
Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Dissertação de Mestrado em Engenharia Informática

2º Semestre, 2007/2008

# Aspect-Oriented Domain Analysis

António Pedro Lopes Borba Rodrigues

Nº 26152

Orientador

Prof. Doutor João Araújo Júnior

Lisboa

31 de Julho de 2008

Nº do aluno: 26152
Nome: António Pedro Lopes Borba Rodrigues

Título da Dissertação: Aspect-Oriented Domain Analysis

Palavras-Chave:

- Desenvolvimento de Software Orientado a Aspectos
- Engenharia de Requisitos Orientada a Aspectos
- Análise de Domínio
- Modelação de *Features*
- Pontos de vista

Keywords:

- Aspects-Oriented Software Development
- Aspect-Oriented Requirements Enginering
- Domain Analysis
- Feature Modelling
- Viewpoints

## Acknowledgments

I would like to thank many people for ending this chapter in my life:

Firstly, I would like to thank my parents Luís and Rosa for putting up with me these 25 years of my life, for supporting me in the best and worst moments of my life… and most important of all, for sticking up together. To my family in general and of course to Adelaide, who not being part of the family, will always be considered as part of it.

Secondly, to my supervisor, Dr. João Araújo Júnior, for accepting me under his wing, guiding me through these last two semesters, who taught me, and showed the right path in order to accomplish the ultimate goal in this chapter: The Masters Degree. To Professora Doutora Ana Moreira and Professor Doutor Artur Miguel Dias for approving the first part of this thesis and therefore, allowing me to continue pursuing this quest. All my teachers in Colégio Moderno, in Cambridge School and finally in Faculdade de Ciências e Tecnologia… without them, it wasn't possible to reach this far.

Thirdly, all the true friends made through life, despite not seeing some in years: from Colégio Moderno: Melo, Vasco and Diogo; and in FCT… a special group I'd like to thank - the "old school" FCK, without any particular order: Alex, Furia, Flip, Dag, Ademar, Fxt, Pinha, Sansão, André Gil, Ivo, Henrik, Miguel and BobCat. In addition to those… the "new school" FCK: Dudu, Marco, Oz and Cátia. Inside these groups, it wasn't fair if I didn't thank in particular to André Gil, Dudu and Oz for the helping hand every time I needed.

I would like to thank Ana Varelas for the support in the last 7 years, and the Alegria's for the advice throughout my whole life.

Last, but not least: the "Universe"

To all the above: I will never forget you. It's needless to say, but you all know… if you ever need me… just call my name.

Thank you all.

# Resumo

A Análise de Domínio (AD) consiste na análise de propriedades, conceitos e soluções para um dado domínio de aplicação. Baseado nessa informação, são tomadas decisões sobre o desenvolvimento de software para aplicações futuras dentro desse domínio. Em AD, a modelação de *features* é muito utilizada para descrever os requisitos comuns e variáveis. Contudo, mostram uma perspectiva limitada do domínio.

Entretanto, as abordagens de requisitos podem ser integradas para especificar os requisitos de domínio. Entre elas, temos as abordagens orientadas a pontos de vista que se destacam pela simplicidade, e eficácia em organizar os requisitos. Contudo, nenhuma delas trata da modularização dos assuntos transversais. Um assunto transversal pode estar espalhado em vários documentos de requisitos. Neste trabalho vamos utilizar e adaptar uma abordagem orientada a pontos de vista estendida com aspectos.

A análise de domínios orientada a aspectos é uma área de crescente interesse, pois endereça o problema de especificar propriedades transversais ao nível de análise de domínios. Esta área tem o intuito de obter uma melhor reutilização a este nível de abstracção através das vantagens da orientação a aspectos. O objectivo deste trabalho é propor uma abordagem que estenda a AD com aspectos usando também modelação de *features* e pontos de vista.

## Abstract

Domain analysis (DA) consists of analyzing properties, concepts and solutions for a given domain of application. Based on that information, decisions are made concerning the software development for future application within that domain. In DA, feature modeling is used to describe common and variable requirements for software systems. Nevertheless, they show a limited view of the domain.

In the mean time, requirement approaches can be integrated to specify the domain requirements. Among them, we have viewpoint oriented approaches that stand out by their simplicity, and efficiency organizing requirements. However, none of them deals with modularization of crosscutting subjects. A crosscutting subject can be spread out in several requirement documents. In this work we will use a viewpoint oriented approach to describe the domain requirements extended with aspects.

Aspect-oriented domain analysis (AODA) is a growing area of interest as it addresses the problem of specifying crosscutting properties at the domain analysis level. The goal of this area is to obtain a better reuse at this abstraction level through the advantages of aspect orientation. The aim of this work is to propose an approach that extends domain analysis with aspects also using feature modeling and viewpoints.

## Acronyms

Below we have a list of acronyms used in this thesis.

| ACRONYM | EXPANSION |
|---|---|
| AOCRE | Aspect-Oriented Components Requirements Engineering |
| AODA | Aspect-Oriented Domain Analysis |
| AORE | Aspect-Oriented Requirement Engineering |
| AOSD | Aspect-Oriented Software Development |
| ARCaDe | Aspectual Requirements Composition and Decision Support |
| CODA | Context-Oriented Domain Analysis |
| DA | Domain Analysis |
| FODA | Feature-Oriented Domain Analysis |
| OODA | Object-Oriented Domain Analysis |
| PREview | Process and Requirements Engineering viewpoints |
| RE | Requirement Engineering |
| SEI | Software Engineer Institute |
| SPL | Software Product Line |
| VORD | Viewpoint-Oriented Requirement Definition |
| XML | Extensible Mark-up Language |

# Table of Contents

# List of Figures

# List of Tables

# List of Listings

# Chapter 1
# Introduction

Software reusability is the key to significant productivity gains and its success is granted mostly by the use of Domain Analysis (DA). DA was firstly introduced in the 80's decade as an activity within domain engineering which is the process where information used in developing systems is identified, captured and organized with the objective of making it reusable when creating new systems. DA also focuses on giving support to systematic and large-scale reuse by identifying and capturing common and variable features of systems within a certain domain in order to improve the efficiency of development and maintenance of those same systems.

DA is a means to build reusable infrastructures to support the specification and implementation of restricted classes of applications. Some domain requirements can be referenced as crosscutting as they "cut across" other domain requirements, and that can be a problem when changes in the domain requirements occur, causing problems to system evolution.

To overcome this issue, some approaches have been developed within the software engineering community, like Aspect Oriented Domain Analysis (AODA) [1], Object-Oriented Domain Analysis (OODA) [2], Context-Oriented Domain Analysis (CODA) [3] and Feature-Oriented Domain Analysis (FODA) [4].

In this work we will propose a new approach which combines AODA with feature modeling and viewpoints.

## 1.1 Context

### 1.1.1 Requirements Engineering

Requirements Engineering (RE) is a relatively new discipline which has been created to cover all the activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system [5]. The use of the term "engineering" implies that systematic and repeatable techniques should be used to ensure that system requirements are complete and consistent.

Requirements are descriptions of how a system should behave [6]. They are also knowledge of application domain and constraints on operation of a system. Requirements management is the process of managing changes to the requirements, which is fundamental as requirements tend to change in order to reflect the changing needs of stakeholders - the people or organizations who will be affected by the system or have some interest in it. They have either direct or indirect influence on system requirements and can also change due to change in environment, business plans and laws.

Also, RE is a process of discovering the needs of stakeholders and documenting them for analysis, communication and implementation. Requirements engineering includes a set of activities such as [5, 6]:

- *Requirements Elicitation and Analysis* – this is the process of discovering the requirements of an intended system (Elicitation is to interpret, analyze, model and validate information from stakeholders). The goal of these activities is to find out about the application domain, what services the system should provide, the required performance of the system and its constraints among others. Interviews, questionnaires, surveys, process models are used as good source of information to capture requirements. Requirements analysis is needed to solve problems and reach an agreement on changes to requirements. As each stakeholder has his/her own requirements, the requirements which cover main functions should have a bigger

priority. Setting priorities is a way to balance desired project scope against restrictions of schedule, budget, staff and quality goals. Requirements analysis can be done concurrently with requirements elicitation process. In this particular sub-phase analysts read the requirements, problems are highlighted, reviewed and conflicts among stakeholders should be solved.

- *Requirements Documentation* – Different diagrams are used for depicting requirements at various levels of detail such as: Entity Relationship Diagrams (ERD) to model the relationships between the system's entities; Data Flow Diagrams (DFD) to model how data is processed by a system in terms of inputs and outputs; Interaction Diagrams to describe scenarios, and Use-Case Diagrams to model the main services of a system, among others;
- *Requirements Validation* – This is the process of showing that the requirements elicited actually define the pretended system. It overlaps requirements analysis as it is concerned with finding problems with the requirements. Validation is important because errors in documentation can lead to higher costs when found during development. While validating, some important checks should be verified as: validation checks, consistency checks, completeness checks, realism checks and verifiability. To perform validation, it is used several techniques such as Requirements reviews, prototyping and test-case generation;
- *Requirements Management* - Is the process of understanding and controlling changes made to system requirements. It is needed to track individual requirements and maintain connections between dependent requirements so that one can be aware of requirements changes. This process should start as soon as a draft version of the requirements documentation is available, but it is good practice to plan how to manage requirements change during the requirements elicitation process.

Some people suggest that a requirement should always be a statement of what a system should do rather than a statement of how it should do it. This seems simple, but it is not what happens in practice.

1. The readers of a document are often practical engineers who can relate to implementation descriptions much better than they can understand very abstract problem statements therefore, requirements have to be written in a way understandable to the likely readers of the document;

2. In almost all cases, the system being specified is only one of several systems in an environment. To be compatible with its environment, and to conform to standards and organizational concerns, it might be necessary to specify implementation policies which constrain the options of the system designers.

Invariably, requirements contain a mixture of problem information, statements of system behavior and properties, design and manufacturing constraints.

Requirements can be categorized into two main types: functional and non-functional. A functional requirement describes a function that a software system should provide, how it should react to specific inputs, how it should behave in specific situations and more importantly, it should describe what the system should not do. They are constrained by the non-functional requirements (such as cost, security or reliability).

Non-functional requirements are requirements that constrain the functional ones. They specify criteria that can be used to establish rules for the operations in a system, rather than behaviors. Non-functional requirements are often called the "-ilities", "constraints", or "quality goals".

Domain requirements are derived from the application domain of the system rather than from the specific needs of system users [7]. They may include references to domain concepts and specific domain terminology. They can be new functional, non-functional requirements or determine how particular computations must be developed. These requirements are very specific, making it hard to understand how they are related to other system requirements. They also can be developed to be core assets in product lines for reuse [8].

Requirements approaches organize requirements according to, for example, goals [9], use cases [10], and viewpoints [11]. In this dissertation we will focus on viewpoints, due to its simplicity and efficiency to organize stakeholder's requirements.

### 1.1.2 Domain Analysis

According to Software Engineering Institute [12], Domain Analysis is "*the process of identifying, collecting, organizing, and representing the relevant information in a domain,*

*based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain".*

In addition to what was said in the introduction, the goals of domain analysis are:

- Establish a specification language that is sufficient for describing all (or most) systems within an application domain.
- Simplicity, i.e. the domain language must be able to express all basic concepts used to describe systems in the domain, therefore it must be as simple as possible;
- Identify a set of architectures and components that can be used to assemble implementations for every specification that can be formulated in the language;
- Define a mapping to match specifications to relevant architectures and components unambiguously and define those architectures and components in such a way that they can be adapted using pre-defined, minimum cost, structured mechanisms.

## 1.2 Motivation: The Problem of Crosscutting Concerns

Requirements are defined during the early stages of a system development as a specification of what should be implemented [5]. They are descriptions of how the system should behave, or of a system property of attribute. Independently of its purpose, but due to the decomposition criteria used to structure the requirements of large systems, it is common to observe requirements that are scattered and tangled throughout the requirements documents. They are called crosscutting requirements. They may be constrained on the development process of the system.

**What are crosscutting concerns?**

Crosscutting requirements are a special kind of requirement which influence the way how other requirements are satisfied by design or code artifacts.

Furthermore, the expression crosscutting influence is used as a synonym for the relationship between two requirements which is established by one crosscutting the other [13]. Crosscutting influences are more subtle if clear references do not exist, and therefore, they cannot be easily concluded from reading a crosscutting requirement that it is crosscutting or, in case it is known that it is, on which other requirements it has influence. Thus even if we do

not find obvious tangling on the requirements representation level there may be, and in fact there does exist crosscutting influences beyond the representation level, that is why they are called candidate. These are dangerous since they may be overlooked in subsequent development stages.

**Why identifying crosscutting concerns?**

Requirements tend to change often [14]. It is especially difficult to adapt complex systems to crosscutting requirements changes as each modification must consider which data or which components deployed on computers in contexts are affected by which requirements.

The best way to deal with crosscutting requirements is to separate them from other requirements and model them independently [15]. This modularization avoids tangled representations in the requirements document, facilitating requirements evolution. On the other hand, if no attention is paid to how the crosscutting requirements interact with other requirements, there is a danger of the nature of these interactions will only become clear during later stages of software development. If problems with these interactions are only discovered at this point, they will, in general, be costly to rectify.

If some of the relationships among requirements (such as crosscutting influences) are obscure, developers run the risk of forgetting about them. At best they detect these influences in later development phases and are able to react accordingly. At worst some influences remain hidden until after delivery of the software system and are then indirectly discovered by users because some requirements are not totally fulfilled by the software. In both cases increased costs will be the result of requirements deficiencies (the later the detection, the higher the costs). The second case may in addition give rise to other problems such as loss of good faith in the developers or even the disastrous failure of critical system components.

If done early enough, identification and documentation of crosscutting requirements and their impact on other requirements (which may, of course, be crosscutting themselves) brings about another important advantage: not to overlook them in subsequent phases. Provided that the crosscutting influences on a certain requirement are documented, it is easily possible to take them into account when design or code artifacts are derived from it or some corresponding maintenance task is performed.

**When to identify crosscutting requirements**

Identification and documentation of crosscutting requirements should be done as early as possible [13] since it improves traceability between requirements and downstream artifacts. It also eases the assessment of change impact, enables the application of aspect orientation from the beginning of the software lifecycle, supports requirements evolution and prevents easy mistakes of crosscutting influences during development and maintenance activities.

Requirements engineers must be prepared to deal with the crosscutting nature of some requirements. When requirements cut across other requirements we may end up with tangled representations of those requirements throughout the requirements document. Consequently, the reaction to change is more difficult, as the impact of the change is more complicated to handle. Hence, it is important to contemplate crosscutting requirements early in the software lifecycle. However, there are several stages of the development and maintenance processes where identification and documentation of crosscutting requirements and influences are possible:

- *During requirements modeling:* After requirements have been elicited they should be modeled. It would be desirable to identify and model crosscutting requirements and influences already at this stage;
- *While writing the requirements specification:* Writing a specification normally follows requirements modeling. However, it is often the case that a specification is written directly after requirements have been elicited thus bypassing the modeling stage;
- *After writing the requirements specification:* Although it is best to identify and document crosscutting requirements and influences when they arise;
- *During downstream activities:* The crosscutting nature of some requirements and their influences may and will also be detected during activities later in development or maintenance and should then be documented.

Domain requirements can also be crosscutting, so they should be modeled as aspectual domain requirements. However, not many approaches have been developed so far that integrates aspects into DA.

## 1.3 Objectives

This work focuses on Domain Analysis with Aspects. Most of DA approaches mainly rely on feature modeling, but feature models just give a partial perspective of the domain. Therefore, their integration to other modeling techniques (such as viewpoints and use cases) should give a broader vision of the domain. In this dissertation we will use viewpoints to describe domain requirements. Moreover, crosscutting concerns are not specifically addressed in most of them, so this approach tries to fulfill this gap.

In summary, the objectives of this dissertation consist of presenting a novel approach based on viewpoints to specify and structure domain requirements having into account the aspectual domain requirements. Also, it integrates feature modeling and an aspectual viewpoint model. In order to achieve this, it is used the PREview method [5, 16] extended with aspects to describe domain requirements through domain viewpoints and integrated with a feature model.

## 1.4 Organization

This dissertation has five chapters. Chapter 1 is this introduction, where the topic of this dissertation is introduced, the context where it is inserted is described, the motivation for this thesis is mentioned, the problem is defined and the goals are mentioned.

The other four chapters are briefly presented below.

- Chapter 2. *Background*. In this chapter it is presented some related work on viewpoints, aspect-oriented requirement engineering, domain analysis and feature modeling.
- Chapter 3. *A Viewpoint and Aspect-Oriented Domain Analysis approach*. Here our approach is presented and applied to a case study both in domain engineering and application engineering.
- Chapter 4. *Case Study and comparison with other approaches*. In this chapter, our approach is applied to a more complex case study – Health Watcher – and is it compared and contrasted with the approaches described in Chapter 2.

- Chapter 5. *Conclusions*. This chapter states the conclusions of this thesis as well its limitations and suggests directions for future work. It finishes with some final remarks.

# Chapter 2
# Background

This chapter presents the background related to this work. It is divided into three main topics: Viewpoints, Aspect-Oriented Requirements Engineering and finally Domain analysis and feature modeling. Several approaches are presented in the topics as well as some concepts to better understand the purpose of the methods used.

## 2.1 Viewpoints

Viewpoint identification should be one of the first steps of a requirements analysis because it helps the elicitation and management of requirements. The term “viewpoint” is a synonymous with a perspective on a system. Each viewpoint represents the perspective of a particular stakeholder on a given problem where each one imposes requirements on the solution to the problem.

Viewpoints focus primarily on the early elicitation stages of the requirements process and using them can help structure the elicitation process. Viewpoints can also help prioritize and manage requirements. Another way of thinking of viewpoints is that they focus the analyst’s attention on the parts of the problem which affect stakeholders, and project the discovered requirements onto the system to be developed.

