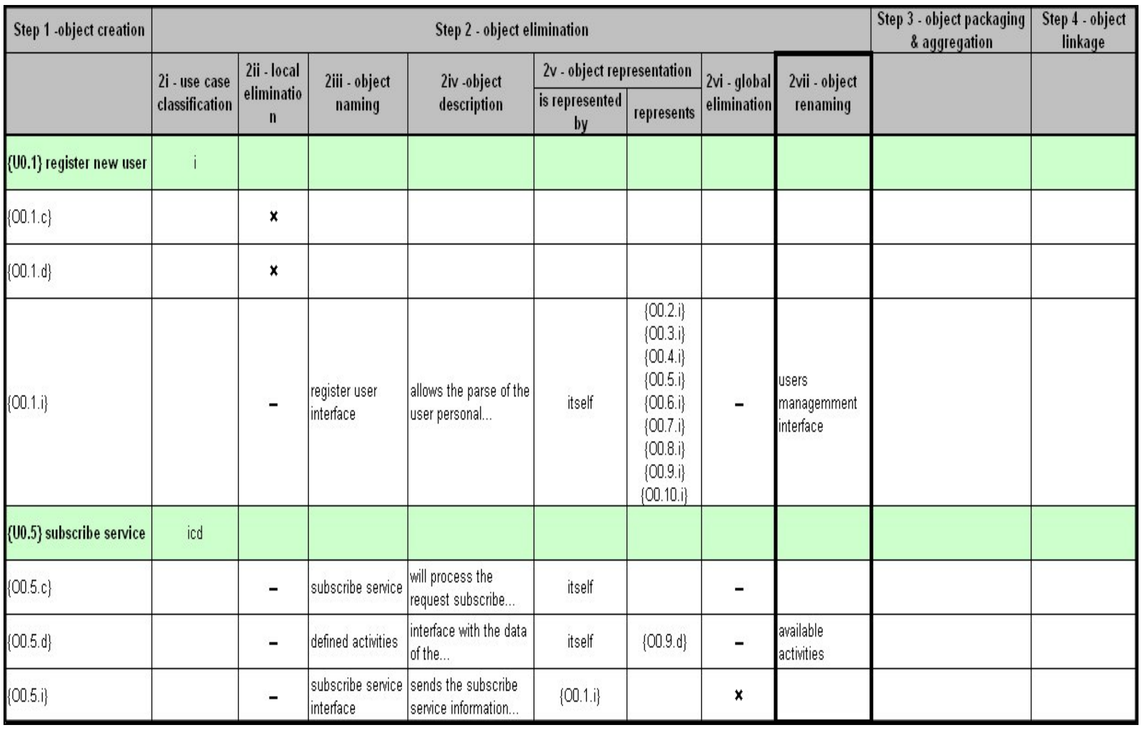
After carrying out the object representation micro-step, the global elimination micro-step is executed. This micro-step is based on the results of the previous one. The objects that are represented by other ones must be eliminated. Figure 34 shows, for instance, the global elimination of {O0.5.i}.



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}	–			
{U0.5} subscribe service	icd									
{O0.5.c}		–	subscribe service	will process the request subscribe...	itself		–			
{O0.5.d}		–	defined activities	interface with the data of the...	itself	{O0.9.d}	–			
{O0.5.i}		–	subscribe service interface	sends the subscribe service information...	{O0.1.i}		×			

Figure 34. Execution of micro-step 2vi – global elimination for Service specification

Next, the objects that have not been eliminated from the previous micro-step and that represent other objects must be renamed. The new name for an object must reflect the system requirements that it represents. In the case study, objects {O0.1.i} and {O0.5.d} are renamed to *"users management interface"*, *"available activities"*, respectively (figure 35).



Step 1 -object creation	Step 2 - object elimination								Step 3 - object packaging & aggregation	Step 4 - object linkage
	2i - use case classification	2ii - local eliminatio n	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}		–	defined activities	interface with the data of the...	itself	{O0.9.d}	–	available activities		
{O0.5.i}		–	subscribe service interface	sends the subscribe service information...	{O0.1.i}		×			

