The Open University of Israel
Department of Mathematics and Computer Science

DRAS
Derived Requirements Generation by Actions and States

Thesis submitted on February, 2009
as partial fulfillment of the requirements towards an
M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
David Bar-On

Prepared under the supervision of Dr. Shmuel Tyszberowicz

Table of Contents

1 Introduction 7
2 Requirements Engineering and AORE Overview 12
2.1 Requirements Engineering 12
2.2 Requirements Specifications Sub-Activities 13
2.3 Textual Requirements Specification 14
2.4 Crosscutting Requirements and Derived Requirements 14
2.5 Early Aspects and AORE 16
3 Problem and Solution Overview 18
3.1 The Problem 18
3.2 Crosscutting FRs 18
3.3 The DRAS Methodology - an Overview 22
3.3.1 Implied Actions 22
3.3.2 Entities and Actions 24
3.3.3 Crosscutting and Modes 25
3.3.4 Action Modifiers 26
3.3.5 Requirements Priorities 27
3.3.6 Contribution and Composition Rules 29
3.4 Putting all together – the DRAS Outline 30
4 The TETRA MS Example 32
4.1 TETRA Overview 32
4.2 TETRA MS Features and Functionality 33
4.3 Simplified Requirements Set 35
4.4 Baseline Requirements (Stakeholders' Requirements) 37
4.4.1 Baseline Requirements - Attributes and Facts 38
4.4.2 Baseline Requirements – System Related 39
4.4.3 Baseline Requirements – Group Call 39
4.5 Crosscutting (Aspectual) Requirements 40
4.5.1 Aspectual Requirements - Emergency Mode 40
4.5.2 Aspectual Requirements – TXI Mode 40
4.5.3 Aspectual Requirements – System Related 40
4.6 Derived Requirements from Baseline and Aspectual Requirements 41
4.6.1 Out-of-Coverage related Derived Requirements 41
4.6.2 Registration related Derived Requirements 42
5 Related Work 43
5.1 Viewpoints 43
5.2 Goal Oriented Requirements Analysis 44
5.3 Modularization and Composition of Aspectual Requirements 44
5.4 Composition Process for Aspect Oriented Requirements (AOR) 45
5.5 Adaptation of the NFR Framework to AORE 45
5.6 Crosscutting Quality Attributes 46
5.7 Theme and Theme/Doc - Finding Aspects in Requirements 46
5.8 Mining Aspects 47
5.9 Other Methods 48
6 Deeper Evaluation of Some AORE Methods 49

6.1 Modularization and Composition of Aspectual Requirements (MCAR) 49
6.1.1 Overview 49
6.1.2 Input Requirements Analysis using MCAR 52
6.1.2.1 Identify and Specify Stakeholders' Requirements 52
6.1.2.2 Identify and Specify Concerns 52
6.1.2.3 Identify the Coarse-grained Concern/Viewpoint Relationship 52
6.1.2.4 Identify Candidate Aspects 54
6.1.2.5 Handle Conflicts between Candidate Aspects 54
6.1.2.6 Compose the Aspects and Requirements 55
6.1.3 Applicability of MCAR for creating Derived Requirements 55
6.2 Composition Process for Aspect Oriented Requirements (AOR) 55
6.2.1 Overview 55
6.2.2 Composition Process for AOR Main Activities 56
6.2.2.1 Identify Concerns 56
6.2.2.2 Specify Concerns and Identify Candidate Aspects 57
6.2.2.3 Compose Candidate-Aspects with Concerns 57
6.2.3 Input Requirements Analysis using Composition Process for AOR 58
6.2.3.1 Identify Concerns 59
6.2.3.2 Specify Concerns and Identify Candidate Aspects 59
6.2.3.3 Compose Candidate-Aspects with Concerns 61
6.2.3.3.1 Identify how each candidate aspect affects the concerns it cuts 61
6.2.3.3.2 Identify Match-Points 62
6.2.3.3.3 Identify Conflicts between candidate aspects 63
6.2.3.3.4 Identify the Dominant Aspect based on "Priority" 63
6.2.3.3.5 Identify Composition Rules 63
6.2.4 Applicability of Composition Process for AOR to create the Derived Requirements 66
6.3 Theme and Theme/Doc - Finding Aspects in Requirements 66
6.3.1 Overview 66
6.3.2 Theme/Doc Approach Major Steps 67
6.3.3 Input Requirements Analysis using Theme and Theme/Doc 69
6.3.3.1 Identifying Actions and Entities 69
6.3.3.1.1 Identifying Actions and Entities per requirement 69
6.3.3.1.2 Actions Identified 71
6.3.3.1.3 Entities Identified 72
6.3.3.2 Create Actions Views 73
6.3.3.2.1 Actions View (Theme/Doc) – Actions by Requirements 73
6.3.4 Applicability of Theme/Doc for creating a Derived Requirement 75
7 The DRAS Methodology 76
7.1 Gathering the Stakeholders' Requirements 76
7.2 Identifying Actions, Entities and Attributes 76
7.2.1 General Lists for all Systems 77
7.2.2 General Lists with Specific System Contents 77
7.3 Identifying Correlations between Actions and Entities 81
7.3.1 Entities used by Action 81
7.3.2 Actions Directly Implied (used) by Action 82

7.3.3 Actions used by Action 83
7.4 Identifying Actions and Entities used by the Input Requirements and their Priorities 84
7.4.1 Split Requirements Text per Action/Entity (Manual) 86
7.4.2 Attributes (Modes and States) of Requirements Parts 86
7.4.3 Requirements Priorities 88
7.5 Identifying Actions used by the Requirements 88
7.6 Identifying Requirements-Actions Attributes 93
7.7 Identifying Match-Points between the Requirements 94
7.7.1 List of Match-Point Candidates between Requirements 96
7.7.2 Remove Redundant Entries 99
7.7.3 Remove Impossible or same Mode/State Match-Points 99
7.7.4 The Final Match-Point Candidates 100
7.8 Evaluating Match-Points 102
7.9 Generating the Derived Requirements 106
8 Summary and Conclusions 110
9 Future Work 112
10 References 116

List of Figures

Figure-1 The MS platform and buttons used by this work 36
Figure-2 Action View for a Subset of Requirements 74

List of Tables

Table 1 Correlation between Base and Crosscut Requirements 53
Table 2 Correlation between the Crosscut Requirements 54
Table 3 Requirements Attributes 60
Table 4 Requirements Attributes and Prioritization 62
Table 5 Derived Requirements 64
Table 6 Actions List 79
Table 7 Modes and States List 80
Table 8 Modes and States Values 80
Table 9 Contradicting Pairs of Mode/State Values 81
Table 10 Entities used by Actions 82
Table 11 Actions Implied by Action 83
Table 12 Actions used by Action Summary 84
Table 13 Input Requirements Split - use of Actions and Entities 85
Table 14 Requirements Attributes of Actions and Entities 86
Table 15 Entities used by Requirements 88
Table 16 Actions per Entity 89
Table 17 Actions Directly used by the Requirements (excerpt) 90
Table 18 Actions Used By Requirements (excerpt) 92
Table 19 Requirement-Actions Attributes (excerpt) 94
Table 20 Candidate Match-Points 98
Table 21 Removed Redundant Candidate Match-Points 99
Table 22 Removed Impossible Match-Point 100
Table 23 Requirements Match-Points 101
Table 24 Match-Points Evaluation 103

Abbreviations

AOP	Aspect Oriented Programming
AORE	Aspect Oriented Requirements Engineering
AOSD	Aspect Oriented Software Design
DR	Derived Requirement
DRAS	Derived Requirements generation by Actions and States (the method defined in this work)
EA	Early Aspects
FR	Functional Requirements
GORA	Goal Oriented Requirements Analysis
MS	Mobile System (GSM/TETRA term for e.g. Cellular Phone)
NFR	Non-Functional requirements
PTT	Push-to-Talk
QA	Quality Attribute
RE	Requirements Engineering
Rx	Receive
SoC	Separation of Concerns
TETRA	Trans European Trunked Radio
Tx	Transmit
TXI	Transmit (Tx) Inhibit

Abstract

When specifying system requirements, many interdependencies may exist between the requirements. Requirements may conflict with one another and they may impact (change, enhance, enhance or override) other requirements as well. In order to avoid the cost and schedule overheads, these interactions and conflicts should be resolved as early as possible in the development process. One method to resolve such interactions and conflicts is to define Derived Requirements (DRs), representing new or modified requirements that are inferred from other requirements.

An important category of requirements are the Functional Requirements (FRs), representing requirements that change or override the function of other requirements they crosscut. This work presents the DRAS (Derived Requirements generation by Actions and States) methodology that helps both to identify FRs that crosscut other FRs and to generate the DRs. To identify crosscutting requirements, the methodology matches the actions used by the requirements and the system modes and states related to these requirements. DRAS is based on the observation that when the same action is used by two requirements, in a similar state of the system, it indicates that one of the requirements may crosscut the other.

In addition to considering the actions used directly by the requirements, DRAS also takes into account the actions implied (activated as a result of) activating these actions, or the actions that imply the use of the actions directly used by the requirements. Whether the implied or implying actions are considered, depends on whether the requirements restrict the use of an action or eases restrictions for its use.

The DRAS input and output are textual specifications and the output is generated during the requirements specification phase of the software development lifecycle. This enables all stakeholders, with or without a technical background, to participate in the process.

1 Introduction

System and product requirements often contain crosscutting requirements, i.e., requirements which interact with each other. Interacting requirements may conflict with one another and they may impact other requirements as well. Crosscutting between requirements usually mean that either existing requirements must be enhanced (changed), or new requirements must be written. Crosscutting requirements influence the selection and definition of system requirements and eventually limit the various architectural choices. It is very important to be able to identify crosscutting requirements as soon as possible in the software development process and to handle them properly. While identifying and handling crosscutting requirements, both functional-requirements (FRs) and non-functional requirements (NFRs) should be considered. A rigorous analysis and understanding of crosscutting requirements and their interactions are essential to derive a balanced architecture. Ignoring interactions between crosscutting requirements results in an incomplete understanding of specified requirements and, consequently, poorly informed architectural choices.

A common resolution to the conflicts or interactions of crosscutting requirements is Derived Requirements (DRs). These are requirements that are inferred, or derived, from other user requirements. They are the outcome of resolving interactions and conflicts between requirements. DRs may be either new requirements or changes (enhancements) to existing requirements. Note that in the context of this work, derived requirements are the result of two or more crosscutting requirements; not expanding and detailing a requirement.

It is very difficult to identify crosscutting requirements in large systems; consequently, methods and tools that can identify crosscutting requirements and define the outcome DRs are needed. If the crosscutting requirements are not identified early enough, for example during requirements analysis, the result is major overhead work during later development phases. This overhead is the effort required to change the system to adhere to the conclusions resulting from the interactions between the requirements. Sometimes system redesign may be needed.

This work presents a methodology to generate textual DRs from stakeholders' textual requirements. The methodology is called *DRAS - Derived Requirements generation by Actions and States*.

The DRAS method enables the generation of textual derived requirements from stakeholders' requirements. It mainly handles specific types of crosscutting requirements, namely crosscutting Functional Requirements (FRs), which may crosscut other FRs. The consequence is that they may change, enhance, or override other requirements they crosscut. For example, a requirement to "not open a window when the outside temperature is below 10 degrees" may crosscut the requirement to "open the window in the morning". The former requirement tentatively crosscuts all requirements related to the action "opening a window", when the temperature decreases below 10 degrees. The outcome of the analysis (for this interaction) should include a decision whether or not to open the window in the morning, when the temperature is below 10 degrees.