**What is a viewpoint?**

A viewpoint represents an encapsulation of partial information about a system's requirements from a particular perspective [5]. The information is partial because it is restricted to one perspective on the system and therefore omits other perspectives' requirements. A viewpoint should not only contain requirements. It is good practice to associate additional information with requirements to help assessing changing requirements and with tracing. A viewpoint provides a convenient structuring mechanism with which associates requirements with this other information, therefore it may contain a set of requirements as well as a definition of the viewpoint's perspectives and a list of the sources from which the requirements were elicited.
Viewpoints are not only concerned with human perspectives. They can be associated with the system's stakeholders; the system's operating environment or the system's domain.

A viewpoint is defined by its focus and one should not exclude any type of viewpoint which an organization may find useful in its requirements engineering process. They generally fall into one of three classes [17]:

- *Interactor viewpoints:* These are the viewpoint of something (human or machine) which interacts directly with the system being specified;
- *Indirect stakeholder viewpoints:* These are the viewpoint of an entity (human, role or organization) which has an interest (stake) in the problem but who does not interact directly with the system;
- *Domain viewpoints:* These represent a set of related characteristics of the domain which cannot be identified with a particular stake but which nevertheless impose requirements which are implicit in the domain.

**Why viewpoints?**

A viewpoint-based approach to requirements analysis recognizes that all information about the system requirements cannot be discovered by considering the system from a single perspective.
The principal advantages offered by viewpoints are:

- Requirements are likely to be more complete than if viewpoints are not explicitly identified. In the latter case, important requirements may be easily overlooked because their viewpoints were never recognized.

- A separation of concerns is provided which permits the development of a set of "partial specifications" in isolation from other viewpoints. This avoids having to confront conflicts with other viewpoint requirements during elicitation.
- Traceability is enhanced by the explicit association of requirements with the viewpoints from which they are derived.

Viewpoints are complementary to a number of other requirements engineering practices. In particular they can help inform the development of system models, such object-oriented or functional models. Similarly, system models can be developed for each viewpoint to help clarify their requirements.

**What is the information provided by viewpoints?**

As stated before, viewpoints are entities that encapsulate some but not all information about system requirements. This information may be taken from various places, like, analysis of existing systems, discussions with system stakeholders just to name a few. In order to complete requirements, all the requirements derived from different viewpoints are integrated.
Each viewpoint is specified by [5]:

- Viewpoint name
- The stakeholders addressed by the viewpoint
- The stakeholder concerns to be addressed by the viewpoint
- The viewpoint language, modeling techniques, or analytical methods used
- The source, if any, of the viewpoint (e.g., author, literature citation)

A viewpoint may also include:

- Any consistency or completeness checks associated with the underlying method to be applied to models within the view;
- Any evaluation or analysis techniques to be applied to models within the view;
- Any heuristics, patterns, or other guidelines which aid in the synthesis of an associated view or its models.

Viewpoint identification should be one of the first steps of a requirements analysis because it helps the elicitation and management of requirements. The term “viewpoint” is a synonymous with a perspective on a system. Each viewpoint represents the perspective of a particular stakeholder on a given problem where each one imposes requirements on the solution to the problem.

Viewpoints focus primarily on the early elicitation stages of the requirements process and using them can help structure the elicitation process. Viewpoints can also help prioritize and manage requirements. Another way of thinking of viewpoints is that they focus the analyst’s attention on the parts of the problem which affect stakeholders, and project the discovered requirements onto the system to be developed.

### 2.1.1 PREview

PREview (Process and Requirements Engineering viewpoints [5, 16]) proposes the use of a process-based around requirements viewpoints to guide the identification, analysis and management of software requirements, this process is depicted in figure 2.1. It recognizes that domain knowledge may be too diverse to guarantee that all requirements can be elicited and that many requirements may be implicit in the domain itself.

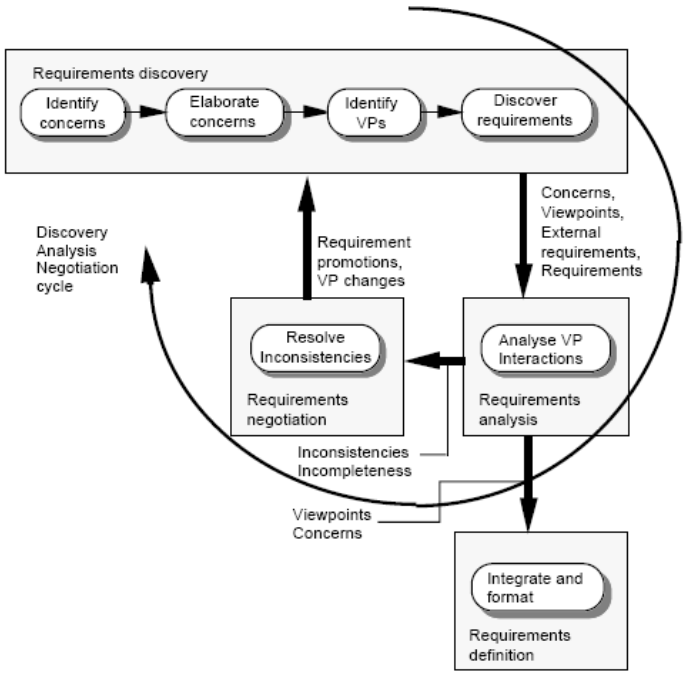


Figure 2.1: The PREview process [16]

In PREview, each viewpoint is an entity that has some, but not all information about the system's requirements. This information could be taken from a process analysis, from existing systems, from stakeholder's brainstorms or from a domain and the complete list of requirements is created from the integration of the requirements taken from all the different viewpoints.

There are two types of requirements: Those which are specific to a viewpoint and those which are global to the system. The latter are called concerns. Concerns are non-functional requirements created from the activities' main objectives. They are crucial to the success of the project, and to be effective they should be detailed in specific and verifiable requirements. PREview assumes that the number of concerns for a specific application is small; never more than 6, otherwise it will be difficult to deal with. Viewpoints in the PREview method are specified using the template depicted in table 2.1.

Table 2.1: PREview template.

| | |
|---|---|
| Name | This is used to identify and refer to the viewpoint and should normally be chosen to reflect the *focus* of the viewpoint. The name may reflect a role in the organization or a part of the system or process to which the analysis is restricted. |
| Focus | A viewpoint's focus defines the scope of the viewpoint. It is expressed as a statement of the perspective adopted by that viewpoint. |
| Concerns | The viewpoint concerns reflect the organizational goals, business objectives and constraints which drive the analysis process. |
| Source | Viewpoint sources are explicit identifications of the sources of the information associated with the viewpoint. |
| Requirements | This is the set of requirements arising from analysis of the system from the viewpoint's focus. The requirements may be expressed in terms of system functionality, user needs or constraints arising from application domain or organizational considerations. |
| Change History | This records changes to the viewpoint as an aid to traceability. It includes changes to the focus, the sources and the requirements encapsulated in the viewpoint. |

Concerns can be described using the template in table 2.2, where basically we have its description and a list of requirements.

Table 2.2: Concern template.

| | |
|---|---|
| Concern name | Identifies the concern |
| Description | Description of how the concern shall influence the system. |
| Requirements | There is a requirement for each viewpoint influenced by the concern. |

### 2.1.2 Other Approaches

There are several viewpoint-oriented approaches for requirements, like VORD (Viewpoint-Oriented Requirements Definition) [16, 18], the one described by Leite and Freeman [19], and the approach described by Nuseibeh, Kramer and Finkelstein [20].

**VORD**

The VORD process includes viewpoint identification, viewpoint service description, cross-viewpoint analysis to discover inconsistencies, omissions and conflicts. VORD also allowed services to be described in any appropriate notation (informal, structured or formal), In addition, multiple specifications, in different notations of the same services can be provided and linked.

**Leite & Freeman Approach**

The Leite and Freeman approach describes a viewpoint-oriented approach where a viewpoint is a mental state. From this definition, they continue describing the approach to requirement elicitation based on collecting information from different viewpoints and implementing views, which is an integration of different types of information. As a viewpoint is a mental state, an individual may provide information from several viewpoints, making this an one-to-many mapping approach. This reflects the fact that people often have different roles in an organization.The main characteristic of this approach is the automatic verification and error detection provided during early stages of the elicitation process.

**Nuseibeh, Kramer and Finkelstein**

Nuseibeh, Kramer and Finkelstein propose a flexible approach, allowing the specification of multiple viewpoints using no predefined notations. Nuseibeh and Finkelstein define a

viewpoint as a way to encapsulate an individual technique, with a defined notation, a set of actions that can be performed on that notation, and a set of rules for consistency relationships with other viewpoints. This way, the design and integration of multiple methods can be supported as a process of creating and tailoring viewpoint templates. This differs from PREview, because in the latter there is a predefined notation to represent a viewpoint, restricting the approach.

The approaches above are limited, as they do not address the crosscutting nature of requirements at early stages.

## 2.2 Aspect-Oriented Requirements Engineering

### 2.2.1 Concepts

The crosscutting nature of some requirements is not represented properly in the domain requirements, but this can be solved by integrating the Aspect-oriented Requirements Engineering concepts. A crosscutting or aspectual requirement is a requirement that affects or influences several other requirements [21], so that: It may constrain the specified behavior of the affected requirements; it may influence the affected requirements in order to change their behavior.

AORE aims to *provide a systematic means for the identification, modularization, representation and composition of crosscutting properties, both functional and non-functional ones, during requirements engineering* [1, 21]. One of the terms to highlight here is *composition*. AORE techniques have a strong focus on composability by providing a refined specification of how a requirements level aspect constrains or influences specific requirements in a system. Such a detailed understanding of the composition relationships between aspectual and non-aspectual requirements leads to an improved understanding of their interaction, inter-relationships and conflicts. This helps to identify trade-offs early in the development life cycle and undertake negotiations with the affected stakeholders. Furthermore, the aspectual requirements and their associated trade-offs can be traced to implementation to ensure that they have been preserved in line with the requirements specification.

An AORE approach is characterized by [1]:

1. The existence of effective means to identify crosscutting properties in a requirements specification. This may be accomplished by providing intuitive guidelines for the purpose, identifying specific keywords or action words in a requirements specification, or using specific tool support.
2. The ability to modularize crosscutting properties connecting to a particular concern in one requirements-level module, i.e., a requirements-level aspect. It is the ability to modularize a crosscutting set of requirements that is the distinguishing characteristic and not the symmetry or asymmetry of the modularization mechanism.
3. The supply of suitable means to represent requirements-level aspects. Again, such representation mechanisms may differ, depending on the application domain or availability of relevant tools.
4. The ability to compose aspectual requirements and non-aspectual requirements to clearly understand the cumulative effect of requirements-level aspects on other system requirements.

Though not an essential characteristic, an AORE approach should be complemented with effective facilities to trace aspectual requirements and any associated trade-offs to latter development stages. Often requirements-level trade-offs are resolved by weakening certain requirements in order to strengthen others as a result of negotiations with stakeholders.

### 2.2.2 Aspect-Oriented Approaches

There are several approaches to AORE, but from those only five are detailed in this section: Aspect-Oriented Component Requirements Engineering (AOCRE) [22], Aspect-Oriented Requirements Engineering (AORE/ARCADE) [21, 23], THEME [24] and the Aspect-Oriented Software Development/Use Cases (AOSD/UC) from Jacobson and Ng [25].

**AOCRE**

The AOCRE approach was one of the first to be proposed. In AOCRE there is a categorization of various aspects of a system (e.g., persistence, distribution) that each component makes available to other components or users. It is committed to component based software development, but does not present evidence of its use in other sorts of software

development. The basic AOCRE process begins after analyzing general application requirements or individual or groups of components requirements. This allows iterative top-down and bottom-up requirements refinement. Engineers characterize a component's aspects, aspect details, provided and required details, functional and non-functional properties, and reason about interrelated components' aggregate aspects. Components and aspects identified are refined into detailed component designs.

**AORE/ARCaDe**

The AORE approach proposes a model that supports separation of crosscutting functional and non-functional properties at the requirements level. These crosscutting properties are classified as candidate aspects. It also provides identification of their mapping and influence on later development stages. The approach is refined in with PREView and XML, where detailed composition rules for aspectual requirements are defined and also separated. They are used to specify how an aspectual requirement influences or constrains the behavior of a set of non-aspectual requirements. Also, early separation of crosscutting requirements makes it possible the determination of their mapping and influence on artifacts at later development stages. Moreover, a conflict resolution scheme is presented and the approach is supported by a tool called ARCaDe which manipulates requirements organized into concerns and viewpoints specified in XML. However, the approach has been only used to identify non-functional concerns as candidate aspects. Our approach is similar to this one as it is based on PREview, but we will adapt it to Domain Analysis and propose a simpler composition mechanism.

**THEME/Doc**

Baniassad and Clarke propose Theme to provide support for aspect-oriented analysis through Theme/Doc [24]. In this approach, themes are characterized as being *more general than aspects and more closely encompass concerns*. Theme is divided into three main activities: Analysis, which involves mapping requirements to concerns in the system; Design, which consists o filling in the design details and make changes that are needed to the design and Composition, which specifies how themes will relate with each other: overlapping or crosscutting one another. This approach is supported by a tool called Theme/UML, which creates graphs of the relationships between concerns and the requirements that mentioned those concerns. One problem with Theme/Doc is that it does not provide enough support for requirements of large scale systems.

**AOSD/UC**

Ivar Jacobson and Pan-Wei Ng demonstrate how to apply use cases and aspect-orientation in building robust and extensible systems. This approach modularizes the requirements into use cases, each of which describes an interaction between the actor and the software system for a certain service, as is done for the classic use cases approach. They also use aspects to map the code from use cases onto domain objects, so they can avoid tangling code between peer use-cases. They develop the idea that a use-case approach can be the basis for aspect-oriented software engineering, representing each use case as an aspect and introducing new concepts such as use-cases slices (a modularity unit where all the specifics of a use case are) and use case modules (a module encapsulating all the elements and artifacts specific to a use-case).

## 2.3 Domain Analysis and Feature Modeling

### 2.3.1 Domain Analysis

Domain analysis [26] is defined as a process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable when creating new systems. During software development, information of several kinds is generated, from requirements analysis to specific designs to source code. Source code is at the lowest level of abstraction and is considered the most detailed representation of a software system.

One of the objectives of domain analysis is to make all the information generated during software development available. In making a reusability decision, that is, in trying to decide whether or not to reuse a component, a software engineer has to understand the context which prompted the original designer to build the component the way it is, that is, the chain of design decisions used in the development process are absent in the source code.

By making this development information available, a re-user has the power to make reuse more effective. A bigger improvement of the reuse process is achieved when one is able to derive common architectures, generic models or specialized languages using domain analysis. How can one find these architectures or languages? It is by identifying features common to a domain of applications, selecting the objects and operations that characterize those features,

and creating procedures that automate those operations. This intensive activity results, after several of the similar systems have been constructed. It is then decided to isolate, encapsulate, and standardize certain repeated operations.

Summing up, identifying and structuring information for reusability is the process of domain analysis. According to [26], the main concepts within DA are:

- *Domain:* In the context of software engineering it is most often understood as an application area, a field for which software systems are developed. Examples include airline reservation systems, payroll systems, communication and control systems, spread sheets, numerical control. Domains can be broad like banking or narrow like arithmetic operations. Broad domains consist of groups of interrelated narrower domains usually structured in a directed graph. Domains, therefore, can be seen as networks in some semi-hierarchical structure where primitive, narrow domains such as assembly language and arithmetic operations are at the bottom and broader, more complex domains are at the top. Domain complexity can be characterized by the number of interrelated domains they require to be operational.

- *Domain Boundary:* Each domain in these domain networks is limited by a boundary that defines its scope. The borders define what objects, operations, and relationships belong to each domain and delimit their operational capability.

- *Domain Analysis:* As stated in the introduction of this chapter, domain analysis can be seen as a process where information used in developing software systems is identified, captured, structured, and organized for further reuse. More specifically, domain analysis deals with the development and evolution of an information infrastructure to support reuse. Components of this infrastructure include domain models, development standards, and repositories of reusable components. Domain and boundary definitions are also activities of domain analysis.

A standard definition of domain analysis is yet to come. Due to the nature of the activities and issues involved and to the fact of being so recent, domain analysis is seen differently by different communities.

### 2.3.2 Feature Modeling

Feature models are a good way of expressing requirements in a domain on an abstract level, according to [27]. They are used to describe the common and variable properties in a product line and to derive and validate configurations of software systems.

A feature model represents a hierarchy of properties of domain concepts. These properties are used to discriminate between systems or applications within that domain. At the root of the hierarchy there is a so-called concept feature, representing a whole class of solutions. Underneath this concept there are hierarchically structured sub-features showing refined properties. Each of the features is common to all instances unless marked as being optional, not necessarily being part of all instances.

According to [28], feature modeling is a key approach to capturing and managing the common and variable features of systems in a system family or a product line. In the early stages of software family development, feature models provide the basis for scoping the system family by recording and assessing. Later, feature models play a central role in the development of a system family architecture, which has to realize the variation points in the feature model.
In application engineering, feature models can drive requirements elicitation and analysis. Knowing which features are available in the software family may help customers decide which features their system should support. Knowing which desired features are provided by the system family and which has to be custom developed helps to better estimate the time and cost needed for developing the system. A software pricing model could also be based on the additional information recorded in a feature model.

Feature models also play a key role in generative software development [29, 30]. Generative software development aims at automating application engineering based on system families: a system is generated from a specification written in one or more textual or graphical domain-specific languages (DSLs). In this context, feature models are used to scope and develop DSLs, which may range from simple parameter lists or feature hierarchies to more sophisticated DSLs with graph-like structures.