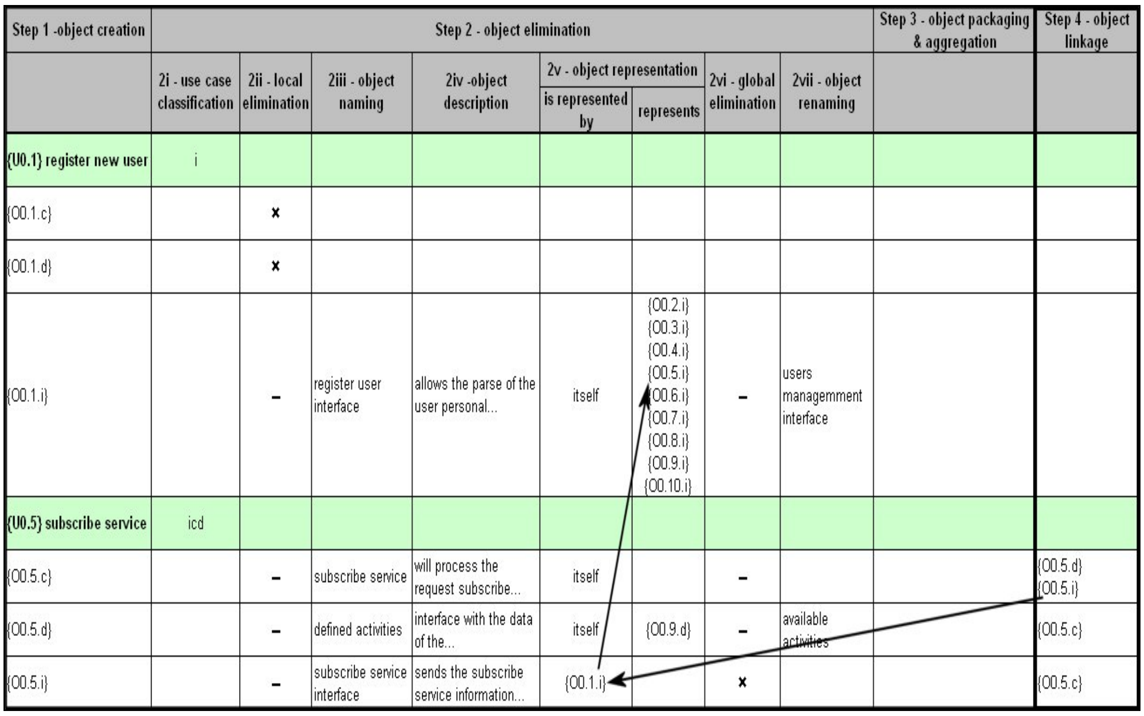
Figure 35. Execution of micro-step 2vii – object renaming for Service specification

In this case study neither aggregations nor packages were used, so the corresponding column in the table remains empty. No package is used because only 4 objects "survive" to the transformation, and there was no evident advantage in treaty some of these objects as a package.

The execution of step 4 is presented in figure 36. For the case study, the links were only derived from the use case classification executed in step 1. The use case {U0.5} was classified as being of type "icd" suggesting the existence of three (or at least two) internal links (relative to the objects generated from the same use case). However, the link between the interface and the data object was not allowed because, as can be seen in the textual description, it is impossible the access to the data by the interface. Additionally, the same transformation made before impose some constrictions to the object connectivity exercise. In particular in micro-step 2v it was decided that {O0.5.i} is represented by {O0.1.i}; and in micro-step 2vi {O0.5.i} was eliminated. Those decisions imply the following links (figure 36):

(1) between {O0.5.c} and {O05.d}, suggested by the "icd" classification;

(2) between {O0.5.c} and {O0.1.i}, caused by the transitivity of the association between {O0.5.c} and {O0.5.i} through the delegation executed by {O0.5.i} in {O0.1.i}.



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}		–	subscribe service interface	sends the subscribe service information...	{O0.1.i}		×			{O0.5.c}

Figure 36. Execution of step 4 – object linkage for Service specification

After executing all the steps, the next action is the translation of the table information to the object diagram. Figure 37 describes the raw object diagram for the *AVAccess* service. This diagram was obtained from the recursive application of the 4SRS technique over the global logical architecture.

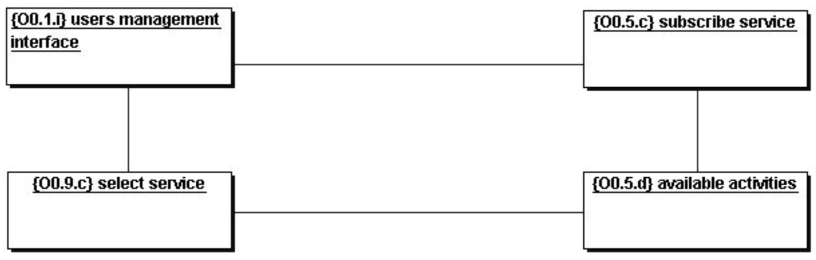

{O0.1.i} users management interface
{O0.5.c} subscribe service
{O0.9.c} select service
{O0.5.d} available activities

Figure 37. USE-ME.GOV's AVAccess service object diagram

4.5. Conclusions

The two cases analysed in this thesis show the application of the 4SRS technique in a real software engineering problem. It was illustrated that this technique is helpful since it ensures the generation of a seamless specification of the architecture requirements.