A common term used to identify crosscutting between entities and the way they are handled is the *Aspect*. Sousa et-al define an aspect as "an abstraction that encapsulates the specification of a crosscutting concern, and where the match-points and the composition rules for the crosscutting concern are defined" [Sousa 03a]. That is, an Aspect is an abstraction of crosscutting requirements that identifies the crossing points (the match-points) and defines what to do at these points (the composition rules).

Some of DRAS includes ideas that have been adopted from existing methods, especially from [Baniassad 04b, Rashid 03, Brito 03]. DRAS uses actions as the primary means for identifying match-points between FRs, i.e., identifying crosscutting FRs and the requirements they crosscut. This is similar to [Baniassad 04b]. Actions are the functions specified by the FRs. In the example above, "open window" is the action used by both requirements. Note that using the same action by both requirements indicates that one of them may crosscut the other.

DRAS takes into account the actions directly used by a requirement, and the actions they imply (trigger), or the actions implied by their use. That is, for a specific action *Act* used by a requirement, DRAS uses:
(a) the actions that their use is the consequence (result) of using *Act*, as *implied* actions, or (b) the actions that *imply* the use of *Act*. Whether the implying actions or the implied actions are used, depends on:

- Implying actions are used when the crosscutting requirement restricts the use of the action.
- Implied actions are used when the crosscutting requirement eases other restrictions for using the action.

For example, assume the action "to refresh the room" requires (and therefore implies) the action "open the window". In this case, if the window should not be opened, then the implying action "refresh room" is also forbidden, i.e. the room should not be refreshed. On the other hand, "the room should be refreshed" implies that the "open the window" action is required.

DRAS observes that the *modes* or *states* of the system (when an action is activated), also determine whether one FR crosses the other. For example, there may be different requirements for opening the window in the summer or in the winter. In this case, requirements that are only relevant for the summer, when the system is in *Summer* mode, usually do not crosscut with requirements that are relevant only for the winter.

The initial requirements for a system or product are usually textual, because the input from stakeholders is usually verbal or textual. The requirements specifications are transformed into a more formal, technical representation (such as UML diagrams [Fowler 03]) only later in the development process. When the data is formally represented, it is easier to identify crosscutting requirements and the requirements they crosscut. However, non-technical stakeholders, such as customers and marketing representatives, usually are not trained to read formal specifications. Therefore, it is advantageous to be able to generate DRs in a textual form and to integrate them with other requirements.

This enables non-technical stakeholders to review and understand the specifications; thus, it was decided to create textual requirements as the output of DRAS.
In order to generate textual DRs from stakeholders' requirements, DRAS first identifies *match-points* [Brito 03] between requirements. A *Match-point* in requirements is a part in them that identify a tentative crosscutting between the requirements (e.g. a common action). This is performed by identifying common actions that are used by the requirements (inspired by [Baniassad 04b]), and by identifying common system modes and states (when these actions are used). The DRs are then created based on the crosscutting requirements. This enables review and evaluation by both technical and non-technical stakeholders.

As an example, following is a simple set of requirements for initiating a call from a cellular system:

R1. When a phone user dials a number, the phone shall initiate a call to the dialed number.

R2. The phone shall allow initiating calls to the police (911 in the US, 112 in Europe, 100 in Israel) under any condition.

R3. The phone shall be allowed to initiate calls and receive calls, only after checking that the user is allowed to use it (bills paid, phone not stolen, etc.).

The first observation is that all of these requirements are FRs, and the action "call initiation" is mentioned in each of them. Therefore, these requirements may crosscut each other. Further analysis reveals that *R2* and *R3* are tentatively crosscutting, because they restrict or ease the restriction for initiating a call. Analyzing each pair of requirements shows that *R3* crosscuts *R1*, because *R3* restricts the specifications in *R1*. Assuming that *R3* has a higher priority than *R1* (e.g., the requirement *R3* has precedence over *R1*), then the result is an enhancement (change) to *R1*. This enhanced derived requirement may be:

R4. When a user of a phone dials a number, the phone shall initiate a call to the dialed number only if the user is legitimate.

In this case, *R3* may be redundant, as *R4* includes its requirements (and crosscut requirements that *R3* crosscuts. It may still be important to keep such requirement, as

usually not all the requirements it crosscuts are identified in the early stages of development. If new requirements it crosscuts will be added later, *R3* may be important for the resolution process.

Another derived requirement is the result of *R2* and *R3* crosscutting each other. That requirement should either allow or disallow illegitimate users to dial the police. A common solution in cellular systems gives *R2* higher priority and therefore allows illegitimate users to dial the police:

R5. Illegitimate user should be allowed to dial the police.

Alternatively, this can be an enhancement to *R4*:

R4 (enhanced). The phone should not allow dialing by an illegitimate user, unless the user dials the police.

The rest of the thesis is organized as follows:

In Chapter 2 provides a general overview of Aspect Oriented Requirements Engineering (AORE), which is the main area of this work. Chapter 3 discusses the problem of generating DRs and outlines how DRAS handles this issue. Chapter 4 defines a set of requirements that are used later throughout the document to evaluate the different methods, including the new methods suggested in this work. The requirements are a very simple set of requirements for a TETRA Mobile Station. Chapter 5 describes different related existing AORE methods and evaluates them for the ability to help generate derived requirements from the set of requirements defined. Chapter 6 further details and evaluates the methods that were found to be most applicable. The evaluation is performed using the same set of requirements that were defined in chapter 4. Chapter 7 is the main chapter of this work. It defines the DRAS methodology to generate the derived requirements semi-automatically, using a prototype tool. The process and algorithms of DRAS are described and the generation of the derived requirements is demonstrated. Chapter 8 summarizes the work and its conclusions, and suggests items for further research to enhance DRAS, including possible integration with requirements management tools.

2 Requirements Engineering and AORE Overview

This chapter gives an overview of requirements engineering with specific focus on Aspect Oriented Requirements Engineering (AORE). This is to allow better understanding of the place of the DRAS methodology in the development lifecycle, as this work deals with the requirements engineering phase of the development cycle.

2.1 Requirements Engineering

Requirements Engineering (RE) methods handle the requirements specifications phase. The RE methods target the following issues:

1) How to gather the requirements and needs from stakeholders.
2) How to verify that the requirements are well understood.
3) How to specify the requirements, in such a way that they will be well understood by the engineers developing the system.

For a detailed description related to Requirements Engineering (RE) methods, one may refer to [Young 04; Kovitz 99].

This work discusses requirements specifications activities (but does not describe how to write good requirements). RE is a major part of these activities, as RE methods handles specifying the functionality of the system. It is crucial that the output of these activities match stakeholders' needs. Otherwise, the system's usability may be sub-optimal (in relatively good cases) and unusable (in the worst case). Bad requirements specifications also lead to over-budgeted projects, because of the large number of changes needed during development after identifying the problems in the requirements.

Note that some development methods assume that requirements cannot be specified well enough during early development (e.g. Agile and Spiral development methods). Therefore, such methods allow for requirements changes during development. However, even when these methods are used, the basic stakeholder requirements and needs still need to be well understood early. In addition, RE methods can usually be applied

thought the development lifecycle, as details and priorities may be specified later during development.

Getting a good understanding of the basic stakeholders' requirements and needs is not a simple task. [Kovitz 99] states that because software development is difficult, *exploratory engineering* should be performed to identify the right requirements and solution. There are many issues involved, mainly because it is difficult to bridge the gap between stakeholders' descriptions and the formal specifications (for the requirements). Consequently, the Requirements Specifications for RE is divided into sub-activities; they help create the bridge between these two types of specifications.

2.2 Requirements Specifications Sub-Activities

Several methods are used to identify requirements specifications. However, however, they all have common activities, as they all have a common output - the requirements specifications. Some of these activities that are common to many methods are (see [Creveling 03] for details):

- **Voice of the Customer (VOC)**: customers' needs are gathered by interviewing the customers.
- **Grouping related needs**: Similar needs by customers are grouped together. During this process, the initial priority for each group of requirements is set.
- **Customers' Validation and Prioritization**: The high level requirements are returned to the customers, in order to verify that they correctly express customer needs. Also, the customers can validate the prioritization made for the requirements.
- **Mapping to Technical requirements**: The customers' requirements are mapped into technical requirements.
- **Concept Analysis**: The appropriate concept for the solution, required by the customers, is specified.
- **Requirements Specification and Writing**: In this activity, the system requirements are specified and written. This activity is detailed below, describing how this work relates to this activity in Requirements Engineering (RE).

2.3 Textual Requirements Specification

A common way to specify the system requirements is textually, were each requirement is specified separately and is identified by a specific tag. For example, the following is a simple set of textual requirements for initiating a call from a cellular phone:

R1. When a phone's user dials a number, the phone shall initiate a call to the dialed number.

R2. The phone shall allow initiating calls to the police (911 in the US, 112 in Europe, 100 in Israel) under any condition.

R3. The phone shall be allowed to initiate and receive calls, only after checking that the user is allowed to use it (bills are paid, phone is not stolen, etc.).

In this example, the *Rx* (x=1,2,3) are the tags of the specific requirements.

The requirements in the specifications are generally split into two types:

- **Functional Requirements (FRs)** – These requirements define the system functionality, as required by the stakeholder. In the above example, *R1* and *R2* are FRs.
- **Non-Functional Requirements (NFRs)** – In general, these requirements enable proper system functionality by defining requirements such as: security, availability, reliability, etc. These NFR types are also called "ilities" (see [Young 04]). In the above example, *R3* is an NFR.

The DRAS methodology mainly handles FRs.

2.4 Crosscutting Requirements and Derived Requirements

During requirements specification, a major issue is the interactions, dependencies, and conflicts that usually exist between requirements. This is due to both inherent dependencies between requirements, and different requirement types. Such requirements are called *Crosscutting Requirements*. The main purpose of DRAS methodology is to help handle crosscutting requirements.

There are several issues with crosscutting requirements:

1) In large systems, it is difficult to identify the relations between requirements
2) In cases of contradiction, it is sometimes difficult to resolve the conflicts.
3) New requirements should be specified because of crosscutting requirements, etc.

The main *types of dependencies* between requirements are:

- **Enhanced functionality** by dependencies between requirements.
  In the requirements set for the example above (section 2.3), requirements *R1* and *R3* are dependent and a new requirement is derived:

  R4. Dialing should not be allowed by an illegitimate user.

  Instead of creating new requirements, *R1* may be enhanced:

  R1. When a phone's user dials a number, the phone shall initiate a call to the dialed number only if the user is legitimate.

  Although this requirement directly results from other requirements, note that in many cases (to verify that they are implemented), it is important to define them specifically. Also, there are different interpretations for dependencies between requirements. For example, in this case the user is allowed to dial a number, so that it will be possible to initiate a call to the police.
- **Conflicts** between requirements. Requirements 2 and 3 conflict when an illegitimate user tries to dial to the police. The common resolution for this conflict is to allow illegitimate users to dial the police (and with no charge). This result can come from either extending Requirement 3, or by defining a new requirement:

  R5. Illegitimate user should be allowed to dial the police.