Feature modeling was proposed as part of the Feature-Oriented Domain Analysis (FODA) [3] method, and since then, it has been applied in a number of domains including telecom systems, template libraries, network protocols, and embedded systems.
The feature-model in figure 2.2 depicts a Home integration system (HIS) [31].

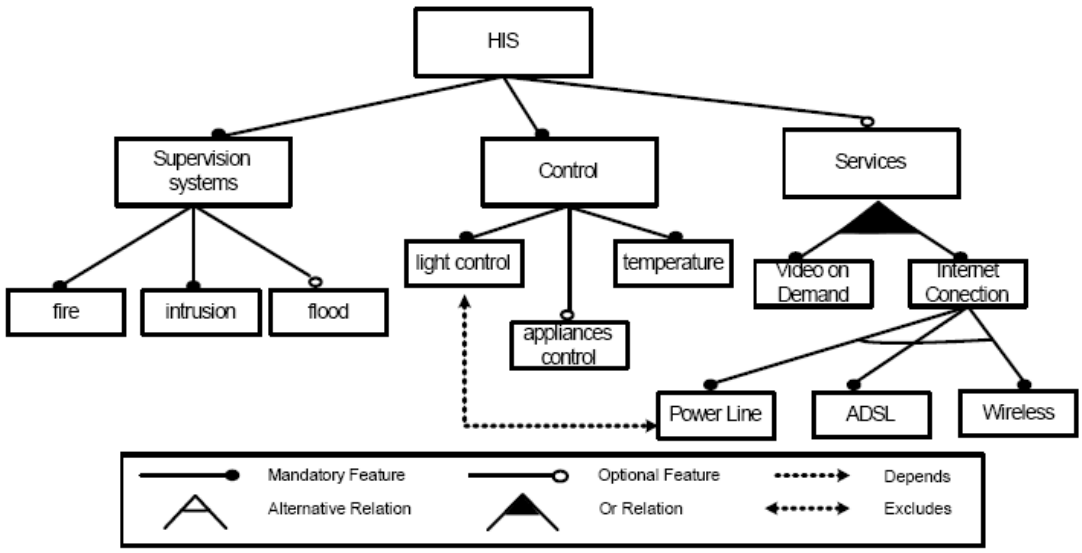


Figure 2.2: HIS feature model [31].

As the legend implies there are mandatory features (e.g. control, supervision systems), optional features (e.g. flood, services), alternative relations (e.g. ADSL, wireless), OR relations (e.g. Video On Demand and Internet Connection) also in this feature-model one can see that the light control excludes the power line.

Other representations are possible, specially the directory-like representation used in many tools, for example, the Feature Modeling Plug-in (FMP) for Eclipse [32]. FMP is an Eclipse plug-in for feature modeling and configuration. It supports a particular form of feature modeling, which is referred to as cardinality-based.

In figure 2.3 it is represented a simple model, similar to the one used in the case study, where it is represented an electronic shop diagram [33], where full bullets represent mandatory features (e.g. Catalog), empty bullets represent optional ones (e.g. Registration and Wishlist) and also is represented an OR relation between “Checkout” and “Review”.

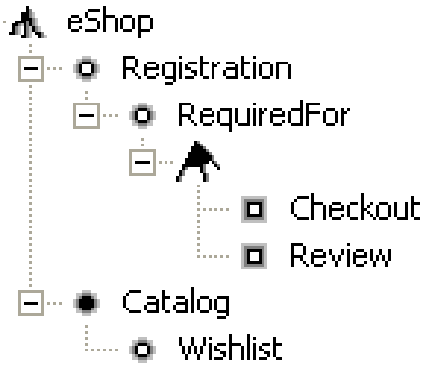


Figure 2.3: FMP simple model [33].

### 2.3.3 Approaches to Domain Analysis

In this section, it will be described different approaches used in DA, mainly Object-Oriented Domain Analysis (OODA) [2, 34], Context-Oriented Domain Analysis (CODA) [3] Feature-Oriented Domain Analysis (FODA) [4] and Aspect-Oriented Domain Analysis (AODA) [1].

**Object-Oriented Domain Analysis (OODA)**

This approach to domain analysis is based on objects and has four main goals: Capture domain-specific knowledge of the application in a form which allows point-to-point verification by domain experts; provide through formal models of the problem domain, a detailed and well documented foundation upon which requirements decisions can be made; transfer the domain knowledge accurately to the software designers and programmers; and finally, communicate the analysis in a form that is easily mapped into an OO design, as well as into alternative, non-OO design schemes.

In order to accomplish these goals OODA is based on building three types of formal models: an information model, a set of state models, and a set of process models. These three types of models are used together with prescribed rules of integration and by the order in which the models are constructed: Information Models, State Models, Process Models, and Boundary Statement/Requirements Definition.

An OO approach has usually three main concepts: Object, Attribute and Relationship.

An object is an abstraction of a set or real-world things such that all real-world things in the set – the instances – have the same characteristics; and all instances are subject to and conform to the same rules.

An attribute is the abstraction of a single characteristic possesses by all the instances that were themselves abstracted as an object.

A relationship is an abstraction of a systematic pattern of association that holds between real-world things that were themselves abstracted as objects.

Most DA methods use an information model that can be OO. Objects offer a superior means of modeling system behavior and should be applicable in modeling commonality and variability.

**Context-Oriented Domain Analysis (CODA)**

Context-Oriented Domain Analysis (CODA) is a specialized approach for analyzing, structuring, and formalizing the software requirements of context-aware systems.

CODA is intended to be relatively simple and concise to lower the accessibility barrier for various kinds of stakeholders while being expressive enough to evolve towards the solution space. In contrast to general-purpose methods for requirements analysis, like use cases [10], and goal models [9], CODA is exclusively specialized for context-aware (functional and nonfunctional) requirements.

It is an approach for modeling context-aware software requirements in a structured, well-defined, and unambiguous way.

The CODA model enforces software engineers to think of context-aware systems as pieces of basic context-unaware behavior which can be refined. The main strength of the refinement is the context in which the system is used. So it is defined a term called *context-dependent adaptation* which is defined as *A unit of behavior which adapts a subpart of a software system only if an associated context condition is satisfied*. The principle of distinguishing basic behavior and context-dependent adaptations lies at the heart of this CODA approach (figure 2.4).

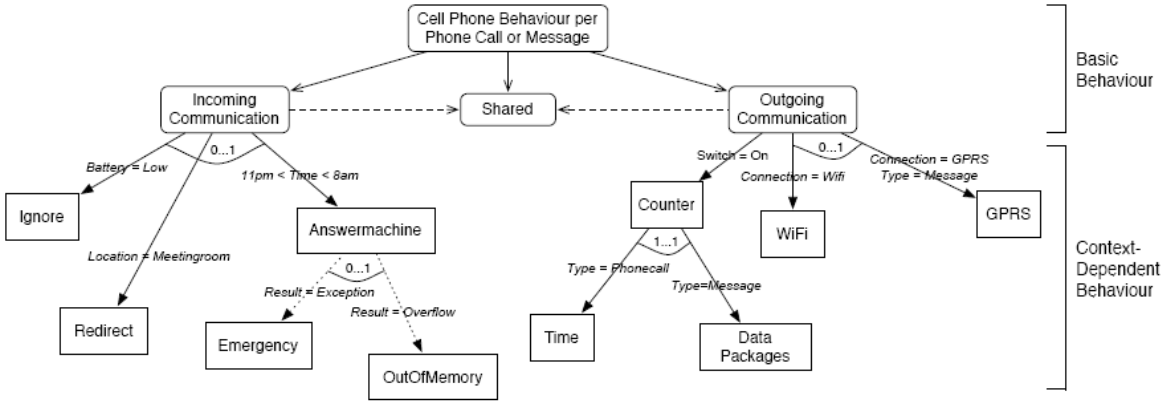


Figure 2.4: CODA example model [3].

CODA establishes a vocabulary, which is used to describe the model. First of all, the model is divided in basic behavior which is mandatory for all and context-dependent behavior which is the variability within the domain.

Context-dependent adaptations are represented by means of rectangular boxes which are attached to relevant variation points

Relationships can be divided in three kinds: inclusions, conditional dependencies and choice points.

- The inclusion relationship means that only if adaptation *A* is applicable, the applicability of adaptation *B* should be verified.
- The conditional dependency relationship has a temporal character: If the return value of adaptation *A* equals *r*, then *B* should be executed subsequently
- and finally variation points and context-dependent adaptations can have multiple context-dependent adaptations associated to them.

Although CODA might seem as being far removed from the solution space, since it has a well-defined syntax and semantics, it can be easily mapped to decision tables which brings it very close to the computational level.

The aim of the CODA approach is to have a concise modeling language for context-aware systems which is accessible to various kinds of stakeholders. Since CODA has a well-defined syntax and semantics, it possesses a sound basis for evolving towards the solution space. However, a deeper understanding of the mapping from CODA to the computational level is still under investigation.

**Feature-Oriented Domain Analysis (FODA)**

One of the most popular domain analysis approaches is Feature-Oriented Domain Analysis (FODA), which specifies the features of a domain through tree-like diagram where commonalities and variabilities are represented.

The Software Engineering Institute (SEI) developed the FODA approach. FODA is based on identifying "features" of a class of systems. A feature is a prominent, user-visible aspect of a system. Domain analysts identify both features that are common to all systems and features that distinguish individual systems or subclasses of systems within a domain.

The FODA process defined three basic activities (see figure 2.5):

- *Context analysis*, where domain analysts interact with users and domain experts to scope the domain. The product of context analysis is a context model. Domain analysts and domain developers use it in subsequent domain engineering activity to understand domain boundaries.
- *Domain modeling*, which produces a domain model with multiple views:
    - A feature model, which is the end user's perspective of capabilities (both common and variable) of applications in a domain.
    - An entity-relationship-attribute (ERA) model, which defines the objects in the domain and their interrelationships. The model is a developer's view: domain and application developers use this information as a basis for deriving implementations of reusable components and applications.
    - Data flow and finite state machine (FSM) models, which are the requirement analyst's view of the functionality of applications within a domain.

The features and ERA models guide and constrain their development, that is, they reflect the commonalities and variations expressed in the features model, and the objects in the ERA model define them. Domain analysts use them subsequently in architectural modeling. Application developers also refine them into requirements for specific applications during application development.

Architecture modeling, which produces high-level design specifications of solutions to the "problems", is defined in the domain model. Architectural modeling produces a model of interacting software processes and a module structure chart. Domain developers use these products as specifications for reusable components. Application developers refine the components into products that meet their application's needs.

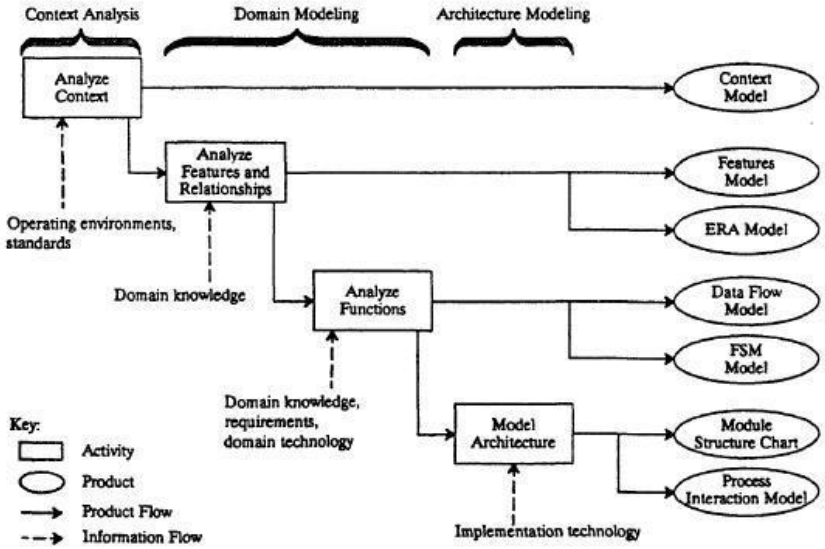


Figure 2.5: FODA's Domain Analysis Process [35].

**Aspect-Oriented Domain Analysis (AODA)**

Despite having achieved many successful results in the area of domain analysis, the notion of aspects has not been addressed explicitly yet [1]. Two basic reasons for using domain analysis in the aspect-oriented development are:

1. *Identification of aspects from the solution domain:* Initially in AOSD, aspects have been identified at the code level by code analysis techniques. The general definition of aspects as being concerns which tend to crosscut over a broader set of modules can, however, be applied throughout the life cycle. As a result, aspects might occur throughout the whole software life cycle. To avoid the problems related to crosscutting, these crosscutting concerns must be identified and specified separately. To cope with this issue recently more attention has been paid to identifying aspects at the requirements analysis and architecture design. Aspects identified during the requirements analysis might not always

be sufficient to cover all aspects, because the requirements analysis focuses on the problem domain rather than the solution domain.

On the other hand, identifying aspects at the architecture design might be too late. To complete the process of aspect identification it is necessary to also consider aspects in the solution domain which is the domain which includes the solution for the requirements in the problem domain. Unfortunately there is no approach yet for identifying aspects from the domain.

2. *Implementing aspects based on solution domain models:* Before implementing the aspects it is required that their structure, properties and behavior are sufficiently understood, otherwise, there will be no solid basis for guaranteeing that the aspect is defined and implemented in a proper way. It seems that in current AOSD practices the analysis of aspects is not explicit and the implementation of aspects depends mostly on the experience and background of the aspect programmer. This will certainly lead to unstable aspects which will need to be modified. Domain models are generally considered as stable abstractions for implementing concerns. In a similar way, to understand the structure of aspects at the more concrete implementation level we need explicit domain models for aspects.

Domain analysis has proven its value for a wide range of applications and domains. But aspects are a new type of concern that was not taken into account before in existing domain analysis approaches. Moreover, aspects impose new types of constraints and extensions to identification and specification of domains (for aspects). So there is a whole new kind of domain analysis to explore and analyze.

### 2.3.4 SPL (Software Product Lines) approaches to Feature Modeling

Feature modeling is commonly used to specify software product lines. They have been used in conjunction with use case driven approaches and goal-oriented approaches described next.

**Use Cases and SPL**

An approach to model use cases to SPLs is discussed in [36]. According to Gomaa [36], to specify functional requirements of SPLs in use case approaches, it is necessary to define different kinds of use cases:

- Kernel use cases, which are necessary to all members of a product line;
- Optional use cases, which are necessary by some members of a product line;
- Alternative use cases, where different use cases are necessary by different members of the product line.

It was introduced the concept of "variation point", which is the location in a use case where change can take place, and can be modeled through the relationships of Include and Extend.
To model use cases to product lines, Gomaa introduced feature modeling for the UML approach. Based on the reusable properties of both approaches, use cases can be mapped onto features. A feature could correspond to a unique use case, a group of use cases or a variation point within a use case, this way use cases and features can complement each other.
There are several ways of modeling features with UML:

- Use Case Packages;
- Metaclasses;
- Tabular representation.

Since this approach deals with functional requirements of a SPL, this proposal shows that use case modeling can be used to represent SPLs. However, it does not contemplate non-functional requirements, which are important in any system development.

**Framework i* and SPL**

In [37] it is proposed the use of i* approach to capture common and variable features in SPL requirements. According to Silva et al [37], the selection of specific features and their decomposition for an individual product with the most important resources of SPLs is eased as it is aspect oriented. Combining aspectual i* with approaches that extract earlier the variability in the life cycle of a SPL development, enriches the aspectual i* variability. This variability can ease the relationship between several types of requirements (for instance, functional and non-functional) in a same model.

To illustrate how a goal oriented strategy can be used to represent variability in requirements models, the authors proposed a series of heuristics to create aspectual i* models from feature models:

- Separation of features in i* model;

- Types of features and model relationships;
- Correctness verification in relations and the mapping between features names and the names of the aspectual i* model elements.

However, the approach needs to be improved to reduce the model scalability, because with the aspect addition, they also have to be represented in the model, hardening its comprehension and legibility. The approach considers that each optional and alternative feature in the model will be mapped into an aspect, and that is not always true. In general, there is less capability of the models complexity management, the representation of all variability makes difficult its comprehension.

**Feature and Goal oriented models**

In [38], the authors, proposed a new extension model oriented, to a tool of Initial Requirements Engineering (OpenOME), which generates an initial feature model of the future system to the stakeholders goals. These goals are identified and analyzed with GORE (Goal Oriented Requirement Engineering), even before the functional and non-functional requirements have been projected. Goal models, as a result of the elicitation process, are a natural source of intentional variability.

According to Yu et al. [38], by the usage of generating programming, feature models represent the variability of the system and lead to the final products configuration. In this proposal, a description is made of how the tool supports the approach and how it manages the traceability between goals and features. Firstly, a feature model is made of how the system is going to be, then a connection to features system oriented is made and finally they are configured using goal reasoning algorithms. In this proposal, it is easy to lead the analyst to mistakes, because the tool automatically generates models, and if the tool is not trustworthy it can implicate future problems.

**MATA (Modeling Aspects using a Transformation Approach) and SPL**

In [39], an approach is described to maintain the separation of features during modeling using a composition language based on graph transformation and UML models, and an approach for detecting undesirable structure interactions between different feature models. According to Jayaraman et al. [39], the basic features are expressed in terms of class diagrams in UML, sequence diagrams and state diagrams and the variable features are specified in UMLT (a

UML representation of graph transformation which indicates how the base models modification are made). This description corresponds to MATA technique which is used in the aspects context.

Features are composed automatically using a graph rewriting mechanism and a double critical analysis is used to find structural feature interaction. These interactions allow verifying if there are inconsistencies between the feature dependency diagram and the UML feature models.