It has also illustrated that this technique can be used to generate the architecture requirements of a component that was obtained in a previous 4SRS generated object diagram.

In both case studies the defined table, created to store the information and the decisions of all steps, confirm that it is not only useful but also crucial for the success of the project to keep the information updated and a trace of all decisions.

Chapter

5

Conclusions

Summary

The conclusions taken during this thesis are presented in this chapter.
It is also presented the future work that will be done in order to improve the technique presented here.

5.1. Results Analysis

In this thesis an extended version of the 4SRS technique was presented, whose aim is to help the software engineer to seamlessly transform the user requirements of a component-based system into its logical architecture of the system that captures de user requirements.

This technique assumes that the user requirements are modelled by UML use case diagrams (and the corresponding descriptions of the use cases) and the first architecture of the system must be modelled by UML object diagrams.

Therefore, the 4SRS technique is a model-based transformation that is based on the mapping of UML use cases into UML object diagrams. However other diagrams, like UML sequence, activity and state diagrams can be also used, in order to help the software engineer to better take his decisions. In this thesis, the usage of other models is not considered.

To give support to the execution of the 4SRS it was defined a table to store all the decisions taken during the transformation task. When apply the 4SRS to a use requirement model the table will be filled with the information and decision taken at each step. The information of the table after the execution of all steps gives the necessary information to generate the object diagram of the system that represents its structure.

The 4SRS can also be used iteratively, i.e., we can apply the 4SRS to a component of the system's architecture, previously obtained by the usage of the 4SRS. The recursively usage of that technique proposes the construction of a new use case diagram that captures the user requirements of a given sub-system. Then, the 4SRS will be applied to the new use case model to obtain the architectural model of the sub-system.

The 4SRS technique highlights the fact that the continuity of models is an important issue. It also shows the importance of having a well defined process to transform requirements models.

The analysis of a real and complex case study allowed us to demonstrate that the 4SRS technique is useful for obtaining the architecture of a pervasive application from its requirements model.

5.2. Future Work

As future work, it is planned to define some rules to allow the direct mapping of use cases into objects without the need to unfold the use cases hierarchy. It would allow the maintenance of the hierarchical structure in the object model analogous to the hierarchical structure existent in the use case model refinement tree.

The 4SRS will be extended in order to enable the transformation of objects diagrams into class diagrams.

It is also planned the definition of a mechanism to automate the 4SRS technique based on the defined tabular transformation and to support it within a IDE tool like GME (The Generic Modeling Environment http://www.isis.vanderbilt.edu/Projects/gme/) or Eclipse (http://www.eclipse.org/).

Bibliography

[1] Catalog of OMG Modeling and Metadata Specifications. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML.

[2] Jamda: The Java Model Driven Architecture 0.2. May 2003 http://sourceforge.net/projects/jamda/.

[3] J. Ø. Aagedal and I. Solheim. New Roles in Model-Driven Development. in Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations, September, 2004. Canterbury, England.

[4] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. in In the Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), April, 2000. The Hague, The Netherlands.

[5] M. Alanen, J. Lilius, I. Porres and D. Truscan. Model Driven Engineering: A Position Paper. in 1st Int. Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'04), June, 2004. Hamilton, Canada, p:25-29.

[6] R.-J. Back, L. Petre and I. P. Paltor. Analysing UML Use Cases as Contracts. in 2nd Int Conf. on the Unified Modeling Language (UML'99), Oct, 1999, p:518-533, Springer.

[7] G. Banavar and A. Bernstein, Software Infrastructure and Design Challenges for Ubiquitous Computing Applications. Communications of the ACM. 45(12): p. 92-96, 2002.

[8] L. B. Becker, C. E. Pereira, O. P. Dias, I. M. Teixeira and J. P. Teixeira. MOSYS: A Methodology for Automatic Object Identification from System Specification. in 3rd IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2000), March, 2000, p:198-201, IEEE CS Press.