The requirements (generated because of these two types of dependencies between requirements) are the main issue of this work. These requirements are called *Derived Requirements*. Functional and non-functional requirements come mostly from stakeholders and from other systems that the system connects to. Usually, derived requirements are written without using any special methods or tools, after thorough

analysis of other requirements. Defining these requirements is often problematic, because defining them requires a very deep understanding of the system and the correlation between many requirements. In many cases, several of these requirements are not defined ahead. Sometimes they are understood only after issues are found during testing, or when the system is already in use. This can cause several issues and defects during system development and system use.

2.5 Early Aspects and AORE

The term *early aspects* refers to aspect-oriented methods that are used during early phases of the development lifecycle. In the requirements specification phase, aspects are the actions and activities that are repeated in different requirements; or they are the cause requirements to "crosscut" each other (i.e., the crosscutting-requirements). Aspects are the interacting parts between requirements, their dependencies, and conflicts. Identifying aspects enables the proper handling of these dependencies and conflicts between requirements. In general, aspects handling enables the generation of additional requirements; the derived requirements resolve the conflicts and add information needed to handle the dependencies. In addition, Aspects handling promotes a better understanding of the system; this helps to later improve system analysis and software design.

The primary purpose of early aspects methods is to find ways for identifying crosscutting concerns from stakeholders' requirements, and to properly compose them with a set of system requirements. Because gathering and specifying system requirements requires a high degree of human (stakeholders) involvement, more than just formal methods are needed. Therefore, tools that were developed to support Early Aspects methods usually do not implement the full process; normally they are used only to assist in the process. Aspect Oriented Requirements Engineering (AORE) deals with aspect oriented methods for Requirements Specification. Some AORE methods were developed before aspect oriented methodology was established (e.g. Goal Oriented Requirements Engineering - see Chapter 5). Others were developed based on AOP methods; they try to extend the use of their techniques to earlier development phases. AORE methods are mainly used

for handling crosscutting requirements, for cases where there are dependencies and conflicts between different requirements.

Several AORE methods are reviewed in Chapter **Error! Reference source not found.**, for their applicability to DRAS methodology developed in this work. An extensive review of Early Aspects and AORE methods can be found in [Chitchyan 05; Araujo 05].

3 Problem and Solution Overview

This chapter explains in more details the issues of handling crosscutting requirements to generate derived requirements and the main approaches to solve these issues that are included in the DRAS methodology. The full description of the methodology is provided in Chapter 7.

3.1 The Problem

The DRAS methodology described in this work handles requirement that are specified textually. Some of the requirements specified for a system may crosscut each other (as explained in Chapter 1, *crosscutting requirements* are requirements in a system specification that interact with each other). Therefore, it is very important to be able to identify crosscutting requirements as soon as possible, to allow generating the proper derived requirements (DRs). While identifying and handling crosscutting requirements, both functional and non-functional requirements should be included. It is also important that the output of that analysis (performed during the requirements elicitation phase) be textual, enabling non-technical stakeholders to review and understand the output.

DRAS is designed to handle these issues for functional requirements. As described later in Chapter **Error! Reference source not found.**, most of the existing methods handle crosscutting NFRs. However, it is important to be able to also treat crosscutting FRs. The DRAS methodology intends to solve this problem by identifying crosscutting functional requirements (along with determining how to handle them), and by specifying the DRs. Both its input and output requirements are textual.

3.2 Crosscutting FRs

This section gives additional and more detailed examples of crosscutting requirements and the crosscutting analysis. The ideas presented in these examples are the basis for the DRAS methodology.

One way to identify crosscutting FRs is according to the *actions* used by the requirements. The Push-to-Talk (PTT) action used in cellular systems will be used as an example.

PTT is used to initiate calls to a pre-selected user or target number in walky-talkies, by pressing a button, also called PTT. As in walky-talkies, these calls are half-duplex, and only one participant can transmit voice at a given time. See chapter 4 for more information regarding the PTT mechanism.

Following are two functional requirements (the crosscutting actions appear in **bold**):

*R1 When PTT is pressed, the phone shall initiate voice **transmission**.*

*R2 When another phone **transmits**, the phone shall not initiate voice **transmission**.*

Since both requirements are about *transmission*, one of them may crosscut the other. In cellular systems, when PTT is used to initiate a call, usually *R2* crosscuts *R1*. That is, a phone will not try to transmit if another phone already transmits. Therefore, the crosscutting resolution may be as follows [the E in *R1(E*) means enhanced]:

*R1(E) When PTT is pressed, the phone shall initiate voice **transmission** unless another phone transmits.*

Note that with *R1(E), R2* may be redundant. However, it is important to keep such crosscutting requirements. Usually not all requirements they crosscut are identified in the early stages of development; new requirements they crosscut may be added later.

Certain issues were identified in the way existing methods use actions to indicate a tentative crosscutting of FRs:

- Actions that are **implied** by the actions directly used by the requirements are not taken into account.
- **Crosscutting-modes** and states are not considered.
- **Action-modifiers** to restrictions are not considered.

1. Implied Actions

In many cases, the use of an action *Act* by a requirement implies the use of other actions by that requirement. These are the actions which are the consequence of using *Act*. For example, the action "pressing the dial button on the phone" implies the use of the action "Transmitting Voice". In addition, actions that imply the use of *Act* may also be relevant to the requirement. For example, when analyzing a requirement about "Transmitting Voice", the action "pressing the dial button" may also have to be considered.

It should be decided which actions to consider: those that are implied by the action *Act*, or those that imply the use of *Act*. This decision depends on whether the requirement restricts the use of *Act* or whether it eases restrictions for the use of *Act*. Restricting the use of *Act* means that all actions that imply its use should also be restricted. Ease of restrictions for the use of *Act* means that all the actions that are implied by it should also be allowed.

For example, a case were an action (transmit) is restricted and therefore an action that implies transmit (initiate a call) is also restricted:

*R3 When **another phone transmits**, the phone shall not initiate voice **transmission**.*

*R4 When **PTT is pressed**, the phone shall **initiate a call**.*

In this case, since initiating a call requires the phone to transmit, a phone should not try to initiate a call if another phone is already transmitting. Note that this deduction requires knowing that initiating a call results in a transmission.

2. Crosscutting and Modes

Modes (or states) of the different entities in the system are also important for determining whether requirements crosscut. Examples for modes of a cellular phone are:

a. Whether it is in a call,

b. Whether the user is in the process of dialing a number, or

c. Whether the user reads SMS messages.

For example:

*R5 In **Call mode** (i.e. during a call), when another phone **transmits**, the phone shall not initiate voice **transmission**.*

*R6 In **Idle mode** (i.e. while not in a call), when PTT is pressed, the phone shall **initiate a call**.*

Although both requirements imply the use of "transmit," none of them crosscuts the other because the modes are orthogonal (mutually exclusive). However, *R7* below crosscuts *R8*, because both requirements are related to the *Call* mode:

*R7 In **Call mode**, when another phone **transmits**, the phone shall not initiate voice **transmission**.*

*R8 When **PTT is pressed**, the phone shall initiate voice **transmission**.*

Note that *R8* does not refer to any specific mode; thus, it is considered to be relevant to all modes, including both *Call* and *Idle* modes. Therefore *R7*, which explicitly refers to the *Call* mode, crosscuts *R8*.

However, especially in systems with many kinds of modes, analysts tend not to explicitly mention in the requirements the mode they refer to; otherwise, the requirements would be very long and difficult to understand. Rather, they consider an implicit default mode. In *R8* for example, this may be the *Idle* mode. Then, of course, *R7* does not crosscut *R8* because they refer to different modes of the system.

There is no way of knowing whether the requirements have implicit default modes. Therefore, requirements that do not mention a specific mode are considered as referring to all modes

3. Action Modifier

Functional requirements usually crosscut when they restrict normal functionality or ease other restrictions.

For example, *R8* above crosscuts *R7* because *R8* specifications restrict the functionality of *R7*.

In the following requirements, the restriction is eased:

*R9 During a call, when another phone transmits, the phone shall not initiate voice **transmission**.*

*R10 In **Emergency mode**, the phone should always be allowed to initiate voice **transmission**.*

In this case, when the phone is in *Emergency* mode, *R10* crosscuts *R9* and the restriction of *R9* is eased by *R10*.

3.3 The DRAS Methodology - an Overview

To solve the issues described above, the DRAS methodology has been developed. The methodology is used to identify and handle functional requirements that crosscut. It first identifies the actions used by each requirement, including the implied actions, the modes (or states) that are relevant for the requirement, and the action modifiers per action. Then based on this information, DRAS identifies the functional crosscutting requirements, the requirements they crosscut, and helps with generating the resulting *derived requirements (DRs)*. The generated requirements are textual, so that all stakeholders (including those with no technical background) can review and understand the requirements.

3.3.1 Implied Actions

When searching for requirements which may crosscut (based on actions), DRAS not only performs comparisons between actions directly used by the requirements, but it also takes implied actions into account. For identifying the implied and implying actions for a certain action *Act*, the methodology uses a knowledgebase that pre-defines lists of all actions that are directly used by each action. The list is defined based on previous knowledge and during initial analysis of the system's requirements. Recursive use of the list allows it to identify all actions that are *implied* by the use of that action. The knowledgebase also specifies whether the implied actions are always activated by *Act*, or they may be only activated by it.

For example:

*R3 When **another phone transmits**, the phone shall not initiate voice **transmission**.*

*R4 When **PTT is pressed**, the phone shall **initiate a call**.*

To identify whether one of the above requirements crosscuts the other, DRAS analyzes recursively the list of actions implied by *call initiation* (as specified by the implied actions knowledgebase) to check if *transmit* is a result of *call initiation*. This is shown in Fig. 1 (*Tx* is the abbreviation for *transmit*):

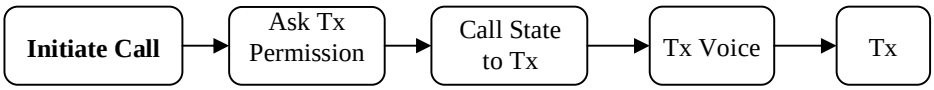

Fig. 1 Implied Actions for Call Initiation

Given that *R3* has a priority not lower than *R4*, then the crosscutting resolution may be:

R4(E) When PTT is pressed, the phone shall initiate a call, unless another phone transmits.