This approach has tool support. As each variable feature is modeled independently one from another, one can, at any time choose a subgroup of available features and generate automatically a model to that group of features. So, this approach is a good example of MATA applied to SPLs. Our approach is similar to this one as it is uses roles as well, but we will adapt and use them to instantiate crosscutting requirements.

# Chapter 3
# A Viewpoint and Aspect-Oriented Domain Analysis Approach

Most of feature modeling approaches have few integration with other domain specification approaches. Their integration should give a broader vision of the domain.

Crosscutting concerns are not specifically addressed, so the approach we propose in this dissertation tries to fulfill that gap. This approach integrates domain analysis with aspect, feature models and viewpoints and it should specify models at both domain engineering and application engineering, but so far only the domain engineering part has been achieved.

## 3.1 How to approach the problem at the domain engineering level

According to [12], domain engineering is a process for creating a competence in application engineering for a family of similar systems. Domain engineering covers all the activities for building software core assets (figure 3.1). These activities include identifying one or more domains, capturing the variation within a domain (domain analysis), constructing an adaptable design (domain design), and defining the mechanisms for translating requirements into

systems created from reusable components (domain implementation). The products (or software assets) of these activities are domain model(s), design model(s), domain-specific languages, code generators, and code components. Our focus is on the Domain Analysis activity.

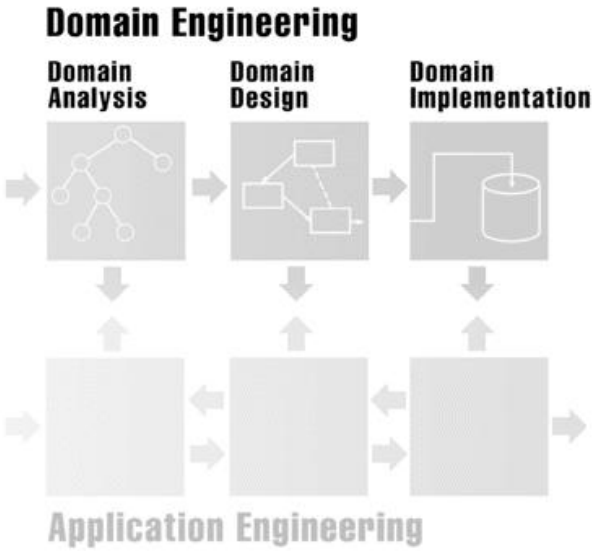


Figure 3.1: Domain engineering process.

## 3.2 Overview of VAODA

We propose an approach called Viewpoint and Aspect-Oriented Domain Analysis (VAODA) which is composed by six main activities:

- Identify and specify domain viewpoints and concerns;
- Specify a feature model;
- Identify crosscutting requirements;
- Specify aspectual viewpoints;
- Compose aspectual viewpoints;
- Integrate aspect-oriented viewpoints with feature model.

The identification and specification of domain viewpoints and concerns consists on taking from the domain documentation all the relevant viewpoints and concerns in order to build them. Then, the specification of the feature model can be done in parallel with the identification of crosscutting requirements, the specification of aspectual viewpoints or the

composition of aspectual viewpoints because despite being independent activities they help each other as the viewpoint specifies what features should be in the feature model and if there are repeated features in the feature model, there is a great chance of existing crosscutting concerns in the domain viewpoints.

The identification of crosscutting requirements within the domain viewpoints is followed by the specification of aspectual viewpoints, where all those crosscutting concerns are encapsulated, followed by the re-writing of the original viewpoints from where they were taken of. With all this specified, the next step is to compose the aspectual viewpoints. Finally, we integrate the aspectual viewpoints with the feature model in a tabular way, allowing one to check which features are mandatory and optional to each aspectual viewpoint.

This process is illustrated in figure 3.2.

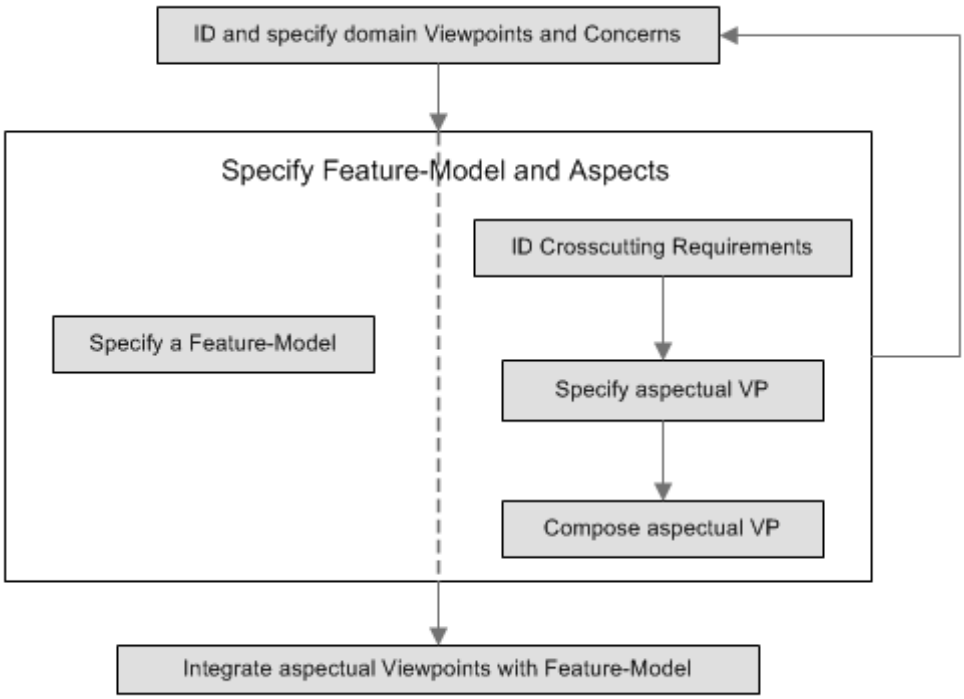


Figure 3.2: Our approach's process.

Our approach uses the concern template and PREview template described in section 2.1.1 with the addition of the field "Type" to the latter, that states if the viewpoint is aspectual or not (table 3.1).

Table 3.1: VAODA's viewpoint template.

| | |
|---|---|
| Name | Identifies the viewpoint. |
| Focus | Defines the viewpoint. |
| Concerns | By default, contains all the existent concerns in the system, yet some can be eliminated if proven not to be used. |
| Source | Specific identification of the information's source associated with the viewpoint. |
| Requirements | Set of requirements found by the system's analysis from the viewpoint's perspective. |
| Change History | Registers the changes in the viewpoint. Includes changing of focus, source and requirements. |
| Type | Aspectual or Non-Aspectual. |

## 3.3 Explaining the approach with an example

To illustrate the approach it will be used a car park example. This case study is a simplified version of the real system implemented in some Portuguese car parks. The requirements are stated as follows:

"To use a car parking system, a client has to get a ticket from a machine after pressing a button. Afterwards, the car is allowed to enter and park in an available place. The system has to control if the car parking is full or if it still has places left and if it is closed or not. When s/he wants to leave the parking place, s/he has to pay the ticket obtained (described above) in a paying machine. The amount depends on the time spent. After paying the client can leave by inserting the ticket in a machine which will open the gate for her/him to leave.

Any complaint related to the system (e.g. miscalculation of the ticket's price or any damages one to the vehicle) can be done to a park attendant that registers it in the system.

In addition to the previous problem, now we want to develop a new system that uses a gizmo like the used in the Green Lane for accessing the park. The Gizmo can be purchased through a simple process where the client gives her/his personal info, her/his debit card and the vehicle which will be associated to the gizmo. It's necessary to activate the gizmo in the ATM. Consider that the registration and the sign up for Green Lane have already been developed.

To enter the park it is necessary to have the gizmo placed in the vehicle's windscreen, and as the driver chooses to enter the park pressing a button, the gizmo is read and the gate is opened. To exit the park the process is similar but it is not needed to push any button. The

amount to pay is, in this case, shown in a display that is embedded in the exit machine. This amount is sent to the client's bank in order to debit. If any problem occurs in the transaction, the bank should notify the park system. A bill is sent to the vehicle's owner every month with the gizmo's passages."

### 3.3.1 Identify and specify domain viewpoints and concerns

Viewpoints represent an encapsulation of partial information about the system's requirements, of a particular perspective. They can be associated to stakeholders, environment and system domain. A viewpoint-oriented approach to requirements elicitation recognizes that all information cannot be obtained only considering one perspective.

The viewpoints identified in this example are: ATM, Bank, Driver, Entry Machine, Exit Machine, Gizmo, Park, Park Attendant, Paying Machine, Vehicle and Vehicle Owner.

The identified concerns are: Availability, Compatibility, Correctness, Multi-Access, Response Time, Security, Usability and Legal Issues. Templates of the viewpoints Entry Machine and exit Machine are shown in tables 3.2 and 3.3. Templates for availability and correctness are shown in tables 3.4 and 3.5. These four templates are just examples to illustrate the approach.

Table 3.2: Template for the Entry Machine viewpoint.

| | |
|---|---|
| Name | Entry Machine |
| Focus | Registers vehicle entry in the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | 1 - Entry Machine detects vehicle.<br>2 - Entry Machine detects entrance mode.<br>2.1 - Entry Machine gives ticket if ticket button is pressed.<br>2.2 - Entry Machine detects and reads Gizmo if Green Lane button is pressed.<br>2.3 - Entry Machine receives, reads and gives card back if Card is inserted.<br>3 - Entry Machine opens gate if entrance mode is valid.<br>4 - Entry Machine closes gate when the passage of the vehicle was detected.<br>5 - Entry Machine calls Park Attendant if Park Attendant button is pressed. |
| Change History | |
| Type | Non-Aspectual |

Table 3.3: Template for the Exit Machine viewpoint.

| | |
|---|---|
| Name | Exit Machine |
| Focus | Registers vehicle exit from the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | 1 - Exit Machine detects vehicle.<br>2 - Exit Machine detects exit mode.<br>2.1 - Exit Machine receives ticket.<br>2.2 - Exit Machine detects and reads Gizmo if vehicle entered using Green Lane.<br>2.2.1 - Exit Machine retrieves system info on the debit value.<br>2.2.2 - Exit Machine Displays the debit value.<br>2.3 - Exit Machine receives, reads and gives card back if Card is inserted.<br>3 - Exit Machine opens gate if exit mode is valid.<br>4 - Exit Machine closes gate when the passage of the vehicle was detected.<br>5 - Exit Machine calls Park Attendant if Park Attendant button is pressed. |
| Change History | |
| Type | Non-Aspectual |

Table 3.4: Template for the Availability concern.

| | |
|---|---|
| Concern name | Availability |
| Description | This describes where availability should be considered in the Car Park system |
| Requirements | 1. Car Park system must be available in order to:<br>1.1 Detect vehicle.<br>1.2 Detect access mode.<br>1.3 Open gate if access mode is valid.<br>1.4 Closes gate when the passage of the vehicle was detected.<br>1.5 Call park attendant if park attendant button is pressed. |

Table 3.5: Template for the Correctness concern.

| Concern name | Correctness |
|---|---|
| Description | This describes where correctness should be considered in the Car Park system |
| Requirements | 1. Car Park system must be correct in order to:<br>1.1 Detect vehicle.<br>1.2 Detect access mode.<br>1.3 Open gate if access mode is valid.<br>1.4 Closes gate when the passage of the vehicle was detected. |

### 3.3.2 Specify Feature Model and Aspects

As some activities described in the approach process can be done in parallel, they were encapsulated in a box called "Specify Feature Model and Aspects", which includes, on one hand, specifying a feature model and, on the other hand, identifying crosscutting requirements, specifying crosscutting requirements and finally composing aspectual viewpoints.

#### 3.3.2.1 Specify a feature model

Based on the information provided by the viewpoint requirements and the domain experts we can derive a feature model. In our example, figure 3.3 shows a feature model for the case study. In our approach this is done manually, by analyzing the viewpoint requirements.

For example, Entry Machine can be defined as a feature model decomposed into several obligatory features (e.g. display, gate, intercom, park attendant button and vehicle sensor), and some variable features, in this case the kind of access to the park (e.g. by card, by ticket or using Green Lane gizmo).

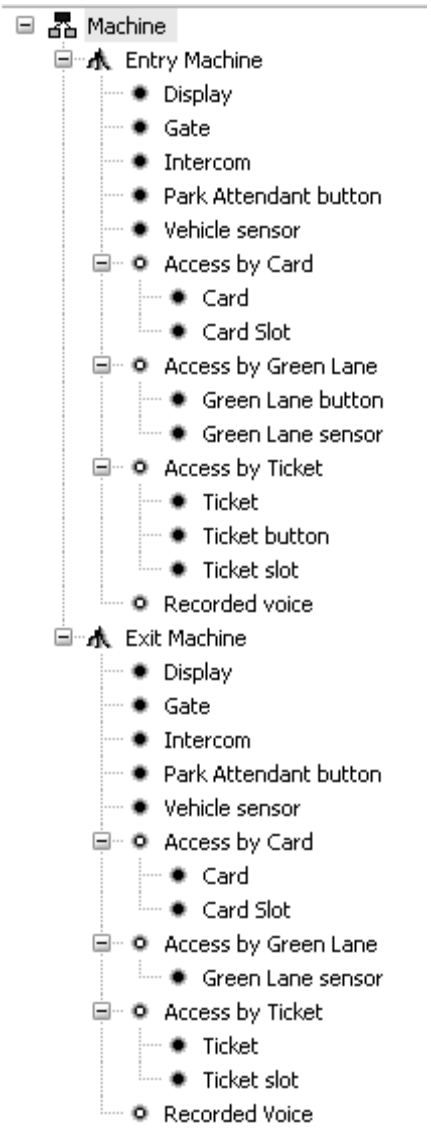

Figure 3.3: FMP model for the case study.

#### 3.3.2.2 Identify crosscutting requirements and specifying and composing aspectual viewpoints

There are requirements scattered in several viewpoints. Those are called "Aspectual Requirements" and are modularized in a new "Aspectual Viewpoint" that will now be defined.

Looking at the viewpoints above one can notice that some requirements are very *similar,* they only defer in the kind of machine involved. For example, we have:

- "Entry machine detects the vehicle", in Entry Machine viewpoint;
- "Exit machine detects the vehicle", in Exit machine viewpoint.

By removing these requirements from the original viewpoints we define aspectual viewpoints. Thus, a requirement to be considered an “aspectual” one, all his sub-requirements have to be similar too.

Sometimes this process can be confused with inheritance of viewpoints. Note that the requirements are similar not equal, as they differ on the subject. Therefore, we cannot specify a superviewpoint “Machine” with, for example, the requirement “Entry machine detects the vehicle” because this should not be inherited by the subviewpoint “Exit Machine”. Thus we will encapsulate this kind of requirement into an aspectual viewpoint and will be redefined using “Roles”. Roles are a kind of variable that can be instantiated and can deal with similar requirements. This technique has been successfully applied to specify crosscutting scenarios [15].

Table 3.6 shows the aspectual viewpoint Machine. The crosscutting requirements include roles (represented by the names preceded with a “|”) that are instantiable parts of the requirements, i.e. during composition they will be instantiated to a concrete value. In the example |Machine will be instantiated to Entry or Exit Machine.

Table 3.6: Template for the Machine aspectual viewpoint.

| | |
|---|---|
| Name | Machine |
| Focus | Registers vehicle entry/exit from the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | AR1 - \|Machine detects vehicle.<br>AR2 - \|Machine opens gate if entrance/exit mode is valid.<br>AR3 - \|Machine closes gate when the passage of the vehicle was detected.<br>AR4 - \|Machine calls Park Attendant if Park Attendant button is pressed. |
| Change History | |
| Type | Aspectual |

Aspects can be also defined for the non-functional concerns, which are often crosscutting. We will not focus on this here as the work in [21] deals with this in detail. The aspect templates for this are similar to the concern templates already described in Table 3.4 and 3.5.

Re-writing the viewpoints which had the aspects we obtain the new viewpoint templates in tables 3.7 and 3.8.

Table 3.7: New version of Entry Machine viewpoint.

| | |
|---|---|
| Name | Entry Machine |
| Focus | Registers vehicle entry in the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | 1 - Entry Machine detects entrance mode.<br>1.1 - Entry Machine gives ticket if ticket button is pressed.<br>1.2 - Entry Machine detects and reads Gizmo if Green Lane button is pressed.<br>1.3 - Entry Machine receives, reads and gives card back if Card is inserted. |
| Change History | |
| Type | Non-Aspectual |

Table 3.8: New version of Exit Machine viewpoint.

| | |
|---|---|
| Name | Exit Machine |
| Focus | Registers vehicle exit from the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | 1 - Exit Machine detects exit mode.<br>1.1 - Exit Machine receives ticket.<br>1.2 - Exit Machine detects and reads Gizmo if vehicle entered using Green Lane.<br>1.2.1 - Exit Machine retrieves system info on the debit value.<br>1.2.2 - Exit Machine Displays the debit value.<br>1.3 - Exit Machine receives, reads and gives card back if Card is inserted. |
| Change History | |
| Type | Non-Aspectual |

Firstly, we need to instantiate the roles in the aspectual viewpoint and then we compose them. Intuitive rules are shown below:

```
Compose Entry Machine with Machine
Bind |Machine to Entry Machine
     Add requirements AR1 to AR4 in Entry Machine.
```

Listing 3.1 Composition rule for Entry Machine in Domain Engineering

```
Compose Exit Machine with Machine
Bind |Machine to Exit Machine
     Add requirements AR1 to AR4 in Exit Machine.
```

Listing 3.2 Composition rule for Exit Machine in Domain Engineering

### 3.3.3 Integrate the feature model with the viewpoint model

The table 3.9 shows the relationships between features and viewpoints. Currently we are doing manually, but we hope to partially automate this process.