[9] U.-M. G. Consortium. Usability-Driven Open Platform for Mobile Government. in, 2004

[10] M. Elkoutbi and R. K. Keller. User Interface Prototyping based on UML Scenarios and High-level Petri Nets. in In Application and Theory of Petri Nets 2000 (Proc. of 21st Intl. Conf. on ATPN), June, 2000. Aarhus, Denmark, p:166-186, Springer.

[11] J. M. Fernandes and R. J. Machado. From Use Cases to Objects: An Industrial Information Systems Case Study Analysis. in 7th Int. Conf. on Object-Oriented Information Systems (OOIS '01), Aug., 2001, p:319-28, Springer-Verlag.

[12] J. M. Fernandes and R. J. Machado. System-Level Object-Orientation in the Specification and Validation of Embedded Systems. in 14th Symp. on Integrated Circuits and System Design (SBCCI '01), September, 2001, p: 8-13, IEEE CS Press.

[13] J. M. Fernandes, R. J. Machado and H. D. Santos. Modeling Industrial Embedded Systems with UML. in 8th ACM/IEEE/IFIP Int. Workshop on Hardware/Software Codesign (CODES'2000), May, 2000. San Diego, CA, USA, p:18-22, ACM Press.

[14] Z. Hu and S. Shatz. Mapping UML Diagrams to a Petri Net Notation to Exploit Tool Support for System Simulation. in Software Engineering and Knowledge Engineering (SEKE'04), 2004, p:213-219.

[15] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process. 1999: Addison-Wesley.

[16] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, Object-Oriented Software Engineering: A Use Case Driven Approach. 1992: Addison-Wesley.

[17] M. T. Kimour and D. Meslati, Deriving objects from use cases in real-time embedded systems. Information and Software Technology. 47(8): p. 533-541, 2005.

[18] A. Kleppe and J. Warmer. Do MDA Transformations Preserve Meaning? An investigation into preserving semantics. in Metamodelling for MDA First International Workshop, November, 2003. York, UK.

[19] A. Kleppe, J. Warmer and W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise. 2003: Addison-Wesley.

[20] A. v. Lamsweerde and L. Willemet, Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering. 24(12): p. 1089-1114, 1998.

[21] Y. Liang, From Use Cases to Classes: a Way of Building Object Model with UML. Information and Software Technology. 45: p. 83-93, 2003.

[22] J. Lilius, D. Truscan and J. M. Fernandes, Integration of DFDs into a UML-based Model-Driven Engineering Approach. Software and Systems Modeling (SoSyM), 2005.

[23] OMG. MDA Guide Version 1.0.1, June, 2003.

[24] OMG. XML Metadata Interchange Specification 1.2, OMG Document: formal/02-01-01. in, January 2002

[25] R. K. Runde and K. Stolen, What is Model Driven Architecture? Research Report 304 ISBN 82-7368-256-0 ISNN 0806-3036, March, 2003

[26] M. Saeki and H. Kaiya. Transformation Based Approach for Weaving Use Case Models in Aspect-Oriented Requirements Analysis. in 4th AOSD Modeling With UML Workshop, 2003.

[27] M. Satyanarayanan, Pervasive computing: Vision and challenges. IEEE Personal Communications. 8(4): p. 10-17, 2001.

[28] B. N. Schilit, N. Adams and R. Want. Context-Aware Computing Applications. in Workshop on Mobile Computing Systems and Applications, 1994. Santa Cruz, CA, U.S, IEEE Computer Society.

[29] S. Schönberger, R. K. Keller and I. Khriss, Algorithmic Support for Model Transformation in Object-Oriented Software Development. Concurrency and Computation: Practice and Experience. 13(5): p. 351-383, 2001.

[30] S. Sendall, G. Perrouin, N. Guelfi and O. Biberstein. Supporting Model-to-Model Transformations: the VMT approach. in Workshop Model Driven Architecture: Foundations and Applications, Technical Report TR-CTIT-03-27. Univ. of Twente, 2003

[31] A. G. Sutcliffe, N. A. M. Maiden, S. Minocha and D. Manuel, Supporting Scenario-Based Requirements Engineering. IEEE Transactions on Software Engineering. 24(12): p. 1072-1088, 1998.

[32] D. Truscan, J. M. Fernandes and J. Lilius. Tool Support for DFD-UML Model-based Transformations. in 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004), May, 2004. Brno, Czech Republic, p:388-397, EEE CS Press.

[33] M. Weiser, The Computer for the 21st Century. Scientific American: p. 66-75, 1991.