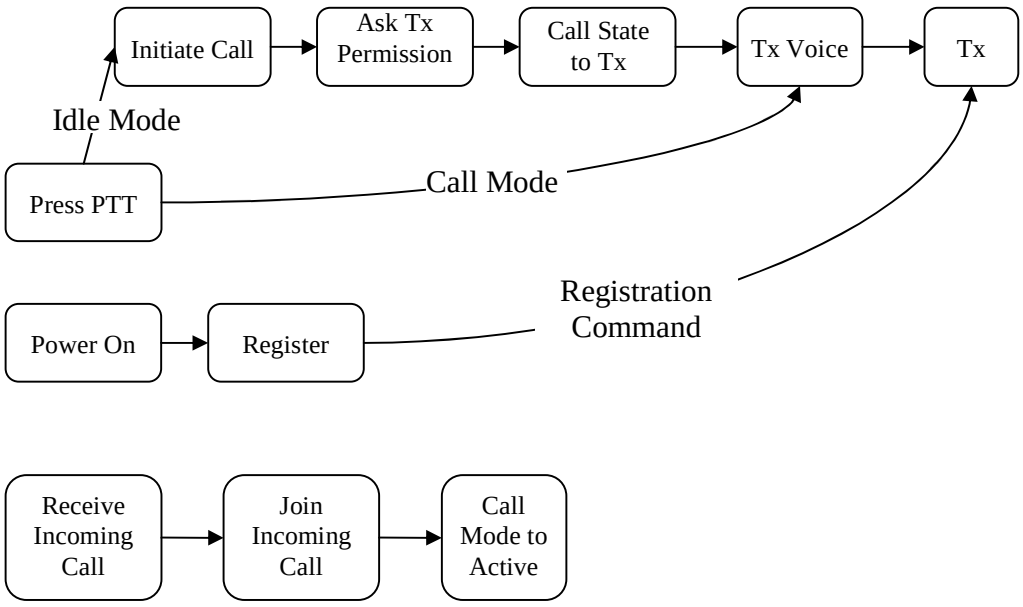

Fig. 2 Implied Actions (partial list)

A more complex example of the implied actions knowledgebase is shown in Fig. 2. It shows that several threads of actions can imply the same action; e.g., both Power On and Initiate Call imply transmission. Therefore, for example, not allowing transmission (Tx) means that call initiation or Power Off full functionality are also restricted.

3.3.2 Entities and Actions

In addition to the actions implied by other actions, the DRAS knowledgebase specifies which actions are related to each *entity* in the system. An entity is a sub-system, a user of the system etc., which is referred to by the requirements. Usually an entity has well defined interfaces with other entities in the system. In the cellular systems example, the entities are the phone, the cellular system, and the phone user. Similar to implied actions, the information about which actions are related to each entity is needed whenever an entity is being referenced to in a requirement. This information is also stored in the DRAS knowledgebase. For example, referring to an entity may mean referring to any action relevant to that entity.

See Fig. 3 below for the actions used by the *Cellular System* entity and the following requirement:

*R11 Illegitimate user should not be allowed to use the **cellular system**.*

R11 means that all actions relevant to the cellular system (initiating a call, etc.) are also not allowed to be activated by an illegitimate user.

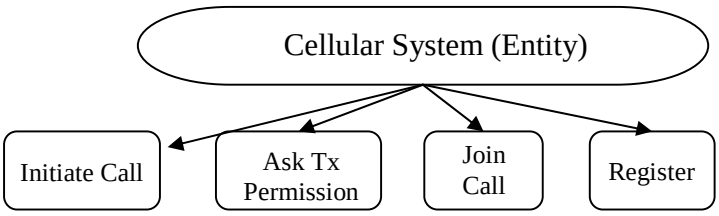

Fig. 3 Actions used by System Entity (partial list)

3.3.3 Crosscutting and Modes

DRAS identifies the *Modes* (and *States*) that each requirement is referring to. Normally, when two requirements relate to two orthogonal (mutually exclusive) modes, these requirements do not crosscut. That is, even if the two requirements use the same (implied) action, it can still be assumed that they do not have match-points (i.e., they do not crosscut) if their modes are orthogonal.

For example:

*R9 During a **call**, when another phone transmits, the phone shall not initiate voice **transmission**.*

*R10 In **Emergency mode**, the phone should always be allowed to **transmit**.*

As shown in Fig. 4, *Idle* mode and *Call* mode are orthogonal. That is, phone can either be in a call (*Call* mode) or not (*Idle* mode). However, the *Emergency* mode crosscuts both, because *Emergency* mode can be initiated no matter if the phone is in a call or not.

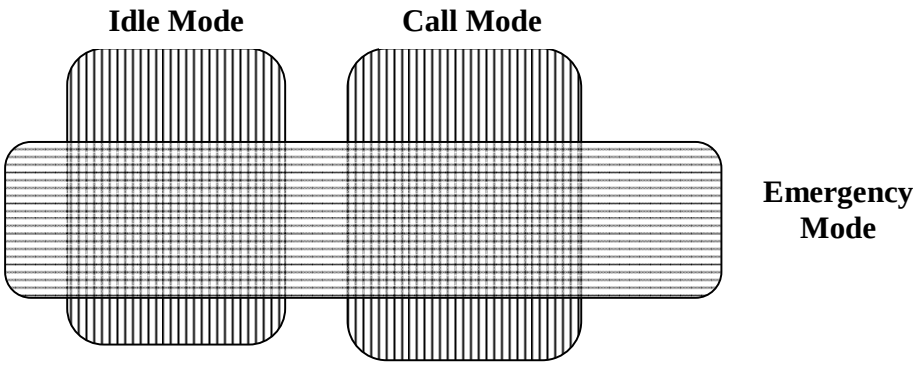

Fig. 4 Crosscutting Modes

In this example, *R9* does not refer to any specific mode; hence it refers to both *Normal* and *Emergency* modes (among other modes). Therefore, since *R10* refers to a call in *Emergency* mode, it tentatively crosscuts *R9* (which refers to a *Normal* mode call). DRAS takes into account requirements specifically related to emergency cases, which have higher priority than the requirements for general cases. A possible resolution to the conflict in the above crosscutting requirements may be:

R9(E) In Call mode, when another phone transmits, the phone shall not initiate voice transmission, unless it is in Emergency mode.

3.3.4 Action Modifiers

For each action used by a requirement, DRAS identifies its action modifiers, which specify restriction or ease of restriction for normal use of the action. DRAS can distinguish between three action-modifiers:

- **Restrict**: action is restricted or not allowed.
- **Unconditional**: action is always allowed, even if it was restricted by other requirements (ease of restriction).
- **None**: action not specifically allowed or restricted in certain modes or states. Usually, actions with a non-action-modifier do not need to determine whether the FR is crosscutting or not.

The information regarding action modifiers helps determine whether two requirements crosscut each other. If the use of an action is not restricted, or a restriction for its use is not eased, then the use of the action does not necessarily mean the requirements crosscut other requirements (unless there is a mistake in the requirements, such as: two contradicting requirements that are erroneously defined). The action modifiers are also propagated to the implied-actions.

Whether to consider the actions that are implied by an action, or to consider the actions that imply it, depends on the way action usage is restricted. If a requirement **restricts** the use of action *Act*, then all actions that **imply** *Act* are also restricted. For example, not allowing transmitting also means not allowing call-initiation, but not allowing call-initiation does not mean not allowing transmitting.

On the other hand, if a requirement **eases** the restrictions for using *Act* or allows using it **unconditionally**, then all actions **implied** by *Act* are also allowed. For example, permitting unconditional call-initiation in *Emergency* mode also means unconditional permission to transmit in this mode. Permitting unconditional transmission, however, does not mean unconditionally permitting call-initiation.

Therefore, an action-modifier is also used to determine the *direction* for identifying implied-actions (see Fig. 5). If an action *Act* is restricted, then the actions that imply *Act* are also restricted. If restrictions are eased ("Unconditional"), then restrictions for using the actions (implied by the action) are also eased.

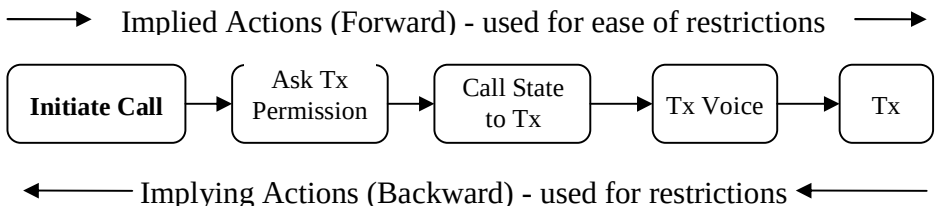

Fig. 5 Restriction and Ease of Restriction for Implied Actions

For example:

*R3 When **another phone transmits**, the phone shall not initiate voice **transmission**.*

*R4 When **PTT is pressed**, the phone shall **initiate a call**.*

Since *R3* restricts transmission according to Fig. 5, *R3* also restricts call-initiation; therefore, *R3* crosscuts *R4*.

Note that having only an action modifier does not mean that one requirement may crosscut the other. Usually, in order to crosscut, the action modifier should also contradict the action-modifier of the other requirement. For example, the following two requirements do not crosscut each other:

*R1 When PTT is pressed, the phone shall initiate voice **transmission**.*

*R10 In **Emergency mode**, the phone should always be allowed to **transmit**.*

Although *R10* eases a restriction for transmission, it does not contradict *R1*, because *R1* refers to permitting transmission and not to restricting transmission.

3.3.5 Requirements Priorities

The resolution of crosscutting between requirements depends on the priority of the requirements. The specification of a requirement with higher priority should override the specifications of requirements with lower priority. The use of relative priorities between

requirements (for handling crosscutting requirements) is inspired by existing methods, such as [Baniassad 04b, Rashid 03].

Note that it is difficult to assign relative priorities for each pair of requirements, i.e., to specify for each pair of requirements which requirement has a higher priority. In order to simplify the process, DRAS assigns one unique priority to each requirement. A functional requirement priority is based on the importance of the actions the requirement refers to and system state the requirement refers to. For example requirement about emergency actions will usually have higher priority than a requirement about other actions. Also, requirements that restrict operation in certain states will usually have higher priority than the requirements for general states.

The decision about a requirement priority is not deterministic and the final decision should be made manually, based on experience, domain knowledge, understanding the customer needs, etc.

Following is an example of conflicting requirements, where the analysis of requirements priorities can be used to resolve the conflict:

R12 Illegitimate users shall not be allowed to initiate calls.

R13 All users should be allowed to initiate a call to the police (an emergency number).

The resolution whether an illegitimate user can dial the police or not, can only be performed manually. That is, it should be determined which of these two requirements has a higher priority to define the proper DR.

It should be noted that assigning a unique priority per requirement is a simplification, as the requirements priorities do not necessarily form a transitive order. Thus, using a unique priority per requirement can only suggest which requirement has a higher priority. A main reason for this is that many of the requirements are unrelated, so it not possible to compare their relative priority. Another reason is requirement that refer to more than one action, as the reference to each action may have its own priority.

For example:

R14 When pressing PTT, the phone shall initiate a call.

R15 Illegitimate users shall not be allowed to transmit.

R16 The phone shall send its location to the system every minute.

R17 During a call, the phone shall not transmit its location.

As initiating a call requires transmission, *R15* is assigned a higher priority than *R14*. However, although sending location to the system also requires transmission, it may still be allowed for illegitimate users, e.g. to allow locating the phone in case of emergency. Therefore, *R16* is assigned a higher priority than *R15*. A conclusion is that *R16* has higher priority than *R14*. However, because of *R17* (which can be the result of a technical limitation of the system), initiating a call will stop sending the location for the duration of the call. That is, *R14* should have higher priority than *R16* to allow imitating calls. We see that different considerations lead to different relative priority of *R14* and *R16* and that the relative priorities between the requirements are not transitive.

3.3.6 Contribution and Composition Rules

After identifying which requirements crosscut which requirements, the effect of the crosscutting should be evaluated. This is performed before the requirements can be composed to generate DRs. Based on [Brito 03], two attributes are identified by DRAS: *contribution* and *composition rules*.

- **Contribution** - indicates whether the function (that the crosscutting requirement defines) conflicts with the function for the requirement it cuts ("**-**"), adds to its functionality ("**+**"), or does not affect it ("**None**").
- **Composition Rules** - based on the relative priority between requirements and the nature of the crosscutting functionality, the crosscutting requirement can be one of the following:
  - **Overlap Before/After** - add functionality before/after the functionality of the requirement it crosscuts.
  - **Override** - replace the functionality.
  - **Wrap** - encapsulate the existing functionality within new functionality.