Fields marked with an “x” are considered mandatory whereas marked with “o” are considered optional. As one can verify, there is a Logical “OR” relation between the non-aspectual viewpoints and the aspectual one built based on the non-aspectual.

Table 3.9: Features VS viewpoints – Domain engineering.

| F\VP | | | Entry Machine | Exit Machine | Machine |
|---|---|---|---|---|---|
| Machine | Entry Machine | Display | x | x | x |
| | | Gate | x | x | x |
| | | Intercom | x | x | x |
| | | PA button | x | x | x |
| | | Vehicle sensor | x | x | x |
| | | Card | o | o | o |
| | | Card Slot | o | o | o |
| | | GL button | o | | o |
| | | GL sensor | o | o | o |
| | | Ticket | o | o | o |
| | | Ticket button | o | | o |
| | | Ticket slot | o | o | o |
| | | Recorded Voice | o | o | o |
| | Exit Machine | Display | x | x | x |
| | | Gate | x | x | x |
| | | Intercom | x | x | x |
| | | PA button | x | x | x |
| | | Vehicle sensor | x | x | x |
| | | Card | o | o | o |
| | | Card Slot | o | o | o |
| | | GL sensor | o | o | o |
| | | Ticket | o | o | o |
| | | Ticket slot | o | o | o |
| | | Recorded Voice | o | o | o |

## 3.4 How to approach the problem at the application engineering level

According to [12], Domain Engineering and Application Engineering are complementary, interacting and parallel processes that contain a model-based, reuse-oriented software production system.

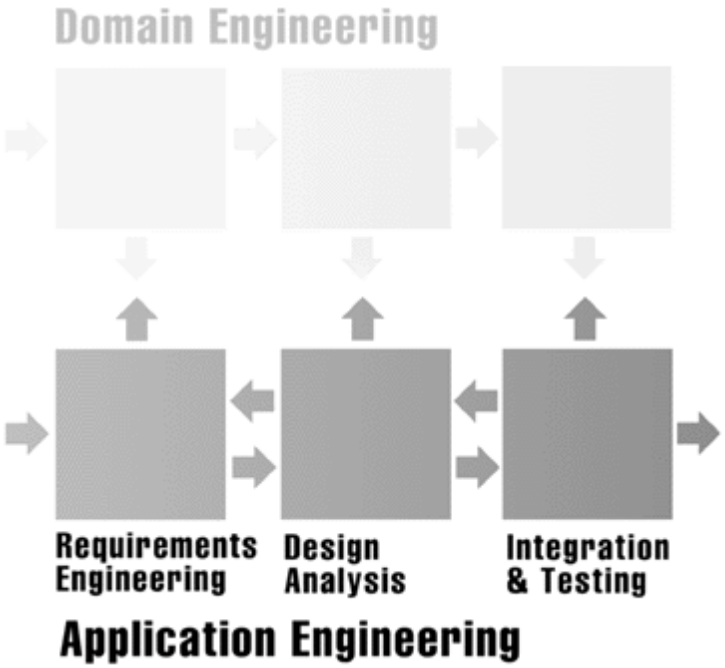

Figure 3.4: Application engineering process.

The focus of application engineering is on a single system while the focus of domain engineering is on multiple related systems within a domain. Usually, application engineering activities include using:

- a domain model to identify customer requirements;
- a generic design (design model) to specify a product configuration;
- a partitioning strategy and coordination model (architecture style) to guide custom development;
- Application generators and software components to produce application code.

Domain engineering is the personification of the principle of design-for-reuse whereas application engineering is the personification of the principle of design-with-reuse.

In VAODA, the application engineering process consists of 4 steps as depicted in figure 3.5:

- Configure Feature Model;
- Reuse Viewpoints according to configured Feature Model;
- Reuse composition rules according to configured Feature Model;
- Simplify the Viewpoint vs Feature Model table.

Configuring the feature model consists of defining an application model within the domain one. Reusing Viewpoints according to configured Feature Model is reusing the viewpoints declared in domain engineering adapted to the specific model configured before; composition rules are also reused according to the application model and finally the Viewpoints vs Feature Model table is simplified.

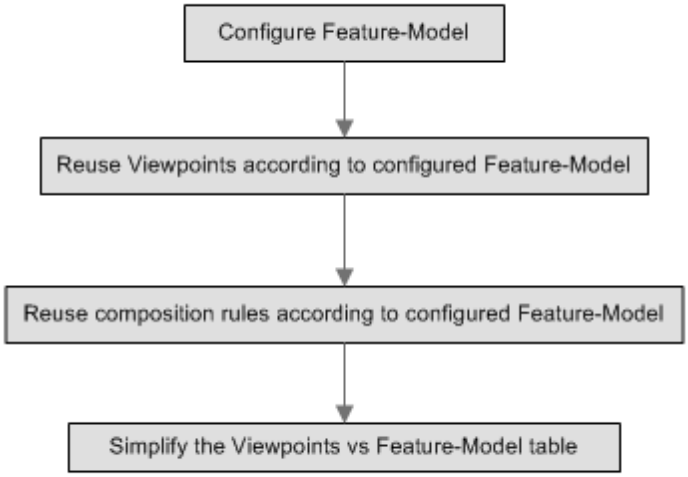


Figure 3.5: VAODA Application Engineering process.

The application engineering process defines how the system assets are used for the development of applications in the domain.

Below we apply the process to the example.

### 3.4.1 Configure the feature model

In this case, the Car Park considered will only be accessed by card, as it is a residential one, only granting access to residents of that specific building.

Based on the information provided by the viewpoint requirements, a feature model of the case study for application engineering can be made. In this case all features are mandatory, as this is a specific type of car park.

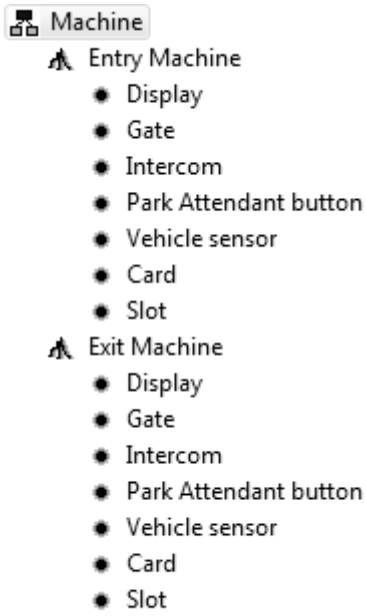


Figure 3.6: Feature Model for the application engineering car park.

### 3.4.2 Reuse Viewpoints and Aspects according to the configured Feature Model

There are several differences between the Car Park at the Domain level, and the Car Park at the Application level, as at the domain level all possibilities have to be taken into account: there will be no ATM, Bank, Gizmo, Paying Machine, for example.

The concerns in this exercise are the same as the ones used for the Domain engineering one: Availability, Compatibility, Correctness, Multi-Access, Response Time, Security, Usability and Legal Issues.

Templates for the viewpoints Entry Machine and Exit Machine are shown in tables 3.10 and 3.11.

Table 3.10: Template for the application engineering version of Entry machine viewpoint.

| Name | Entry Machine |
|---|---|
| Focus | Registers vehicle entry in the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | 1 – Entry Machine receives, reads and gives card back if Card is inserted. |
| Change History | |
| Type | Non-Aspectual |

Table 3.11: Template for the application engineering version of Exit machine viewpoint.

| | |
|---|---|
| Name | Exit Machine |
| Focus | Registers vehicle exit from the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | 1 – Exit Machine receives, reads and gives card back if Card is inserted. |
| Change History | |
| Type | Non-Aspectual |

The aspectual viewpoints also have to be simplified to just express the access by card. Table 3.12 shows the aspect machine for card access.

Table 3.12: Template for the application engineering version of Machine viewpoint.

| | |
|---|---|
| Name | Machine |
| Focus | Registers vehicle entry/exit from the park |
| Concerns | Availability, Correctness, Multi-Access, Response Time, Security |
| Source | Assignment |
| Requirements | AR1 – \|Machine detects vehicle.<br>AR2 – \|Machine opens gate if card is valid<br>AR3 – \|Machine closes gate when the passage of the vehicle was detected.<br>AR4 – \|Machine call Park Attendant if Park Attendant button is pressed. |
| Change History | |
| Type | Aspectual |

### 3.4.3 Reuse composition rules according to configured Feature Model

The composition rules are very similar to the ones already specified in the DE.

```
Compose Entry Machine with Machine
Bind |Machine to Entry Machine
     Add requirements AR1 in Entry Machine.
     Add requirements AR2 to AR4 in Entry Machine.
```

Listing 3.3 Composition rule for Entry Machine in Application Engineering.

```
Compose Exit Machine with Machine
Bind |Machine to Exit Machine
     Add requirements AR1 in Exit Machine.
     Add requirements AR2 to AR4 in Exit Machine.
```

Listing 3.4 Composition rule for Exit Machine in Application Engineering.

### 3.4.4 Simplify the Viewpoints vs Feature Model table

In this case, as it was seen above all features of the configured feature model are mandatory so all will be present in Entry and Exit Machines as well as in Machine.

Table 3.13: Features vs viewpoints – Application Engineering.

| F\VP | | | Entry Machine | Exit Machine | Machine |
|---|---|---|---|---|---|
| Machine | Entry Machine | Display | x | x | x |
| | | Gate | x | x | x |
| | | Intercom | x | x | x |
| | | PA button | x | x | x |
| | | Vehicle sensor | x | x | x |
| | | Card | x | x | x |
| | | Slot | x | x | x |
| | Exit Machine | Display | x | x | x |
| | | Gate | x | x | x |
| | | Intercom | x | x | x |
| | | PA button | x | x | x |
| | | Vehicle sensor | x | x | x |
| | | Card | x | x | x |
| | | Slot | x | x | x |

## 3.5 Tool support (Aspect Finder)

In addition to the approach, a tool was developed in order to find the candidates to aspectual requirements among the viewpoints of a given system. This tool was developed in Java, also using the Swing widget toolkit (an API for providing a graphical user interface (GUI) for Java programs). The Aspect Finder is organized in 3 major packages: Interface, Manager and Data, which are connected as follows:

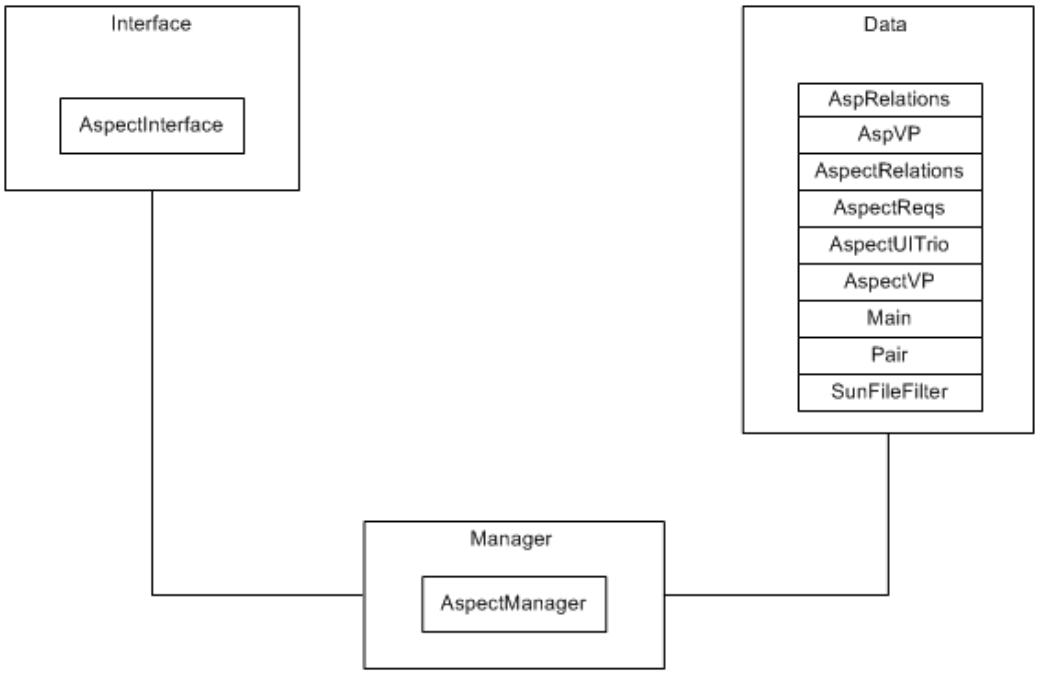


Figure 3.7: AspectFinder Packages.

In the Data package, all the structures for the prototype are implemented, in the Manager package, the class AspectManager is implemented where are the structures are filled, and all the functions concerning the management of the structures are implemented. Finally, in the Interface package, the class AspectInterface is implemented. This is where all the functions related to the interface are implemented.

### 3.5.1 The Interface

Aspect Finder is divided into two tabbed panels: the "Viewpoint" tab (figure 3.8) and the "Aspect Finder" tab (figure 3.9).

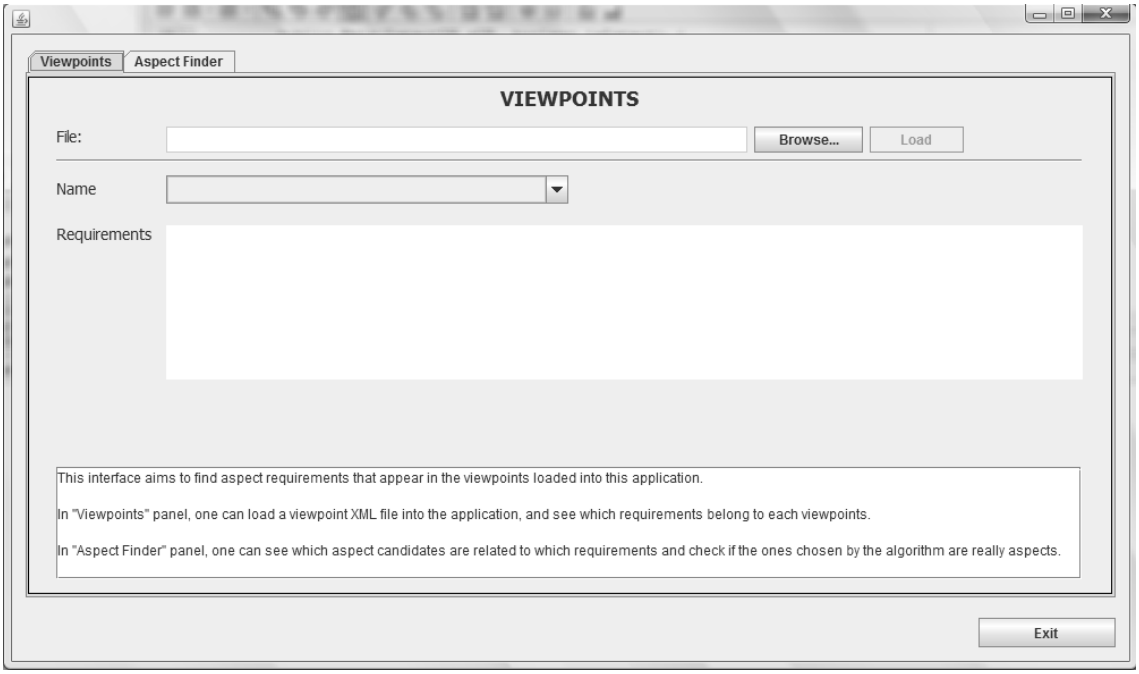


Figure 3.8: Viewpoint panel.

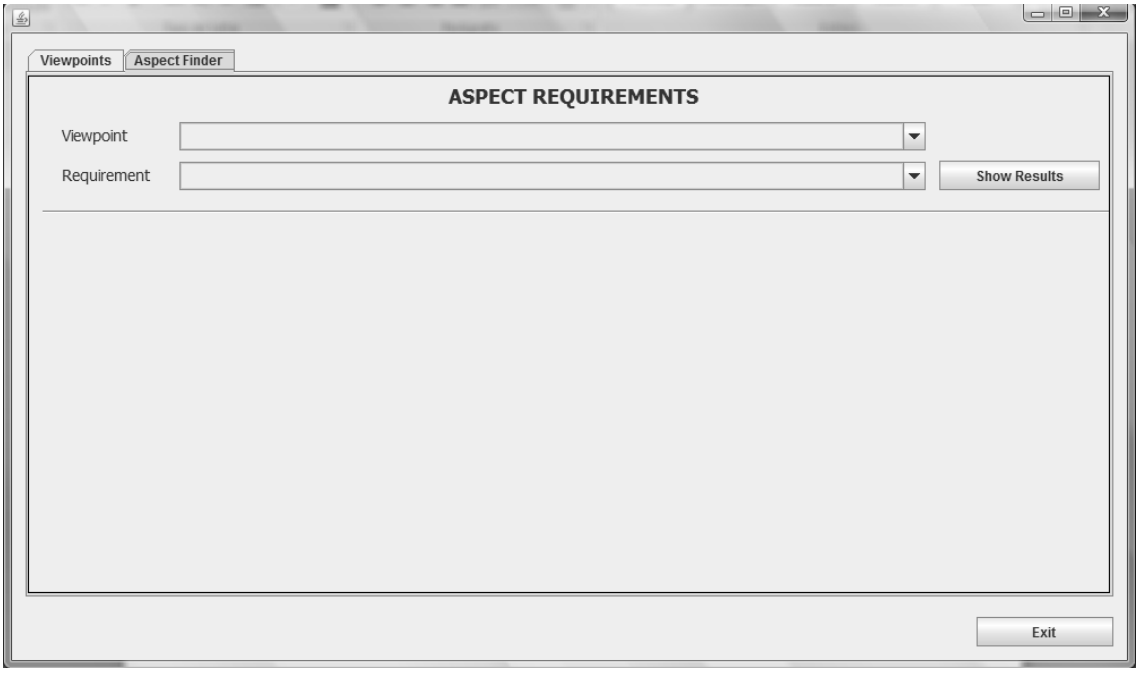


Figure 3.9: Aspect Finder panel.