3.4 Putting all together – the DRAS Outline

The DRAS methodology is based on the activities described earlier. Fig. 6 shows the process map for this methodology. Chapter 7 provides a full description of the methodology.

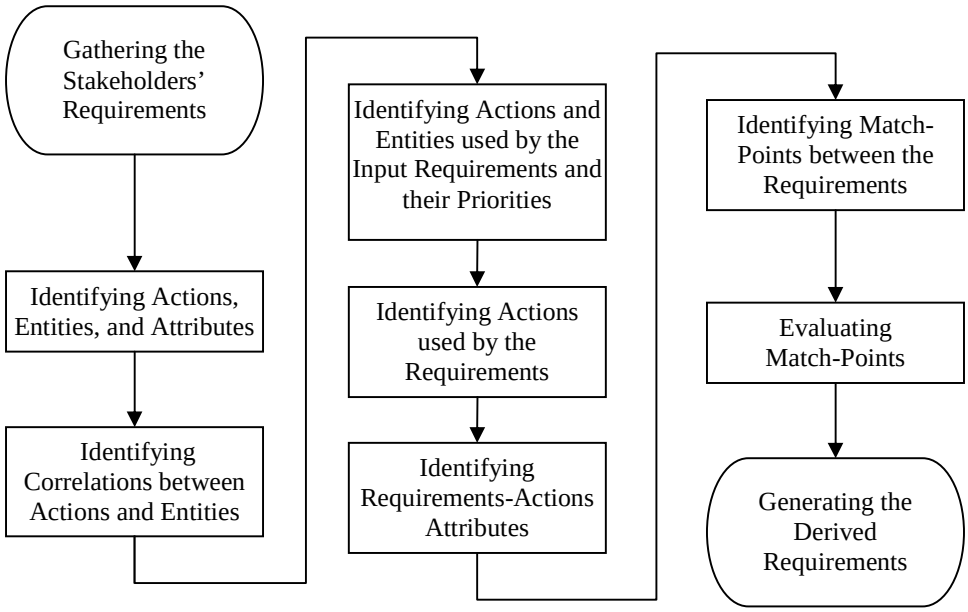

Fig. 6 DRAS Process Map

The functionality of each of the step is as follows:

1. **Gathering the Stakeholders' Requirement** (the input requirements for the process).
2. **Identifying Actions, Entities, and Attributes**.
3. **Identifying Correlations between Actions and Entities**.
4. **Identifying Actions and Entities used by Input Requirements and their Priorities**, including identifying requirements priorities and for each Action or Entity, their appropriate Modes and States.
5. **Identifying Actions used by the Requirements**, directly or indirectly.
6. **Identifying Requirements-Actions Attributes**, i.e., in what conditions the action is performed (according to the specified requirement).

7. **Identifying Match-Points between the Requirements**, using their common attributes (the common Actions, Modes, States, and Constraints).
8. **Evaluating Match-Points** to identify which of them should result in a derived requirement.
9. **Generating the Derived Requirements** according to the match-points identified.

4 The TETRA MS Example

This chapter gives an overview to TETRA, especially the TETRA MS (Mobile Station - the TETRA phone) and defines a set of requirements for the MS that are used later for evaluating different methods. The requirements are a small subset of the real TETRA MS requirements.

4.1 TETRA Overview

TETRA is a cellular system, mainly used for public safety and transit systems (police force, train systems, etc). TETRA voice services include both phone calls and push-to-talk (PTT) type of calls. PTT services support both Group and Private calls. The TETRA air interface standard is defined by ETSI in [TETRA]. In TETRA, as opposed to cellular systems such as GSM, only the air interface and equipment interface (used by end users) are standardized. (In GSM and other cellular systems, the interface between different system components is also standardized.) In many cases, the TETRA system is owned by the customer, while in most other cases a cellular system is owned by an operator that sells services to customers.

A primary differentiator between TETRA and most other cellular systems is its emergency services features. Some of emergency services features are: Emergency Alarm, Emergency/Priority Call, Call Preemption, Ambience-Listening, Hot-Mic, and more. These services usually don't exist in cellular systems. Another difference is that in TETRA, a user should be able to start talking almost immediately (less than one-half of a second) after starting a call using PTT. In comparison, PTT services currently supported cellular systems, such as GSM, allow the user to start talking only after few seconds. Emergency and Priority calls add many interactions between features. Therefore, they are significant and a major part of the requirements defined in this work.

Most TETRA systems include a control center, where a human operator can: control calls, interrupt calls, connect (patch) different calls, broadcast to a site/system, control emergency operations, and more.

4.2 TETRA MS Features and Functionality

Following is a short description of TETRA MS features and functionality that used to define the requirements set used in this work:

- **Registration (to the System)**: In cellular systems, the MS (phone) usually registers to a system before it can get service from it. Registration is required for several reasons:
  1) Authenticating the MS and the system.
  2) Ensuring that the MS is authorized to get support from the system (e.g., the user has paid his bill, or the user is a member of the police force that owns this system).
  3) Allowing the system to know that the MS is active, etc.
     One known exception is an emergency call (such as a 911 call in the US) in systems (such as GSM), where in any case, the emergency call should be allowed.

- **In/Out of Coverage**: Cellular systems coverage is limited due to: 1) their RF signal propagation distance is limited, and 2) because of system and MS loss of synchronization starting from a certain distance (because of a delay in receiving the signal). The MS can register and get services from a system only when it is within the system coverage range. While out of the system coverage range, the MS periodically searches for the system. The user services allowed (when out of coverage) are limited to local MS functions (e.g., browsing the phone book).

- **Group Call (Half Duplex)**: A major (and maybe the main) service for cellular systems used for public safety (such as TETRA systems) is the Group Call, which allows a user to talk to a group of people. This service simulates a walkie-talkie service, where all over-the-air radios that are tuned to the same frequency can hear all

other over-the-air radios. In cellular systems, there are mechanisms that prevent anyone from interrupting the talking party, although there are also mechanisms that will allow a graceful interruption in high priority cases. The Group Call is in Half Duplex mode. If two users were allowed to talk at the same time (as in a full-duplex phone call), a third user would not be able to hear any of them.

A Group Call is usually initiated by pressing and holding the Push-To-Talk (PTT) button; the transmitting phone continues to transmit until the user releases the PTT button. Therefore, the user does not need to again press (push) the PTT button to end the Group Call. After the PTT button is released, the call usually remains active for a few more seconds, allowing others to respond. (It is also possible [by pressing the PTT button separately each time] to start a new call, but if the system is busy, it may mean that no resources will be available for a response.)

A Group Call can be received whenever the MS is busy in a call, or when the MS is idle. The decision whether to receive the new incoming call, while the MS is busy with another call, is usually based on the priority for each of the two calls.

- **Idle vs. Call Mode**: Normally in a cellular system, MS functionality is different if it is NOT in a voice call, than if it is in a call. For example, if NOT in a call, almost all received incoming calls will be accepted; if in a call, only higher priority incoming calls will be accepted. For the purpose of this work, Idle Mode is when the MS is NOT in a call, while Call Mode is when it is in a call. (In reality, the definition is far more complex, e.g., there is a duration after the Group Call ends when incoming calls [for that group] will have higher priority, because it is assumed that the incoming call is a continuation of the previous call.)

- **Call Priority**: In TETRA, each call has an assigned priority. The priority can be predefined for the user, set according to the state of the MS, etc. Using Call Priority, the system can preempt (stop) an active call to free resources for another higher priority call; in a call, MS may switch to another higher priority call, etc. In TETRA there are 15 levels of priority. But in this work only two will be discussed: Normal

and Emergency.

- **Emergency Call Priority**: In TETRA, Emergency Call Priority is the highest call priority. It has a distinct name because it is used not just to set higher priority, but also to indicate a serious problem, such as in situations of life and death. Therefore, an Emergency Priority Call provides additional functionality that is not allowed in other cases, such as longer or unlimited talk time.

- **Emergency Alarm and State**: In TETRA systems, the usual functionality is to be able to initiate an Emergency Priority Call, but the MS should first be in an Emergency Mode. This is usually performed by pressing a designated emergency button. When the button is pressed, the MS sends "emergency alarm" signals to the Control Center and it enters the "emergency state". While in the emergency state, the MS is usually limited in its functionality (e.g., it will not receive non-emergency priority calls). This limited functionality guarantees that the MS is free to perform functions that are needed to handle the emergency situation.

- **TXI (Transmission Inhibit)**: There are some cases where it may be dangerous to allow the MS to transmit any signal. For example, if the user is in an explosive area or in a hospital. For such cases, the MS user can set the MS so that it cannot transmit (unless it is in an emergency mode). In TXI mode, the MS may receive Group Calls (because it can only listen to these calls), but it cannot initiate calls or receive one-to-one calls (e.g., a phone call).

4.3 Simplified Requirements Set

The following sections define the requirements set for the purpose of this work. The requirements are a highly simplified subset of the requirements for the TETRA MS. Two sets of requirements are defined. One set includes **baseline requirements** for normal, non-crosscutting functionality for the MS. The other set includes **crosscut requirements**

related to emergency/priority/TXI, where normal behavior is suspended or changed because of required higher priority activities. The requirements for higher priority activities, defined by the second set, are crosscutting requirements that cut the first set of requirements.

The interactions between different crosscutting requirements will be discussed later, using the different AORE methods described in this work.

Note that in systems like TETRA, there are several other crosscutting requirements and some of them crosscut each other. For this work, the requirements were simplified so as to clarify which of the functional requirements crosscut which of the other functional requirements. Also, there is no reference to the TETRA over-the-air protocol (defined by [TETRA]), although all incoming or outgoing commands/messages referenced in this work (except for the User Interface) are performed using this protocol.

For the defined requirements, a simplified TETRA MS is assumed:

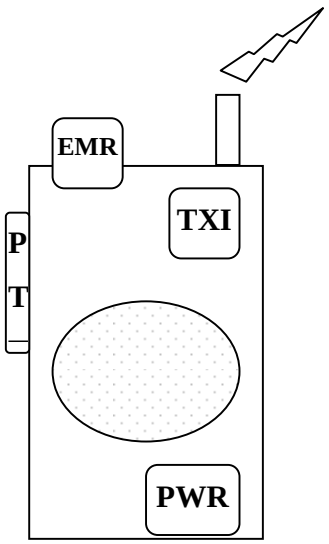

Figure-1 The MS platform and buttons used by this work

- It supports only half-duplex group calls. No other types of calls (private, phone, etc.) are supported.
- It can initiate an outgoing call to only two predefined groups: one used when in Normal Call Mode (the Normal Group), and the other used when in Emergency Call Mode (Emergency Group). The user cannot dial a group number and cannot select within pre-defined groups.
- Pressing the Emergency button only puts the MS into Emergency Mode; it does not cause the MS to send an Emergency Alarm signal to the Control Center.
- A single cell system is assumed. Therefore, there is no need to handle cell handover re-registration on a new cell when in TXI mode, etc.
- Only the following buttons are available for users (as shown in Figure-1): Power On/Off (PWR), Push-To-Talk (PTT), Emergency (EMR), and Tx Inhibit (TXI).

4.4 Baseline Requirements (Stakeholders' Requirements)

The defined requirements are simplified TETRA stakeholders' requirements. The defined requirements are based on experience, and not on one of the methods developed for discovering stakeholders' requirements. The methods include Viewpoints, Use-cases, Goals or Problem Frames; see [Rashid 02; Rashid 03] for references to some of the above-named methods. Because only stakeholders' requirements are the basis for this work (and they had to be simplified for this purpose), the above-named methods were not used.

Note that in some cases, functional requirements and user interface requirements (using the platform in Figure-1) were combined into one requirement. This was done so as to simplify the handling of the requirements set and traceability for this work. An example of such a simplification is Req-150. The functional requirement is to allow switching between emergency and normal mode; the user interface requirement is that switching be done by pressing the EMR button.

4.4.1 Baseline Requirements - Attributes and Facts

Before defining requirements, some attributes (used by the requirements) must be defined. Some of this data could have been defined as requirements, but for the sake of simplicity, it is given below.

Attributes:

- **Coverage Mode** - whether MS is within system coverage: ***In-Coverage*** or ***Out-of-Coverage*****.**
- **MS Call Mode** - whether MS is in call mode: ***Idle*** **mode** or ***Call*** **mode.** (For short, *Idle/Call* mode will be used later instead of *Idle/Call Call* mode.)
- **Tx Mode** - whether MS is allowed to transmit (Tx): ***Tx-Allowed*** **(*TXA*)** or ***Tx-Inhibit*** **(*TXI*).**
- **Call Priority** - from low to high call priority: ***Normal*** or ***Emergency*****.**
- **Priority Mode** - from low to high MS priority mode: ***Normal*** or ***Emergency*****.** (For short, *Normal/Emergency* mode will be used later instead of *Normal/Emergency Priority* mode.)

The actions taken when pressing different **MS buttons** (except for PTT) are:

- **Emergency (EMR) Button**: Toggles the MS between *Emergency* Mode and *Normal* Mode. The initial priority mode at power-on is *Normal*.
  [A general note when toggling to all modes: ***toggle*** *-try to change the mode name from "toggle" to "change mode," because the second name is better in some cases. I decided to keep only "toggle" so as to reduce the number of actions in the list during the evaluation of the Theme/Document.]*
- **TXI button**: Toggles between TXI and TXA modes. The initial Tx Mode at power-on is Allowed (TXA).
- **Power (PWR) button**: Toggles between Power-on and Power-off for the MS.

Other general attributes are:

- **Priority of Calls**: All calls with the same Call Priority have the same priority.

4.4.2 Baseline Requirements – System Related

The following requirements are for MS registration at power-up. *Normal* mode and *Tx Allowed* mode are assumed. According to the requirements, these are always the modes when the MS starts at power-up, regardless of the modes it was in during a previous power-off. (Note that this is for simplification only; in a real case, the MS can remember the Priority and Tx Modes during the power off/on cycle.) The detailed registration process, including sub-steps (such as authentication), is not included.

Req-250: **On power-on**, MS shall register to the system.

Req-260: **On power-off**, the MS shall de-register first from the system, if it is successfully registered.

Req-270: MS shall be able to **power-off** in any state.

4.4.3 Baseline Requirements – Group Call

The following requirements assume that the MS is in *Normal* priority and *Tx Allowed* modes. The requirements, when the MS is in *Emergency* priority or *Tx Inhibit* modes are defined separately, and regarded as crosscut functional requirements.

Req-310: **Pressing PTT in *Idle* mode** shall initiate a request for an outgoing group call to the system, with *Normal priority*, to the predefined *Normal group*. If acknowledged by the system, MS shall toggle to *Call* mode and may start transmitting voice.

Req-320: **Pressing PTT in *Call* mode** shall cause the MS to ask the system for *permission to Tx voice*, when no one else is transmitting in the call. The MS may start to *Tx* voice only if allowed by the system. The PTT shall be ignored when someone else is already transmitting in the call.
Note: this type of call is half-duplex, i.e. in this case, parallel transmission by several participants in the call is not allowed.

Req-330: **When receiving *incoming* Group call in *Idle* mode**, MS shall toggle to *Call* mode and join the call.
Note: in Group Call there is usually no need to acknowledge the receipt of the incoming call message, because such an acknowledge will probably collide with the acknowledge of other group call participants. This is why the MS can listen to a group call while in TXI Mode.

Req-340: **When receiving *incoming* Group call in *Call* mode**, the MS shall internally reject the call, without notifying the system.
Note: rejection is done internally to the MS and no message is sent over the air.

4.5 Crosscutting (Aspectual) Requirements

4.5.1 Aspectual Requirements - Emergency Mode

Req-520: **Pressing PTT in *Emergency* mode** shall **always** allow the MS to initiate a call, as soon as possible, to the *Emergency group* with *Emergency priority*.

Req-540: **When receiving *Incoming* Call with *Emergency priority,*** the MS shall join the call if it is not engaged in *Emergency* call.

4.5.2 Aspectual Requirements – TXI Mode

Req-610: **When in *TXI* mode**, MS shall ignore any request to transmit.

4.5.3 Aspectual Requirements – System Related

Req-710: **While MS is *unregistered***, no system related operations should be allowed by the MS (e.g., the MS shall not be allowed to initiate calls and should reject all incoming calls).

Req-720: **When MS is *out of coverage***, pressing PTT shall be ignored.

Req-730: **When MS powers-on while it is *out of coverage***, it should not try to register.

Req-740: When MS is *out of coverage*, MS shall not try to transmit.

4.6 Derived Requirements from Baseline and Aspectual Requirements

The requirements in this section are derived from aspectual and baseline requirements. These requirements were generated based on previous knowledge about real-life TETRA MS requirements and behavior. No specific method was used to generate them (including not using the DRAS method developed in this work). (In fact, these requirements were specified before DRAS was developed.) The purpose for defining these requirements is: to be able to evaluate the effectiveness of the different methods (for correlating between baseline and crosscut requirements). Each method is expected to "generate" these requirements. Whether this is the case, and how easy this is achieved is a major part of the evaluation that follows.

4.6.1 Out-of-Coverage related Derived Requirements

Req-1110: When MS is powered-on but registration to the system was not successful yet, the power-off button press shall cause the MS to power-off.
[Resolves the conflict between Req-270 and Req-710. Practically means that the only button that is active before successful registration is the power-on/off button.]

Req-1120: When MS is out-of-coverage and is unregistered, the MS shall register to the system once it is in coverage.
[Resolves the conflict between Req-250 (that requires that the MS will register on power-up) and between Req-730 (that does not allow registration when MS is initially out of coverage).]

Req-1130: On power-off, when MS is out of coverage, the MS shall be allowed to power-off without trying to de-register from the system.
[Resolves the conflict between Req-260 and Req-740. Allows the MS to power down without de-registration first, if out of coverage.]

Req-1340: MS shall not try to register to the system when in *TXI* mode.

[Resolves the conflict between Req-250/Req-1120 and Req-610. Allows the MS not to register if it was turned on out of coverage and then put into TXI mode.]

Note: the conflict solved here is between crosscutting and a derived requirement because of other crosscut requirements.

4.6.2 Registration related Derived Requirements

Req-1410: When MS is unregistered in *Normal* mode, a PTT press shall be ignored.

[Resolves the conflict between Req-310 and Req-710. MS in normal mode should not ask to talk while out of coverage.]

5 Related Work

As described in Chapter 2, different AORE methods were suggested for handling crosscutting (aspectual) requirements. Some of these methods directly refer to crosscutting requirements and how to combine them with other requirements. Others handle the separation of concerns based on customers' requirements.
The following is a summary of the applicability of several AORE methods to the DRAS methodology developed in this work. The applicability evaluation is a summary of a through review done for these methods. Methods that are directly relevant to the work will be further evaluated in the following Chapter 6, which discusses how well they handle the requirements defined earlier in Chapter 4 (identifying derived requirements).
Some papers include an exhaustive survey of existing methods and approaches; for example, see [Chitchyan 05]. The description includes the main characteristics for each of the methods and a description of their processes. The characteristics are partly based on [Bakker 05; Chitchyan 05] which characterizes the different approaches and tries to split the characteristics into a few major categories. However, several categories currently exist, so each approach is characterized separately.

5.1 Viewpoints

Viewpoints [Finkelstein 96] are used to specify the system from the perspectives (viewpoints) of each of its users (Actors in the Use Cases terminology). Usually each of these perspectives is partial and incomplete, because of the different roles for each user. However, a separate evaluation for each viewpoint is needed in order to define the full system's specifications. For a complex system, using viewpoints allows the Separation of Concerns between different viewpoints, and provides a more manageable means of handling the system's specifications. Viewpoint-oriented methods do just that.
[Nuseibeh 04] presents a viewpoint as an encapsulating knowledge representation, process, and specification; all from the user viewpoint. Several Viewpoints-Based Requirements Engineering (VBRE) methods exist. [Silva 02] for example, introduces an approach for classifying and diagnosing discrepancies between viewpoints.

Although the main purpose for Viewpoint methods is to verify that requirements cover all viewpoints, they deal with the Separation of Concerns and not specifically with identifying crosscutting requirements. Therefore, these methods will not be evaluated further.

5.2 Goal Oriented Requirements Analysis

Goal Oriented Requirements Analysis (GORA) is described in [Mylopoulos 01].
It explores the alternatives for achieving the goals in a given set of high level requirements. GORA correlates *Softgoals* (non-functional requirements) with goals and other softgoals; this is similar to analyzing crosscutting aspectual requirements.
An enhancement of this method is Aspects in Requirements Goal Models (ARGAM) by [Yu 04; Chitchyan 05].

One main purpose GORA is the evaluation of alternatives. Note that the term “Softgoal” is defined in [Chung 00] as a framework for handling Non-Functional Requirements. Although using the NFR Framework method itself is not mentioned in this paper, this paper clearly relies on this framework. The correlation analysis mainly handles NFRs as a whole; consequently, GORA is not well suited to correlating between base and crosscutting **functional** requirements, such as those defined in this work.
For similar reasons, ARGAM will not be evaluated any further; it is mainly used to identify non-functional (Softgoals) aspects.
However, parts of the methods are relevant. goal and softgoal correlation analysis, where baseline requirements replace goals and crosscutting requirements, are used instead of softgoals. The evaluation of alternatives may be relevant for selecting the right derived requirements (from the different resolution alternatives).

5.3 Modularization and Composition of Aspectual Requirements

The Modularization and Composition of Aspectual Requirements (MCAR) method is described in [Rashid 03; Rashid 02]. This method defines an AORE process model from

identifying stakeholders' requirements and concerns related to these requirements, to resolving conflicts and determining their influence on later architecture and design development stages. ([Bakker 05] calls this method AORE, but since AORE is a general term, the method will be called MCAR in this work.)

Although this method is mainly applicable for crosscutting NFRs, some of the methods it uses are also applicable for crosscutting FRs. In the context of this work, stakeholders' requirements are the baseline requirements, and aspectual concerns are the crosscutting requirements. Since the purpose of this work is to evaluate the effectiveness of composing baseline and crosscutting requirements (to get derived requirements), not all steps for this method are applicable. However, some steps are applicable; so the method is further evaluated using the TETRA requirements in Section 6.1.

5.4 Composition Process for Aspect Oriented Requirements (AOR)

This method is described in [Brito 03]. It describes the process of composing crosscut concerns with concerns (requirements) they cut across. The method is mainly applied for non-functional concerns (requirements), but as shown below, it also includes techniques that are applicable for functional requirements.