In the first one, it is possible to load a XML file containing viewpoints and their requirements (figure 3.10). When the "Browse" button is pressed, a file chooser dialog appears, allowing the user to choose only *.XML files. Only then, the "Load" button will become enabled in order to load all the structures with the specific data.

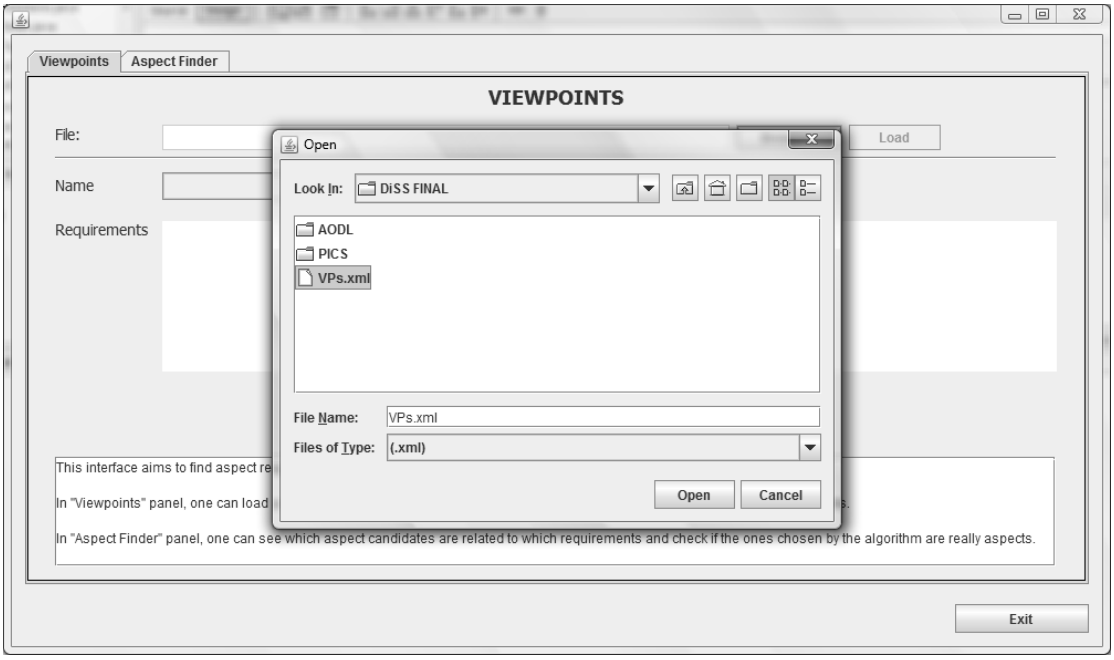


Figure 3.10: Open Dialog Box.

Finally, after all the data was loaded, the Combo Box "Name" displays all the viewpoints names and the Text Area "Requirements" displays the corresponding requirements to the chosen viewpoint (figure 3.11).

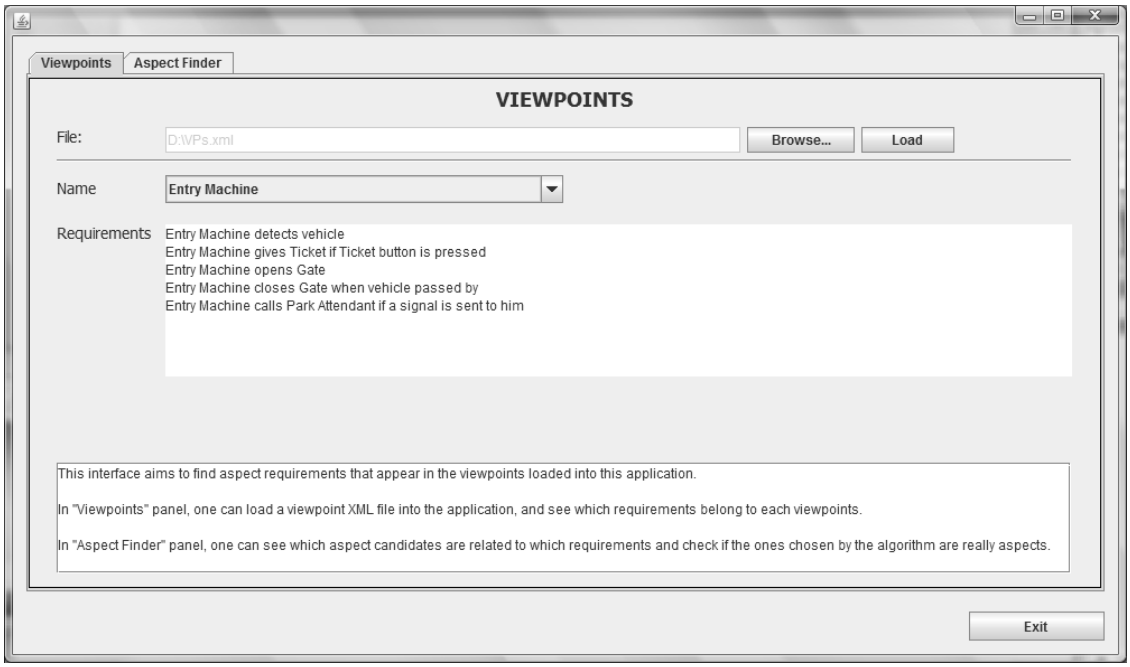


Figure 3.11: File loaded.

In the later one (figure 3.9), from the viewpoints and requirements already loaded in the previous panel, one can find which requirements affect the requirement chosen at the top.

This influence is determined by an algorithm. It was decided that a requirement is an aspectual candidate if there are at least 50% equal words between both requirements.

The Combo Boxs “Viewpoint” and “Requirements” are used to choose one requirement among all. After hitting the “Show Results” buttons the aspectual candidates are displayed in the bottom half of the panel as shown in figures 3.12 and 3.13.

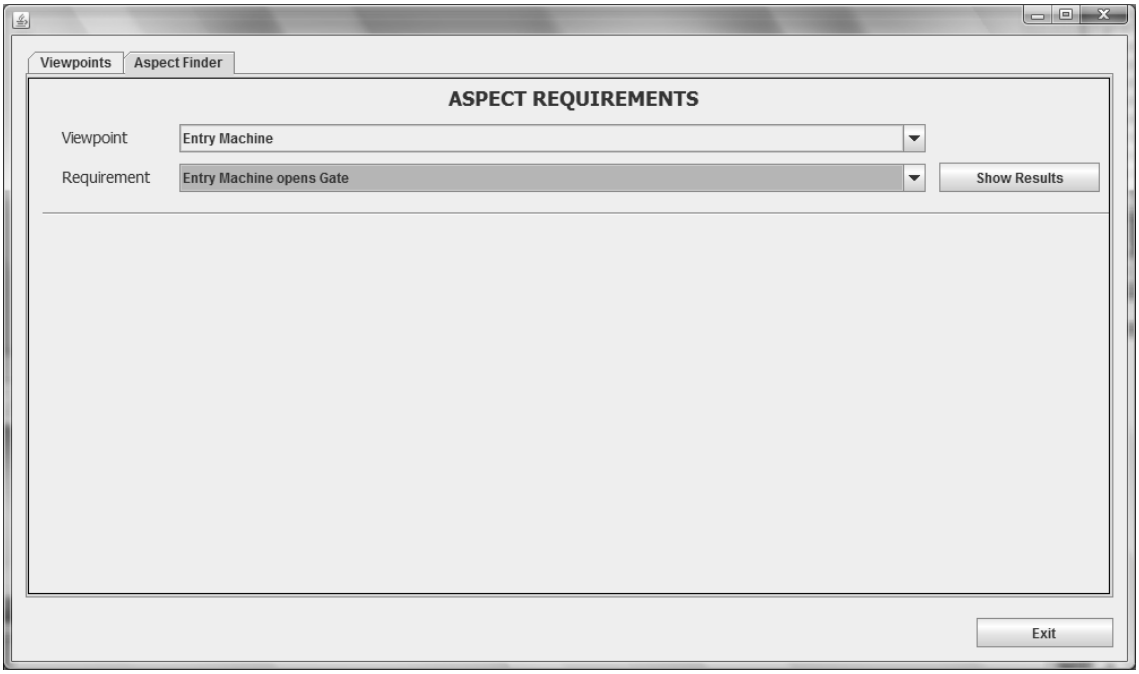


Figure 3.12: Aspect Requirements.

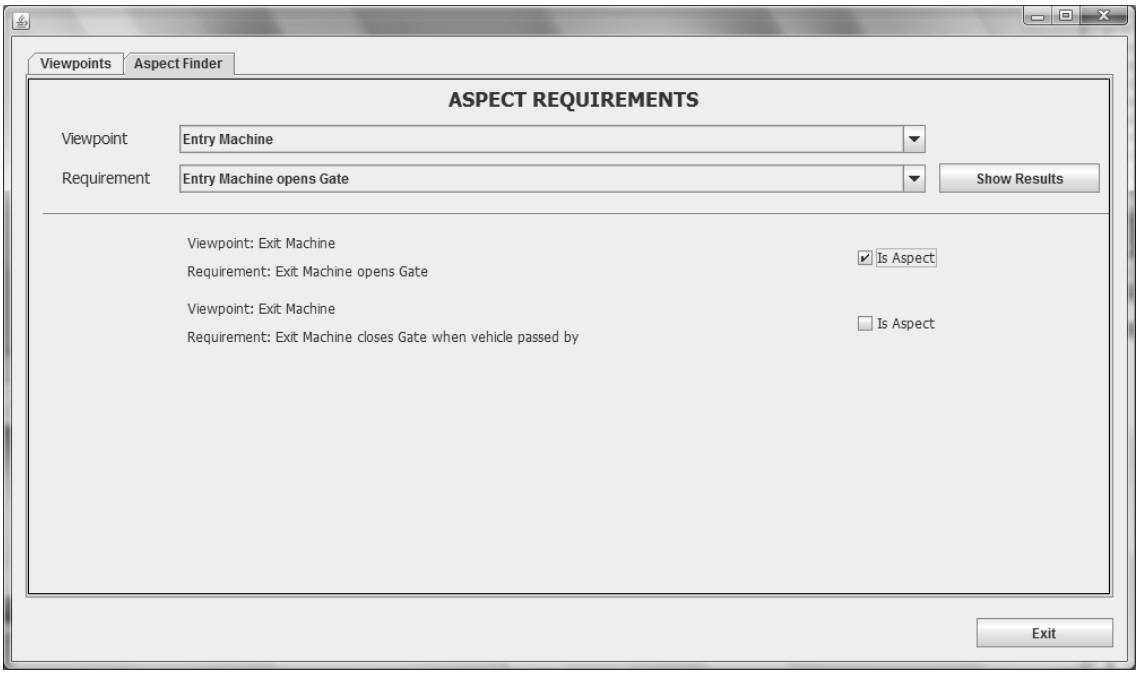


Figure 3.13: Results showed and IsAspect checked.

As the word “candidate” implies, not all requirements are really aspectual requirements. They are just chosen among all the requirements as possible aspectual ones. For that matter, there is a Check Box in front of each pair Viewpoint-Requirement in order to really point out the fact of that requirement being really an aspectual one. In that case, one can manually check that box to confirm that the candidate is really an aspectual requirement.

It was stated above that the relationship between a requirement and its candidates is in both ways, so if one is “checked” the other one is “checked” as well automatically.

All the process can be repeated over and over, just having to return to the “Viewpoint” panel and load a new *.XML file, containing new viewpoints and requirements. The “Exit” button is self-explanatory and is used to exit the program.

This tool is just to help finding aspectual functional requirements among a large list of Viewpoints and requirements. For further works, this tool can be upgraded in order to save the aspectual requirements and the corresponding aspectual viewpoint into the *.XML file.

# Chapter 4
# Case study and comparison with other approaches

In this chapter it will be presented another case study, this time the Health Watcher system, where VAODA will be applied. Following this, a comparison between VAODA and other approaches will be shown, namely versus viewpoint-oriented approaches, aspect-oriented approaches and finally versus domain analysis approaches.

## 4.1 Case Study – Health Watcher

The Health Watcher is described in [40], and consists on a real health complaint system developed to improve quality of the services provided by health care institutions.

This document specifies the requirements for the City Hall Public Health System named HEALTH-WATCHER, providing for developers, the required information for the system

development. The purpose of the system is to collect and control the complaints and notifications, also providing important information to the people about the Health System. With the deployment of HEALTH-WATCHER system, the Public Health System will considerably improve:

- The complaint control (denounces and notifications).
- Quality of service on spreading its information.

The citizen will be able to access the system asking information about the health services or making his/her complaint. Example of possible queries that a citizen can make are, for example: information about diseases (description, symptoms, disease prevention), vaccination campaigns, information about the complaint made by the citizen (e.g. complainer name and data, complaint date, complaint description and observations, situation (opened, suspended, closed), technical analysis, analysis date, employee that made the analysis), which health units take care of a specific specialty, which are the specialties of a health unit.

In the event of a complaint, this will be registered on the system and addressed by a specific department (represented by a registered assistant), which will be able to carry out the procedure and return an answer when the analysis is accomplished. This solution will be registered on the system, being available for queries. The types of complaints are:

- Animal Complaint: Animals apprehension, control of vectors (rodents, scorpions, bats, etc.), diseases related to mosquitoes, animals maltreatment. Additional information needed: Kind of animal; amount of animals; disturb date; disturb location data.
- Food Complaint: Cases where it is suspicious the ingestion of infected food. Additional information needed to provide: victim's name; victim's data; amount of people who ate the food; amount of sick people; amount of people who were sent to a hospital and amount of deceased people; location where the patients were treated; suspicious meal.
- Diverse Complaint: Cases related to several reasons, which are not mentioned above (restaurants with hygiene problems, leaking sewerage, suspicious water transporting trucks, etc.).

The product will also be put to public usage in kiosks in several strategic points, on which the citizen itself will make its complaints and information requests. Also, the new system must allow exchange of information with the SSVS system (Sanitary Surveillance System). Finally,

a report with statistics showing the frequency of kinds of complaints is sent to the director of each health unit, every month.

Next the VAODA approach is going to be applied to this case study.

### 4.1.1 How to approach the problem at the domain engineering level

#### 4.1.1.1 Identify and specify domain viewpoints and concerns

The viewpoints in this case study are: Citizen, PHS Department Assistant, HU Administrator, System Administrator, SSVS System Administrator, HU Director and HW Kiosk (Tables 4.1-4.7).

The concerns considered in this case study are: Security, availability, response time, compatibility, multi-access, and usability (Tables 4.8-4.13).

**Viewpoints**

Table 4.1: Citizen viewpoint

| | |
|---|---|
| Name | Citizen |
| Focus | Request information and make complaints. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment, Internet |
| Requirements | 1. Citizen registers in the system.<br>2. Citizen logs into the system.<br>3. Citizen requests information.<br>3.1. Citizen requests information about diseases.<br>3.2. Citizen requests information about vaccination campaigns.<br>3.3. Citizen requests information about complaints.<br>3.4. Citizen requests information about health units (specialties).<br>3.5. Citizen requests information about specialties (health units).<br>4. Citizen makes a complaint.<br>4.1. Citizen makes an animal complaint.<br>4.2. Citizen makes a diverse complaint.<br>4.3. Citizen makes a food complaint.<br>5. Citizen logs out of the system. |
| Change History | |
| Type | Non-Aspectual |

Table 4.2: PHS Department Assistant viewpoint

| | |
|---|---|
| Name | PHS Department Assistant |
| Focus | Analyze the complaint and return the answer. |
| Concerns | Security, availability, usability. |
| Source | Assignment, Internet |
| Requirements | 1. PHS Department Assistant logs into the system.<br>2. PHS Department Assistant processes a complaint.<br>2.1. PHS Department Assistant analyzes a complaint.<br>2.2. PHS Department Assistant updates a complaint.<br>2.3. PHS Department Assistant returns an answer.<br>2.4. PHS Department Assistant registers a complaint in the HW System for further queries.<br>3. PHS Department Assistant logs out of the system. |
| Change History | |
| Type | Non-Aspectual |

Table 4.3: HU Director viewpoint

| | |
|---|---|
| Name | HU Director |
| Focus | Retrieve information concerning the frequency of kinds of complaints |
| Concerns | Security, compatibility, availability, usability. |
| Source | Assignment, Internet |
| Requirements | 1. HU Director logs into the system.<br>2. HU Director retrieves information (statistics report).<br>3. HU Director logs out of the system. |
| Change History | |
| Type | Non-Aspectual |

Table 4.4: HU Administrator viewpoint

| | |
|---|---|
| Name | HU Administrator |
| Focus | Maintain the information about health units. |
| Concerns | Security, availability, multi-access, compatibility, usability. |
| Source | Assignment, Internet |
| Requirements | 1. HU Administrator logs into the system.<br>2. HU Administrator manages the information about health units.<br>2.1. HU Administrator inserts health unit information.<br>2.2. HU Administrator edits health unit information.<br>2.3. HU Administrator removes health unit information.<br>3. HU Administrator gets information from HW about vaccines.<br>4. HU Administrator logs out of the system. |
| Change History | 1. HU Administrator sends to HW System health unit available campaigns and its dates. |
| Type | Non-Aspectual |

Table 4.5: System Administrator viewpoint

| | |
|---|---|
| Name | System Administrator |
| Focus | Maintain the information about HW System |
| Concerns | Security, availability, usability. |
| Source | Assignment, Internet |
| Requirements | 1. System Administrator logs into the system.<br>2. System Administrator manages the System Information.<br>2.1. System Administrator inserts HW System information.<br>2.2. System Administrator edits HW System information.<br>2.3. System Administrator removes HW System information.<br>3. System Administrator adds a new kind of complaint.<br>4. System Administrator logs out of the system. |
| Change History | |
| Type | Non-Aspectual |

Table 4.6: SSVS System Administrator viewpoint

| | |
|---|---|
| Name | SSVS System Administrator |
| Focus | Exchange information with the HW System |
| Concerns | Security, availability, compatibility, response time, usability. |
| Source | Assignment, Internet |
| Requirements | 1. SSVS System Administrator logs into the system.<br>2. SSVS System Administrator manages information about vaccine campaigns.<br>2.1. SSVS System Administrator sends information about diseases and vaccination campaigns.<br>2.2. SSVS System Administrator gets information about diseases and vaccination campaigns.<br>3. SSVS System Administrator logs out of the system. |
| Change History | |
| Type | Non-Aspectual |

Table 4.7: HW Kiosk viewpoint

| | |
|---|---|
| Name | HW Kiosk |
| Focus | Give information and retrieve complaints. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment, Internet |
| Requirements | 1. HW Kiosk logs into the system.<br>2. HW Kiosk registers a citizen.<br>3. HW Kiosk verifies login.<br>4. HW Kiosk retrieves Information.<br>5. HW Kiosk makes a complaint.<br>6. HW Kiosk logs out of the system. |
| Change History | |
| Type | Non-Aspectual |

**Concerns**

The concerns of the case study are security, availability, response time, multi-access, usability and compatibility specified in tables 4.8-4.13.