The main purpose for the additional approaches introduced by this method over MCAR (described in Section 6.1), is the identification of match-points between elements of the model, and the use of crosscutting operators (Overlap, Override, and Wrap). These methods seem to be valuable for evaluating requirements defined in this work, in order to generate derived requirements. Therefore, these methods are further evaluated in Section 6.2.

5.5 Adaptation of the NFR Framework to AORE

This method is described in [Sousa 03a]. The method is an enhancement to the method defined in [Rashid 03]. It also includes parts from the method defined in [Mylopoulos 2001].

This method is applicable to requirements defined in this work. However, its enhancement over [Rashid 02; Rashid 03] is not enough to justify a detailed analysis, in addition to the analysis already given for that method. Therefore, this method will not be further evaluated in this work.

5.6 Crosscutting Quality Attributes

Crosscutting Quality Requirements method is described in [Moreira 02, Brito 02]. The method proposes a model to identify and specify ***Quality Attributes*** (QA) that crosscut requirements at the requirements analysis stage. QA is a non-functional concern, such as response time, accuracy, security, and reliability. This is the same as in a NFR, but from the point-of-view of the functional requirement.

This method is only partly applicable for generating derived requirements from the requirements defined in this work. It mainly handles NFR and Quality Attributes requirements. Also, the main methods it uses are also included in other methods evaluated in this paper ([Brito 03; Rashid 03]). Therefore, this method will not be discussed any further in this work.

5.7 Theme and Theme/Doc - Finding Aspects in Requirements

This method is defined in [Baniassad 04a; Baniassad 04b]. The *Theme approach* [Baniassad 04a] is a method and set of tools developed for early identification of aspects in the software development life cycle. The *theme* notion represents a system feature. Themes can be either *base themes* (which may share some structure of behavior with other base themes), or *crosscutting themes* (*aspects*) which have a behavior that overlays base themes functionality. The Theme/Doc approach can identify aspects from FR interrelated behaviors, not just aspects from the NFR (as most other methods identify).

Because the Theme/Doc approach helps identify aspects from FR interrelated behaviors (not just aspects from a NFR, like most other methods identify), it has the potential to be highly applicable for requirement types defined in this work. The approach is mainly used for the Theme/Doc tool, to discover whether aspects in requirements are applicable. This work describes only the requirements phase and does not delve any further to the design phase. This method is further evaluated in Section 6.3.

5.8 Mining Aspects

Mining Aspects by [Loughran 02] support storage and mining aspects, with special focus to AOSD; this is a specific type of mining for existing assets. Mining existing assets generally refers to locating useful information (from an organization's asset base) for reuse in new applications. Asset mining can occur at many different stages, throughout the software development lifecycle. Typical assets for mining can include: program code, designs, system architectures, specifications, etc. Effective mining requires support tools that effectively store the data and enable a relatively fast retrieval of data (for the mining process). [Rosenhainer 04] suggests identifying aspects in requirements. [Sampaio 05] describes the approach for mining aspects in requirements in his document, based on Theme/Doc [Baniassad 04b; Rosenhainer 04], but utilizes corpus-based natural language processing (NLP) techniques. [Garcia-Duque 06] presents a method to separate aspects from specification. To support the identification of crosscutting concerns and allow the mining for aspects, the specs are first represented in a formal model, using an enhanced version of SCTL-MUS (Simple and Causal Temporal Logic - Model of Unspecified States) by [Pazos-Arias 01].

Mining aspects for requirements deals with methods to store requirements data, so that they allow automatic or semi-automatic retrieval and identification of aspects. Identifying aspects in requirements is highly relevant to identifying crosscutting requirements and the requirements they cut across, therefore, these methods are highly relevant to this work. Mining aspects methods are not used in this work, although they are relevant candidates for further enhancements (see Section 9).

5.9 Other Methods

Several other AORE methods are suggested; only a few of them are mentioned here. These methods will not be evaluated for applicability to DRAS.

[Grundy 99] proposes the AORE method for c**omponent-based software**. The proposal addresses some difficult issues regarding component requirements engineering, by characterizing components (based on different aspects of the applications a component addresses). Examples of components' aspects are: User Interface, Collaboration, Persistency, Distribution, and Configuration.

[Pang 04] proposes an aspect-oriented refinement for the Agile **Feature-Driven Development (FDD)** lifecycle by [Palmer 02]. The refinement includes using a *boundary condition exploration*, while building a features list. This method was proposed by the authors to assist with detection and prevent inconsistencies between features. The method is based on a fact they found; most inconsistencies and conflicts (between features) happen across the boundary condition for features. To refine feature planning and design, they adopt aspect-oriented development methods.

[Sousa 04] proposes a **Use Case** driven approach for the Separation of Concerns from requirements. It adapts some use-case activities (from the Unified Software Development Process by [Booch 99]) in requirements, analysis and design, and includes NFR framework activities (by [Mylopoulos 01; Sousa 03a]).

6 Deeper Evaluation of Some AORE Methods

In this chapter we further evaluate some AORE methods which are described in Chapter 5 and are applicable for generating derived requirements. We evaluate how well these methods can obtain the derived-requirements, defined in Chapter 4.6. The conclusions of the current chapter helped to define the DRAS methodology which is fully described in Chapter 7.

The evaluated methods are based on the conclusions in the previous chapter:

- Section 6.1: Modularization and Composition of Aspectual Requirements (MCAR) [Rashid 03].
- Section 6.2: Composition Process for AOR [Brito 03].
- Section 6.3: Theme/Doc for Finding Aspects in Requirements [Baniassad 04a; Baniassad 04b].

6.1 Modularization and Composition of Aspectual Requirements (MCAR)

6.1.1 Overview

The Modularization and Composition of Aspectual Requirements (MCAR) is described in [Rashid 03]. It focuses on modularization and composition of requirements level concerns that cut across other requirements. The method is mainly applicable for NFRs, such as availability, security and other requirements that cannot be encapsulated by a single viewpoint or use-case. The method intends to:

1) Support the separation of crosscutting FR and NFR properties, and
2) Help identify the mapping and influence of requirements level aspects on artifacts at later development phases; thus establishing critical tradeoffs before the architecture is derived.

The method is supported by the Aspectual Requirements Composition and Decision tool (ARCaDe). XML is used in the tool to define the different requirements and aspects.

ARCaDe is not evaluated in this work, because our main purpose is to understand whether the principles for this method are helpful in identifying derived requirements.

The main process steps (performed using the ARCaDe tool) are:

1. **Identify and specify stakeholders' requirements and concerns**: This mainly involves the separation of FRs and NFRs. The output is XML Viewpoints for main entities identified in the requirements. Each Viewpoint defines a set of requirements.
2. **Identify and specify concerns**: It identifies crosscutting concerns (from NFRs) that have the potential of becoming Aspects. The concerns are (NFR) requirements that crosscut other requirements. The output is XML Concerns definitions. Each Concern defines a set of requirements.
3. **Identify coarse-grained concern/viewpoint relationship**: It relates to viewpoints and concerns. The output is a Viewpoints/Concerns relationship matrix.
4. **Identify candidate aspects**: From the concerns/viewpoints relationships defined in the previous step, it identifies the concerns that crosscut several viewpoints, and therefore, are candidate aspects. In the XML definition, these Concerns are transformed into Aspects.
5. **Define composition rules**: Composition rules define the relationship between aspectual requirements and viewpoint requirements at a fine granularity (unlike the relationship matrix defined earlier, which was used only to identify aspects). The output is an XML definition of composition rules.

   The composition rules define how the requirements are *constrained* by aspectual requirements, and what is the expected output for these constrains. The operator-action Constrained and expected Output (defined in this paper) are:

   a. *Constraint Actions*: enforce, ensure, provide, applied, exclude.
   b. *Constraint Operators*: during, between, on, for, with, in, XOR.
   c. *Outcome Actions*: satisfied, fulfilled.
6. **Compose the aspects and viewpoints**: Using composition rules, the aspects and viewpoints are composed. The process helps identify conflicts between aspects

that constrain the same requirements. In practice, the composition itself may be delayed until conflicts are resolved.

7. **Handle conflicts between candidate aspect**s: It determines how aspects contribute to other aspects, in case they constrain the same requirement. It is also used to determine which aspectual requirement is "stronger" by setting importance *weights* to the aspects (in case of a conflict between aspects). The output is a contribution table that shows the aspect contribution to another aspect to be positive or negative. The table is used to resolve conflicts between aspects.
8. **Specify aspects dimensions**: It is used to determine the aspects' influence on architecture, and design development stages that come later. It also identifies their mapping to a *function*, *decision,* or *aspect*.

Although the method is mainly applicable for identifying crosscutting NFRs, some of the methods it uses are also good for identifying crosscutting FRs. In the context of this work, stakeholders' requirements are the baseline requirements and aspectual concerns are the crosscutting requirements. Since the purpose of this work is to evaluate the effectiveness of composing baseline and crosscutting requirements, in order to generate derived requirements, not all steps for this method are applicable. Therefore, not all steps for these methods will be evaluated further. Also, the requirements and composition rules will not be specified using XML because the purpose of this work is only to define *textual* derived requirements. Note that it may be possible to translate back the XML to requirements; so this may be the subject of a future work.

These are the method's steps that will be used to evaluate requirements defined in this work:

- **Identify and specify stakeholders' requirements**
- **Identify and specify concerns**
- **Identify the coarse-grained concern/viewpoint relationship**: These relate to baseline and crosscutting requirements.
- **Identify candidate aspects**: It identifies which of the crosscutting requirements can be considered as aspects.

- **Handle conflicts between candidate aspects**: It determines the weights of the crosscutting requirements and resolve conflicts.
- **Compose the aspects and requirements**: It generates the derived requirements.

During the evaluation of this method, the definition for both base requirements and crosscutting requirements (defined in Chapter 4 of this work) were enhanced. In addition, the method enabled a better identification and definition of derived requirements. This method was the first excellent, applicable method evaluated using this set of requirements. The enhancements were done mainly while creating Table 1 and Table 2.

6.1.2 Input Requirements Analysis using MCAR

This section evaluates the use of MCAR to analyze the input requirements, defined in Chapter 4.

6.1.2.1 Identify and Specify Stakeholders' Requirements

These are the base and crosscutting requirements defined in Chapter 44.2.

6.1.2.2 Identify and Specify Concerns

These are the base and crosscutting requirements defined in Chapter 4.

6.1.2.3 Identify the Coarse-grained Concern/Viewpoint Relationship

The table below identifies the crosscut/baseline (coarse-grained concerns/viewpoints in the original method terms) requirements relationship. The numbers, for the related derived requirements, were already inserted into the table for the purpose of this method evaluation. In practice, they are defined only at the end of the process.

Table 1 Correlation between Base and Crosscut Requirements

Baseline Req or Crosscut Req	Req-250	Req-260	Req-270	Req-310	Req-320	Req-330	Req-340
Req-520				√ 1230	√ 1240		
Req-540						(√)	√ 1250
Req-610	√ 1340	√ 1330		√ 1320	√ 1320		
Req-710			√ 1110	√ 1410	√		
Req-720					√		
Req-730	√ 1120						
Req-740		√ 1130					

Legend: **"√"** – indicates where requirements are related
"(√)" – indicates where related requirements do not affect each other
"<number>" – indicates a new/enhanced derived req. no.