Table 4.8: Security concern template

| Concern name | Security |
|---|---|
| Description | 1. HW System must be a secure system adequate to:<br>2. HW System must guarantee communication safety between: |
| Requirements | 1.<br>1.1. Register a username and a password set by the user.<br>1.2. Validate a user log in.<br>1.3. Allow any user to access the system after logging in.<br><br>2.<br>2.1. Users.<br>2.2. Other systems while exchanging data (such as reports). |

Table 4.9: Availability concern template

| Concern name | Availability |
|---|---|
| Description | HW System must be available to permit: |
| Requirements | 1. Users login.<br>2. Users make complaints as well as request several information.<br>3. Other systems administrators add/edit/remove information in runtime.<br>4. Exchange of data with SSVS System. |

Table 4.10: Response Time template

| Concern name | Response Time |
|---|---|
| Description | HW System must react on a very short time to: |
| Requirements | 1. Validate a user log in.<br>2. Generate and show forms for requests in runtime.<br>3. Generate and show forms for complaints in runtime.<br>4. Display warning and error messages in runtime. |

Table 4.11: Multi-Access concern template

| Concern name | Multi-Access |
|---|---|
| Description | HW System must be capable to allow several users to: |
| Requirements | 1. Make requests or complaints.<br>2. Log in/log out simultaneously.<br>3. Register at the same time. |

Table 4.12: Usability concern template

| Concern name | Usability |
|---|---|
| Description | HW System Interface needs to be: |
| Requirements | 1. Very friendly, easy and intuitive system.<br>2. Show help tips when necessary. |

Table 4.13: Compatibility concern template

| Concern name | Compatibility |
|---|---|
| Description | HW System must be synchronized in order to: |
| Requirements | 1. Send statistics report.<br>2. Get information of Health Units for vaccination campaigns.<br>3. Send/receive other systems information. |

#### 4.1.1.2 Specify Feature Model and Aspects

##### 4.1.1.2.1 Specify a feature model

In this section a feature model to the health watcher system is specified (figure 4.1). There are mandatory features (e.g. Complaint report, complaints) and the optional features in this case study are the types of complaint that a citizen can make and the types of information requests.

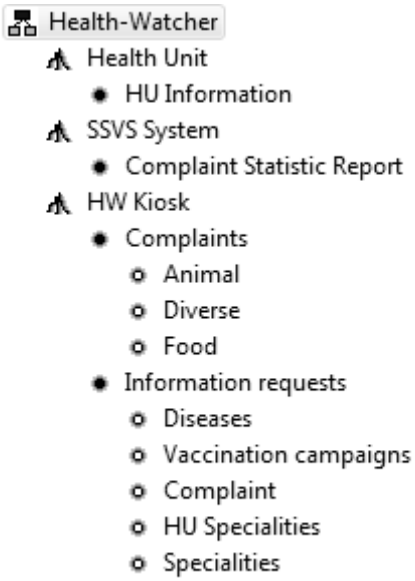


Figure 4.1: Health-Watcher feature model for domain engineering

#### 4.1.1.2.2 Identify crosscutting requirements and specifying and composing aspectual viewpoints

In this case study, all intervenients have to log into the system in order to do their specific actions and when the session is over they have to log out of the system.

So the action of logging in and logging out are crosscutting to the viewpoints.

So |HW User will be instantiated to any user who needs to access the system.

**Specify aspectual viewpoint**

The identified aspectual viewpoint is specified in Table 4.14.

Table 4.14: HW User Login viewpoint

| | |
|---|---|
| Name | HW User Login |
| Focus | Access the system. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment |
| Requirements | AR1. \|HW User logs into the system.<br>AR2. \|HW User logs out of the system. |
| Change History | |
| Type | Aspectual |

Now all viewpoints that had aspectual functional requirements have to be re-written (Tables 4.15-4.21).

Table 4.15: Citizen viewpoint

| | |
|---|---|
| Name | Citizen |
| Focus | Request information and make complaints. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment, Internet |
| Requirements | 1. Citizen registers in the system.<br>2. Citizen requests information.<br>2.1. Citizen requests information about diseases.<br>2.2. Citizen requests information about vaccination campaigns.<br>2.3. Citizen requests information about complaints.<br>2.4. Citizen requests information about health units (specialties).<br>2.5. Citizen requests information about specialties (health units).<br>3. Citizen makes a complaint.<br>3.1. Citizen makes an animal complaint.<br>3.2. Citizen makes a diverse complaint.<br>3.3. Citizen makes a food complaint. |
| Change History | |
| Type | Non-Aspectual |

Table 4.16: PHS Department Assistant viewpoint

| | |
|---|---|
| Name | PHS Department Assistant |
| Focus | Analyze the complaint and return the answer. |
| Concerns | Security, availability. |
| Source | Assignment, Internet |
| Requirements | 1. PHS Department Assistant processes a complaint.<br>1.1. PHS Department Assistant analyzes a complaint.<br>1.2. PHS Department Assistant updates a complaint.<br>1.3. PHS Department Assistant returns an answer.<br>1.4. PHS Department Assistant registers a complaint in the HW System for further queries. |
| Change History | |
| Type | Non-Aspectual |

Table 4.17: HU Director viewpoint

| | |
|---|---|
| Name | HU Director |
| Focus | Retrieve information concerning the frequency of kinds of complaints |
| Concerns | Security, compatibility, availability. |
| Source | Assignment, Internet |
| Requirements | 1. HU Director retrieves information (statistics report). |
| Change History | |
| Type | Non-Aspectual |

Table 4.18: HU Administrator viewpoint

| | |
|---|---|
| Name | HU Administrator |
| Focus | Maintain the information about health units. |
| Concerns | Security, availability, multi-access, compatibility. |
| Source | Assignment, Internet |
| Requirements | 1. HU Administrator manages the information about health units.<br>1.1. HU Administrator inserts health unit information.<br>1.2. HU Administrator edits health unit information.<br>1.3. HU Administrator removes health unit information.<br>2. HU Administrator gets information from HW about vaccines. |
| Change History | 1. HU Administrator sends to HW System health unit available campaigns and its dates. |
| Type | Non-Aspectual |

Table 4.19: System Administrator viewpoint

| | |
|---|---|
| Name | System Administrator |
| Focus | Maintain the information about HW System |
| Concerns | Security, availability |
| Source | Assignment, Internet |
| Requirements | 1. System Administrator manages the System Information.<br>1.1. System Administrator inserts HW System information.<br>1.2. System Administrator edits HW System information.<br>1.3. System Administrator removes HW System information.<br>2. System Administrator adds a new kind of complaint. |
| Change History | |
| Type | Non-Aspectual |

Table 4.20: SSVS System Administrator viewpoint

| | |
|---|---|
| Name | SSVS System Administrator |
| Focus | Exchange information with the HW System |
| Concerns | Security, availability, compatibility, response time. |
| Source | Assignment, Internet |
| Requirements | 1. SSVS System Administrator manages information about vaccine campaigns.<br>1.1. SSVS System Administrator sends information about diseases and vaccination campaigns.<br>1.2. SSVS System Administrator gets information about diseases and vaccination campaigns. |
| Change History | |
| Type | Non-Aspectual |

Table 4.21: HW Kiosk viewpoint

| | |
|---|---|
| Name | HW Kiosk |
| Focus | Give information and retrieve complaints. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment |
| Requirements | 1. HW Kiosk registers a citizen.<br>2. HW Kiosk verifies login.<br>3. HW Kiosk retrieves Information.<br>4. HW Kiosk makes a complaint. |
| Change History | |
| Type | Non-Aspectual |

**Composing aspectual viewpoints**

The composition rules are specified in listings 4.1-4.7. Basically they say which requirements are added to the viewpoints.

```
Compose Citizen with HW User
Bind |HW User to Citizen,
     Add requirement AR1 in Citizen.
     Add requirement AR2 in citizen.
```

Listing 4.1 Composition rules for Citizen in Domain Engineering.

```
Compose PHS Department Assistant with HW User
Bind |HW User to PHS Department Assistant
     Add requirement AR1 in PHS Department Assistant.
     Add requirement AR2 in PHS Department Assistant.
```

Listing 4.2 Composition rules for PHS Department Assistant in Domain Engineering.

```
Compose HU Administrator with HW User
Bind |HW User to HU Administrator
     Add requirement AR1 in HU Administrator.
     Add requirement AR2 in HU Administrator.
```

Listing 4.3 Composition rules for HU Administrator in Domain Engineering.

```
Compose System Administrator with HW User
Bind |HW User to System Administrator,
     Add requirement AR1 in System Administrator.
     Add requirement AR2 in System Administrator.
```

Listing 4.4 Composition rules for System Administrator in Domain Engineering.

```
Compose SSVS System Administrator with HW User
Bind |HW User to SSVS System Administrator,
     Add requirement AR1 in SSVS System Administrator.
     Add requirement AR2 in SSVS System Administrator.
```

Listing 4.5 Composition rules for SSVS System Administrator in Domain Engineering.

```
Compose HU Director with HW User
Bind |HW User to HU Director,
     Add requirement AR1 in HU Director.
     Add requirement AR2 in HU Director.
```

Listing 4.6 Composition rules for HU Director in Domain Engineering.

```
Compose HW Kiosk with HW User
Bind |HW User to HW Kiosk,
     Add requirement AR1 in HW Kiosk.
     Add requirement AR2 in HW Kiosk.
```

Listing 4.7 Composition rules for HW Kiosk in Domain Engineering.

#### 4.1.1.3 Integrate the feature model with the viewpoint model

The integration between the feature model and the viewpoints will be made in a tabular mode showed in table 4.22.

Table 4.22: Feature Model vs Viewpoints table for Domain Engineering Health-Watcher

| F\VP | | | | Citizen | PHS Dep Assistant | HU Administrator | HU Director | System Administrator | SSVS Administrator | HW Kiosk |
|---|---|---|---|---|---|---|---|---|---|---|
| Health Watcher | Health Unit | | HU Information | | | x | | x | | |
| | PHS | PHS Department | PHS Dep. Assistant | | | | | x | | |
| | HW Kiosk | Complaints | Animal | o | | | | o | | o |
| | | | Diverse | o | | | | o | | o |
| | | | Food | o | | | | o | | o |
| | | Information requests | Diseases | o | | | | o | o | o |
| | | | Complaints | o | o | | | o | | o |
| | | | HU Specialities | o | | | | o | | o |
| | | | Specialities | o | | | | o | | o |
| | | | Vaccination campaigns | o | | | | o | o | o |
| | SSVS | | Complaint Statistic Report | | | | x | x | | |

## 4.1.2 How to approach the problem at the application engineering level

### 4.1.2.1 Configure a feature model

In this case, it will be simulated that a citizen will make a food complaint, only. Information requests were not selected either.

In this section a feature model to the health watcher system is specified (figure 4.2). All features are mandatory features.

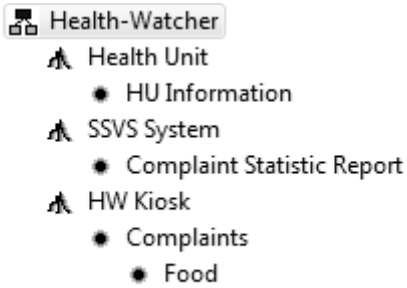


Figure 4.2: Health-Watcher feature model for application engineering

#### 4.1.2.2 Reuse viewpoints according to configured feature model

Since a citizen will make a food complaint, only, there are several differences between the Health Watcher system at the Domain level, and the Health Watcher system at the Application level, as at the domain level all possibilities have to be taken into account, while in application engineering is an example of a possible configuration that the system may have.

The viewpoints in this case study are: Citizen, PHS Department Assistant, HU Administrator, System Administrator, SSVS System Administrator, HU Director and HW Kiosk, but only Citizen, PHS Department Assistant, HU Director and HW Kiosk will be described because are the ones reused in this configuration.

The concerns are the same as the ones described in the domain engineering, so they will not be reused.

**Specify aspectual viewpoint**

Our aspectual viewpoint is the same as specified in the domain, repeated in (Table 4.23) to facilitate the reading.

Table 4.23: HW User Login aspectual viewpoint

| | |
|---|---|
| Name | HW User Login |
| Focus | Access the system. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment |
| Requirements | AR1. \|HW User logs into the system.<br>AR2. \|HW User logs out of the system. |
| Change History | |
| Type | Aspectual |

Now all viewpoints that had aspectual functional requirements have to be re-written to only present what matters to the configuration.

Table 4.24: Citizen viewpoint

| | |
|---|---|
| Name | Citizen |
| Focus | Request information and make complaints. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment, Internet |
| Requirements | 1. Citizen registers in the system.<br>2. Citizen makes a food complaint. |
| Change History | |
| Type | Non-Aspectual |

Table 4.25: PHS Department Assistant viewpoint

| | |
|---|---|
| Name | PHS Department Assistant |
| Focus | Analyze the complaint and return the answer. |
| Concerns | Security, availability. |
| Source | Assignment, Internet |
| Requirements | 1. PHS Department Assistant processes a food complaint.<br>1.1. PHS Department Assistant analyzes a food complaint.<br>1.2. PHS Department Assistant updates a food complaint.<br>1.3. PHS Department Assistant returns an answer.<br>1.4. PHS Department Assistant registers a food complaint in the HW System for further queries. |
| Change History | |
| Type | Non-Aspectual |

Table 4.26: HU Director viewpoint

| | |
|---|---|
| Name | HU Director |
| Focus | Retrieve information concerning the frequency of kinds of complaints |
| Concerns | Security, compatibility, availability. |
| Source | Assignment, Internet |
| Requirements | 1. HU Director retrieves information (statistics report). |
| Change History | |
| Type | Non-Aspectual |

Table 4.27: HW Kiosk

| | |
|---|---|
| Name | HW Kiosk |
| Focus | Give information and retrieve complaints. |
| Concerns | Security, availability, response time, multi-access, usability. |
| Source | Assignment |
| Requirements | 1. HW Kiosk registers a citizen.<br>2. HW Kiosk verifies login.<br>3. HW Kiosk retrieves Information.<br>4. HW Kiosk makes a food complaint. |
| Change History | |
| Type | Non-Aspectual |

All other viewpoints are not reused because of the configuration chosen for this application.

**Concerns**

The concerns are the same as the ones described in the domain engineering, so they will all be reused.

#### 4.1.2.3 Reuse composition rules to configured feature model

The composition rules specified at domain level are reused here (Listings 4.8-4.11).

```
Compose Citizen with HW User
Bind |HW User to Citizen,
     Add requirement AR1 in Citizen.
     Add requirement AR2 in citizen.
```

Listing 4.8 Composition rules for Citizen in Application Engineering.

```
Compose PHS Department Assistant with HW User
Bind |HW User to PHS Department Assistant,
     Add requirement AR1 in PHS Department Assistant.
     Add requirement AR2 in PHS Department Assistant.
```

Listing 4.9 Composition rules for PHS Department Assistant in Application Engineering.

```
Compose HU Director with HW User
Bind |HW User to HU Director,
     Add requirement AR1 in HU Director.
     Add requirement AR2 in HU Director.
```

Listing 4.10 Composition rules for HU Director in Application Engineering.

```
Compose HW Kiosk with HW User
Bind |HW User to HW Kiosk,
     Add requirement AR1 in HW Kiosk.
     Add requirement AR2 in HW Kiosk.
```

Listing 4.11 Composition rules for HW Kiosk in Application Engineering.

#### 4.1.2.4 Simplify the viewpoints vs feature model table

The following table shows the integration of the Feature model vs the Viewpoint model for this specific configuration of the system.

Table 4.28: Feature Model vs Viewpoints table for the application engineering Health-Watcher.

| F\VP | | | | Citizen | PHS Dep Assistant | HU Administrator | HU Director | System Administrator | SSVS Administrator | HW Kiosk |
|---|---|---|---|---|---|---|---|---|---|---|
| Health Watcher | Health Unit | | HU Information | | | x | | x | | |
| | PHS | PHS Department | PHS Dep. Assistant | | | | | x | | |
| | HW Kiosk | Complaints | Food | x | x | | | x | | x |
| | SSVS | | Complaint Statistic Report | | | | x | x | | |

As we can notice one of the main advantages of the approach is the reuse of domain artifacts.

## 4.2 Evaluation – Comparison with other approaches

In this chapter VAODA will be compared and contrasted with several other approaches from different scopes. In order to do so, criterion was applied, and the fields were filled with a set of possible resolutions:

- "Yes", if the criterion was satisfied;
- "No", if the criterion was not satisfied;
- "Not specified", if the criterion is not specified in the document.

The six selected criteria are listed below:

1. "Non functional requirements", specifying if an approach supports non functional requirements;
2. "Modeling crosscutting concerns", evaluating if an approach supports modeling crosscutting concerns;
3. "Tool support", considering the tools available to work with the specified approach;
4. "Feature model", specifying if an approach has feature models associated;
5. "Conflict management", considering if an approach has any kind of conflict management;

6. “Composition mechanisms”, evaluating if an approach has any kind of composition mechanisms.

This criteria was chosen based on the importance that each one represents within viewpoint-oriented, domain analysis and requirement engineering approaches.

Non-functional requirements and modeling crosscutting NFR is important because crosscutting requirements should be found as early as possible in order to minimize changes throughout the software lifecycle. Having a tool to support automation of some processes within the approaches is also a plus. In order to find commonalities and variability within a domain system one needs feature modeling. Conflict identification is important so that agreements between stakeholders can be negotiated, allowing trade-off decisions to be recorded and communicated to the other system developers.

Finally, the ability to compose requirements (crosscutting and non-crosscutting), analyze their interrelationships and obtain a more complete view of the requirement artifacts in order to identify conflicts earlier in the development is of major importance.

### 4.2.1 Applying the criteria

In this chapter we will compare the features of our approach (VAODA) with the ones from other approaches. We will compare VAODA to other viewpoint-oriented approaches, requirement engineering approaches and to domain analysis approaches.

#### 4.2.1.1 VAODA vs Viewpoint-Oriented approaches

As VAODA is also a Viewpoint-Oriented approach which incorporates aspects it makes sense to compare it with viewpoint-oriented approaches that do not support aspects.

Table 4.29 shows a side-by-side comparison of our approach with VORD, Leite and Freeman’s and Nuseibeh, Kramer and Finkelstein’s.

Table 4.29: VAODA vs Viewpoint-Oriented approaches.

| Criterion\Approach | VAODA | VORD | Leite and Freeman | Nuseibeh, Kramer and Finkelstein |
|---|---|---|---|---|
| **Non functional Requirements** | Yes | Yes | No | Yes |
| **Modeling crosscutting concerns** | Yes | No | No | No |
| **Tool Support** | Yes | No | No | Yes |
| **Feature model** | Yes | No | No | No |
| **Conflict management** | No | Yes | Yes | Yes |
| **Composition mechanisms** | Yes | No | No | No |

VAODA, VORD and Nuseibeh et al. all support non functional requirements, whilst Leite and Freeman's does not.

Our approach, VAODA, support non functional requirements.

In order to support non functional requirements, VORD does it on the viewpoint documentation phase. In order to document non functional requirements, the documentation of the viewpoint, includes non functional requirements. Nuseibeh, Kramer and Finkelstein add support for non functional requirements.

As our approach incorporates aspects, we do support cross-cutting concerns. Actually identifying cross-cutting requirements is one step of our approach. All the other approaches, VORD, Leite and Freeman, and Nuseibeh et al. do not support crosscutting concerns, they do not even support it by other means.

By supporting crosscutting concerns we were obliged to incorporate some kind of composition mechanism into our approach, as we did. We were obligated to have composition, as we needed to know "how" and "where" crosscutting concerns affect other concerns on our system. All the other three approaches do not have composition mechanisms.

In order to manage conflicts VORD analyzes conflicting requirements and sets a weight to each one like a priority. Leite and Freeman's approach compares viewpoints through "views". Nuseibeh et al. have conflict management but it is not specified. Our approach does not

incorporate mechanisms to manage conflicts, but one such mechanism is suggested as future work. Nevertheless, it can easily adapt the mechanism provided by ARCADE [23]

VORD manages conflicts by analyzing the conflicting requirements and setting a weight to each one like a priority. Leite and Freeman's approach compares viewpoints through "views". Nuseibeh et al. has a conflict management but is not specified.

Although the name of our approach does not suggest it, our approach is the only one of the four approaches to support feature models.

With all the previous stated features of the approaches still there is one important feature that has not been analyzed. We must analyze what is the tool support for each of the approaches. VAODA (our approach) has a tool named "Aspect Finder"[1] which is used to search for functional crosscutting requirements in a set of viewpoints and the FMP[2] plug-in for eclipse to generate feature models. Nuseibeh et al. provide a tool named "The Viewer". On the other hand, VORD, and Leite and Freeman do not have any tool support.

#### 4.2.1.2 VAODA vs Aspect-Oriented approaches

Being an aspect-oriented technique, we must also compare VAODA with other aspect-oriented approaches.

Table 4.30 shows our approach compared to other aspect-oriented approaches, namely AOCRE, AORE, Theme and AOSD/UC.

[1] Built by the author of this work
[2] http://gsd.uwaterloo.ca/projects/fmp-plugin/download/

Table 4.30: VAODA vs AO approaches.

| Criterion\Approach | VAODA | AOCRE | AORE | Theme | AOSD/UC |
|---|---|---|---|---|---|
| **Non functional Requirements** | Yes | Yes | Yes | Yes | Yes |
| **Modeling crosscutting concerns** | Yes | Yes | Yes | Yes | Yes |
| **Tool Support** | Yes | Yes | Yes | Yes | Yes |
| **Feature model** | Yes | No | No | No | No |
| **Conflict management** | No | No | Yes | No | Yes |
| **Composition mechanisms** | Yes | Yes | No | Yes | Yes |

All the approaches in the previous table do support non functional requirements.

Being aspect oriented approaches they all support modeling crosscutting concerns. AOSD/UC uses Use Case Slices crosscutting the component model. We stated that all approaches supported modeling crosscutting concerns, although AORE never demonstrated its ability to model functional crosscutting concerns.

On the tool support front, we already looked at VAODA. Theme uses the Theme/Doc at the requirements level, and Theme/UML at the design level. For AORE the tool to use is ARCaDe. Serendipity-II is used for AOCRE. AOSD/UC can use any generic UML Case tool

Only VAODA has a feature model associated.

When it comes to conflict management, only AORE and AOSD/UC support it. Neither of the other approaches does support conflict management.

Being Aspect Oriented approaches it is expected that they include composition mechanisms. AORE is the only one who does not include composition mechanism.

#### 4.2.1.3 VAODA vs Domain analysis approaches

In this section we compare our approach with other domain analysis approaches. Our approach has already been compared to approaches based in viewpoints and to requirements engineering approaches. The missing comparison is with other domain analysis approaches.

Table 4.31 shows our approach compared to other domain analysis approaches, namely OODA, CODA, FODA and AODA.

Table 4.31: VAODA vs DA approaches.

| Criterion\Approach | VAODA | AODA | FODA | OODA | CODA |
|---|---|---|---|---|---|
| **Non functional Requirements** | Yes | Yes | No | Yes | Yes |
| **Modeling crosscutting concerns** | Yes | Yes | No | No | No |
| **Tool Support** | Yes | Not Specified | Yes | Yes | Yes |
| **Feature model** | Yes | No | Yes | Yes | No |
| **Conflict management** | No | Not Specified | No | Not Specified | Yes |
| **Composition mechanisms** | Yes | Yes | Yes | Yes | Yes |

Among all the approaches only FODA does not support non functional requirements.

VAODA and AODA are the only ones who support modeling of crosscutting concerns.

When using this approaches tool support only is not available to AODA. FODA can use one of two tools; FMP plug-in or pure::variants[3] both for eclipse. On the other hand, both OODA and CODA can use any generic UML Case Tool.

FODA is a feature model based approach. Both OODA and VAODA also incorporate feature models in themselves. On the other hand, AODA and CODA do not support feature models.

In order to manage conflicts CODA uses four types of resolution mechanisms which case-by-case one is usual than the others. This case-by-case approach is used between context-dependent adaptations. In order to resolve conflicts, it is possible to enumerate all interactions

[3] http://www.pure-systems.com/Variant_Management.49.0.html

and then resolving them with relationships like exclusion, inclusion, etc. FODA, OODA and AODA do not possess conflict resolution mechanisms.

When it comes to composition mechanisms, all of the approaches support them in some way. Both OODA and CODA support feature interaction and as such this can be seen as a composition mechanism, as they specify “how” and “where” a feature influences and interacts with other features in the domain. FODA implements composition rules for features. These rules can be of two types; requires rules or mutually-exclusive-with rules. With these rules, FODA indicates which feature combinations are valid and which are not.

# Chapter 5
# Conclusions

Domain analysis is the process where information used in developing systems is identified, captured and organized with the objective of making it reusable when creating new systems. DA also is a means to build reusable infrastructures to support the specification and implementation of restricted classes of applications.

Some domain requirements can be referenced as crosscutting or aspectual as they “cut across” several components, which can lead to a problem when performing changes in the system because there can be dependencies across it and when one is changed the others may change as well, causing some system malfunctioning.

To overcome this problem, one area of interest and research is standing out within domain analysis - aspect-oriented domain analysis (AODA) - because it addresses the problem of specifying crosscutting properties at the domain analysis level.

This Master’s thesis proposed an approach to handle crosscutting requirements (i.e. candidate aspects) in domain analysis, using the PREview method integrated to a feature model called VAODA (Viewpoints and Aspect-Oriented Domain Analysis).

The final approach is composed of two main parts, the domain and application engineering parts. The Domain Engineering process is subdivided into six activities: Identify and specify viewpoints and concerns; Specify a feature model; Identify crosscutting requirements (i.e candidate aspects); Specify aspectual requirements; Compose the aspectual viewpoints; and lastly integrate aspect-oriented viewpoints with the feature model. The Application Enginering process consists of four steps: Configure Feature Model; Reuse Viewpoints according to configured Feature Model; Reuse composition rules according to configured Feature Model and finally Simplify the Viewpoint vs Feature Model Table. The approach was applied to both the car park and the health-watcher's case studies. A tool was developed, in order to help identifying the aspectual functional requirements within a set of domain viewpoints.

## 5.1 Contributions

The approach described in this dissertation is an approach for domain analysis that integrates viewpoints, aspects and feature modeling. So, it is an approach, whose artifacts are more reusable because the problem of crosscutting concerns is addressed as they are modularized and modeled. A tool support was developed to identify aspectual requirements.

Viewpoints contribute to the completeness of the AODA approach by adding all the different points of view to a domain system. This fact is important as all stakeholders have an important role within a domain system, and it has to be configured taking all their choices into account.

AO techniques have been studied and developed throughout years, namely AOSD, AORE, AORA, but AODA has not been the target of much research, which opened a door to this thesis, and we hope this thesis contributed to the development of AODA.

With the feature modeling addition to AODA, one can study the commonalities and variabilities of a domain system and therefore, be able to configure it to a specific case.

## 5.2 Limitations

The tool support developed for the VAODA approach is limited as it only deals with functional requirements, instead of dealing with both non-functional and functional ones.

A better composition language should have been developed in order to better characterize the relation between the aspectual viewpoints and the remaining ones.

Conflict management and feature interaction should be added in the future to make this approach more complete.

## 5.3 Future work

Our current work lacks some points that should be focused in the future so it could lead to a better approach.

There is an approach similar to ours, AORE/ARCaDe which deals with non-functional requirements, while VAODA deals with functional ones. A good way to improve either approach is to fill in the gaps in each one: Add aspectual functional requirements to AORE and add aspectual NFR to VAODA.

As far as the tool support is concerned, it could be improved in order to accomplish NFR as well. This would be an important step because there is no tool support that addresses both non-functional and functional requirements, which would come in handy.

A much more complete composition language is also a matter to take into account, when dealing with aspectual requirements.

VAODA also should be extended to incorporate conflict management mechanisms, for example, adapting the conflict management mechanisms of ARCADE. Finally feature interaction management should also be contemplated as future work.

# Chapter 6
# References


1. Araujo J, Baniassad E, Clements P, Moreira A, Rashid A, Tekinerdogan B. Early Aspects: The Current Landscape. *Technical Notes, CMU/SEI and Lancaster University* 2005
2. Shlaer S, Mellor SJ. An object-oriented approach to domain analysis. *ACM SIGSOFT Software Engineering Notes* 1989; 14(5): 66-77
3. Desmet B, Vallejos J, Costanza P, De Meuter W, D'Hondt T. Context-Oriented Domain Analysis. *LECTURE NOTES IN COMPUTER SCIENCE* 2007; 4635: 178
4. Kang K, Cohen S, Hess J, Novak W, Peterson AS. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-21, Nov* 1990
5. Sommerville I, Sawyer P. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc. New York, NY, USA, 1997.
6. Ali MJ. Metrics for Requirements Engineering. *UMEA University* 2006
7. Sommerville I. *Software Engineering*. Addison-Wesley, 2006.
8. Lee Y, Zhao W. Domain Requirements Elicitation and Analysis-An Ontology-Based Approach. *ICCS2006, Part IV, LNCS* 2006; 3994: 805-813
9. van Lamsweerde A. Goal-Oriented Requirements Engineering: A Guided Tour. *Proceedings of the 5th IEEE International Symposium on Requirements Engineering* 2001: 249
10. Jacobson I. Object-Oriented Software Engineering - a Use Case Driven Approach. *Addison-Wesley* 1992
11. Finkelstein A, Kramer J, Nuseibeh B, Finkelstein L, Goedicke M. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. *International Journal of Software Engineering and Knowledge Engineering* 1992; 2(1): 31-58
12. Mellon C. Software Engineering Institute http://www.sei.cmu.edu/.
13. Rosenhainer L. Identifying Crosscutting Concerns in Requirements Specifications. *Proceedings of OOPSLA Early Aspects* 2004
14. Bubl F, Balser M. Tracing cross-cutting requirements via context-based constraints. *9 thConference on Software Maintenance and Reengineering, Manchester, Great Britain. IEEE computer, March* 2005

15. Whittle J, Araujo J. Scenario modelling with aspects. *Software, IEE Proceedings-[see also Software Engineering, IEE Proceedings]* 2004; 151(4): 157-171
16. Sommerville I, Sawyer P. Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering* 1997; 3: 101-130
17. Sommerville I, Sawyer P, Viller S. Viewpoints for requirements elicitation: a practical approach. *Requirements Engineering, 1998. Proceedings. 1998 Third International Conference on* 1998: 74-81
18. Kotonya G, Sommerville I. Requirements engineering with viewpoints. *Software Engineering Journal* 1996; 11(1): 5-18
19. Leite J, Freeman PA. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering* 1991; 17(12): 1253-1269
20. Nuseibeh B, Kramer J, Finkelstein A. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering* 1994; 20(10): 760-773
21. Rashid A, Moreira A, Araújo J. Modularisation and composition of aspectual requirements. *Proceedings of the 2nd international conference on Aspect-oriented software development* 2003: 11-20
22. Grundy J. Aspect-Oriented Requirements Engineering for Component-based Software Systems. *4th IEEE International Symposium on Requirements Engineering* 1999: 84-91
23. Rashid A, Sawyer P, Moreira A, Araujo J. Early aspects: a model for aspect-oriented requirements engineering. *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on* 2002: 199-202
24. Clarke S, Baniassad E. *Aspect-Oriented Analysis and Design*. Addison-Wesley Professional, 2005.
25. Jacobson I, Ng PW. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.
26. Prieto-Díaz R. Domain analysis: an introduction. *ACM SIGSOFT Software Engineering Notes* 1990; 15(2): 47-54
27. Riebisch M. Towards a More Precise Definition of Feature Models. *Modeling Variability for Object-Oriented Product Lines: Workshop at the 17th European Conference on Object-Oriented Programming, ECOOP 2003, Darmstadt, Germany, July 21st, 2003* 2003
28. Czarnecki K, Helsen S, Eisenecker U. Staged configuration through specialization and multilevel configuration of feature models. *Software Process Improvement and Practice* 2005; 10(2): 143-169
29. Weiss DM, Lai CTR. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.
30. Czarnecki K, Eisenecker UW. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 2000.
31. Trinidad P, Benavides D, Ruiz-Cortes A. Isolated Features Detection in Feature Models? *CAiSE Short Paper Proceedings* 2006
32. Czarnecki K, Antkiewicz M, Kim CHP, Lau S, Pietroszek K. fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates. *Conference on Object Oriented Programming Systems Languages and Applications* 2005: 200-201
33. Czarnecki K, Kim CHP, Kalleberg KT. Feature models are views on ontologies. *Software Product Line Conference* 2006
34. Cohen S, Northrop LM. Object-Oriented Technology and Domain Analysis. *5th International Conference on Software Reuse* 1998: 86-93
35. Wartik S, Prieto-Diaz R. Criteria for Comparing Reuse-Oriented Domain Analysis Approaches. *International Journal of Software Engineering and Knowledge Engineering* 1992; 2(3): 403-431
36. Gomaa H. *Designing software product lines with UML*. Addison-Wesley Boston, 2004.

37. Silva C, Alencar F, Araújo J, Moreira A, Castro JFB. Tailoring an Aspectual Goal-Oriented Approach to Model Features. *In: The 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08), San Francisco Bay, USA.* 2008
38. Yu Y, do Prado Leite JCS, Lapouchnian A, Mylopoulos J. Configuring features with stakeholder goals. *Proceedings of the 2008 ACM symposium on Applied computing* 2008: 645-649
39. Jayaraman P, Whittle J, Elkhodary AM, Gomaa H. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis. *LECTURE NOTES IN COMPUTER SCIENCE* 2007; 4735: 151
40. Soares S, Laureano E, Borba P. Implementing distribution and persistence aspects with aspectJ. *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* 2002: 174-190