Notice that not all conflicting requirements created new derived requirements. These are cases where the behavior is imposed by the TETRA system, and there are no alternative behaviors for the MS behavior. For example, if the MS is out of coverage range (Req-720), it cannot start a call (Req-310). Also, in the specific cases presented, all of these cases are related to ignoring key presses (mainly PTT) in certain situations. It is assumed that in these cases the requirement to ignore the key press is enough, since functionality is as if the key was not pressed. Therefore, there is no need to enhance the related baseline requirement

While evaluating the method described in [Brito 03] (see Section 6.2), it was found that derived Req-1340 is also a result of the crosscut Req-610 crosscutting derived requirement Req-1120 (in addition to Req-610 crosscutting Req-250). This finding suggests that an additional step is needed: the evaluation of crosscutting requirements vs. derived requirements (vs. baseline requirements). For simplicity, this step was not done.

6.1.2.4 Identify Candidate Aspects

As seen in Table 1, all crosscutting requirements are candidates to becoming aspects. This is not surprising, because the crosscutting requirements chosen for this work are known to be important crosscutting requirements in the real world. Note that the baseline requirements that are most affected by the crosscutting requirements are Req-310 and Req-320. The reason is that both of them handle the initiation of a group call (which is the main subject of the requirements set defined here), and therefore, are the most affected by the crosscutting requirements.

6.1.2.5 Handle Conflicts between Candidate Aspects

Table 2 shows the correlation between crosscut requirements.

Table 2 Correlation between the Crosscut Requirements

Baseline Req or Crosscut Req	**Req-520**	**Req-540**	**Req-610**	**Req-710**	**Req-720**	**Req-730**	**Req-740**
Req-520			- 1310	- 1210	- 1220		
Req-540			(√)	(√)			
Req-610						+	+
Req-710					+		
Req-720							
Req-730							
Req-740							

Legend: "+" or **"-"** indicate whether the requirements positively or negatively contribute to each other.

The table shows that new/enhanced requirements are needed only when the crosscutting functional requirements contribute negatively to each other. Only in these cases, there are conflicts between requirements that should be solved.

Note that in this case, aspectual requirements are also crosscutting each other to generate derived requirements (unlike the MCAR method). In the MCAR method, aspects crosscut only viewpoints (equivalent to baseline requirements). This is because MCAR handles mainly crosscutting NFRs, while here the aspects are FRs.

The original method also gives weights to conflicting crosscutting requirements, to allow easier resolution of conflicts between requirements. This step was not used, but will be used when the method from [Brito 03] is evaluated (see Section 6.2).

6.1.2.6 Compose the Aspects and Requirements

The composition of the requirements is performed according to identified relations and conflicts, as shown in Table 1 and Table 2. The result is the derived requirements.

6.1.3 Applicability of MCAR for creating Derived Requirements

This method includes techniques that are very applicable to the type of crosscutting functional requirements defined in this work. Not all of the steps are relevant, because the requirements are already detailed and not given initially at a level of viewpoints (use-cases or similar level of presentation). Although mapping the requirements to later development phases was not used, it may also be applicable.

Using the method, it was possible to identify all derived requirements. The visualization of correlations between the different requirements (using the tables above) is very useful. However, all of this was mainly manual work, because the method does not provide a tool for automatically identifying the correlation between baseline (stakeholders') requirements and crosscutting (concerns) requirements. In other words, the identification of correlations between requirements is only based on expert judgment.

6.2 Composition Process for Aspect Oriented Requirements (AOR)

6.2.1 Overview

This method is defined in [Brito 03]. It describes the process to compose crosscut concerns with concerns (requirements) they cut across. The method is mainly applied for

non-functional concerns (requirements), i.e., NFRs. But as shown below, it includes techniques that are also applicable for functional requirements.

The main concepts used by this method are:

- *Match-Point*: A point where one or more crosscutting concerns are applied to a given functional concern (functional requirement, in the context of this work). Match-point is an abstraction of the *join-point* concept used in AOSD (for an example, see [Laddad 03]).
- *Conflicting Aspect*: Conflicting concerns are identified by a match-point.
- *Dominant Aspect*: It identifies a concern with higher priority – used for resolving conflicts.
- *Composition Rules*: It includes a sequential list of simpler compositions for crosscutting concerns, some operators, and model elements.

The method has three main activities:

1. **Identify concerns**
2. **Specify Concerns and discover which of the concerns are crosscutting** (i.e., candidate aspects)
3. **Compose crosscutting concerns with other concerns** (uses match-points and composition rules, defined to them)

6.2.2 Composition Process for AOR Main Activities

The following is a description of the method's main activities, based on [Brito 03].

6.2.2.1 Identify Concerns

In this step, the system concerns (requirements) are identified, both functional and non-functional. This can be performed using any known method, with no specific approach being used here.

6.2.2.2 Specify Concerns and Identify Candidate Aspects

This step starts with specifying the concerns and ends with identifying which of them are crosscutting (i.e., candidate aspects). Functional concerns can be specified by using a set of scenarios, sequence diagrams, etc. For each of the non-functional concerns, the method assigns the following attributes:

- *Name*: The name of the non-functional concern.
- *Description*: A short description.
- *Priority*: The importance of the concern. It may take any of these values: Very Important, Important, Average, Low, and Very Low. This helps in conflicts resolution.
- *Decomposition*: It explains how concerns can be decomposed into simpler ones.
- *Where*: It is a list of models and their elements (e.g., use cases, classes, sequence diagrams) that require the concern. It helps in identifying match-points.
- *Contribution*: It describes how a concern affects other concerns. It can be positive (+) or negative (-) and helps identify conflicts.

6.2.2.3 Compose Candidate-Aspects with Concerns

The goal of this activity is to integrate candidate aspects with the concerns it cuts, in order to obtain the whole system. The main steps guiding the composition are:

1. **Identify how each candidate aspect affects the concerns it cuts across**: The following composition rules *crosscut-operators* are used to determine the type of conflict. The operators are similar to those used by other aspect-oriented methods for aspectual actions (e.g., see [Laddad 03] for this use in AspectJ):
    - ***Overlap*** (*Before* or *After*) – The candidate aspect is applied before or after the concerns it traverses.
    - ***Override*** - The behavior of a candidate aspect replaces the behavior of the concern it traverses.

- ***Wrap*** (Around) - The candidate aspect "encapsulates" the concern it traverses.

2. **Identify *match-points:*** It is based on the "Where" attributes of different concerns. This step identifies the match-points where the composition will occur. Note that match-points do not occur in the requirements, but rather in artifacts such as: use-cases, classes, and sequence diagrams.
   To represent match-points, the method uses a bi-dimensional table that lists the *Model Element* (**ME_i**) under study, and the stakeholders for the system. Each cell in the table may be filled with a list of *Candidate Aspects* (**CA_i**) that affect each Model Element. Each filled cell represents a *match-point* (**MP_i**).
3. **Identify conflicts between candidate aspects:** It is based on the "Contribution" attribute of concerns. The identification of conflicts results (from identified compositions) is required. More than one candidate's aspects (applied to the same match-point) may conflict with each other.
4. **Identify the *dominant aspect:*** It is based on "Priority." This step is used to resolve the conflicts identified in the previous step, by prioritizing candidate aspects and identifying dominant candidate aspects.
5. **Identify *composition rules:*** It is based on the previous step. In this last step, the actual composition rules are defined, including which candidate aspects will be used (i.e., which of them will be used as aspects), where and how.

6.2.3 Input Requirements Analysis using Composition Process for AOR

This section evaluates the use of the Composition Process for AOR to analyze the input requirements defined in Chapter 4. The primary (additional) approaches introduced by this method (versus MCAR described in Section 6.1) are:
1) Identifying match-points between elements of the model and
2) Using crosscutting-operators (Overlap, Override, Wrap).
Since the input for this work is already defined requirements, and the output are also requirements, the match-points only will be identified between requirements.

Crosscutting-operators will also be used between requirements, usually between the Baseline and Crosscutting requirements.

6.2.3.1 Identify Concerns

This step was already performed as part of the requirements specifications in Chapter 4, by identifying the crosscutting (aspectual) requirements.

6.2.3.2 Specify Concerns and Identify Candidate Aspects

Since the match-points (that should be identified in this work) are already in the requirements, and not in the artifacts of later development phases, not all attributes (defined by the methods for crosscutting requirements) are applicable:

- *Where* is not used as defined; it is related to artifacts used in later phases. Instead, I used a similar idea to combine the *contribution* attribute to a list of requirements that other requirements crosscut.
- *Decomposition* is not used as the requirements defined in this work. It is not split between other requirements (although in practice, this may be required in certain cases).
- *Name* and *Description* are part of the requirements definition, but are not important for the process itself.
- ***Priority*** is used to identify the relative importance of requirements.
- ***Contribution*** is used to identify whether the crosscutting requirement affects the requirement it cuts, negatively or positively. A positive effect usually causes an extension of the original requirement. A negative effect is a conflict and usually causes the contribution to give up requirements (in certain cases).

Part of this step was performed earlier as part of the requirements specifications in Chapter 4. Therefore, this analysis is partly produced by reverse engineering.
Table 3 defines the value for baseline attributes and crosscut requirements. Which requirements crosscut other requirements (candidate aspects) is not defined, because they are the same as in the evaluation by [Rashid 03], in Table 1.

Table 3 Requirements Attributes

Requirement	Baseline / Crosscut [Input]	Non-Conflicting Related Requirements (from Table 1 and Table 2)	Contribution of Conflicting Related Requirements (from Table 1 and Table 2)	Importance Priority [1-5]
Req-250	Base		610 (+), 730(+)	3
Req-260	Base		610(+), 740(+)	2
Req-270	Base		710(-)	3
Req-310	Base		520(+), 610(+), 710(+), 720(+)	4
Req-320	Base		520(+), 610(+), 710(+), 720(+)	4
Req-330	Base			4
Req-340	Base		540(+)	3
Req-520	Crosscut		610(-), 710(-), 720(-)	6
Req-540	Crosscut			5
Req-610	Crosscut	740	730(-)	6
Req-710	Crosscut	720		3
Req-720	Crosscut	710		3
Req-730	Crosscut			3
Req-740	Crosscut	610		2

The default value used for the *priority* attribute is “Average” or “Important,” depending on the importance of functionality to the user. A new “Critical” level, for highest priority, was added. This level is used for life threatening related requirements - Emergency and Tx Inhibit. Know that in TETRA, an Emergency call has a higher priority than a call to a police center from a cellular system (911 in the USA, 112 in Europe, etc.). This is similar to call priorities defined for TETRA (where there are four levels to an emergency call), but the highest is treated in a special way (because its intended use is only for life threatening situations).

For requirements where it is not critical for functionality to work in all situations (such as in de-registering from the system during power-off), the assigned priority is “Low.” For ease of use, numeric values identify the priorities (1 – Very Low, 2 – Low, 3 – Average, 4 – Important, 5 – Very Important, and 6 – Critical).

A list of "*Non-Conflicting Requirements*" was added to a column to assist in setting a requirement's priority. Using this information, the relative priority for the requirements is set while deciding the priority for each requirement.
The *Contribution* attribute is defined by the (+)/ (-) added to conflicting requirements; the (+)/ (-) indicate whether the conflicting requirements have higher or lower priority. Only requirements with higher priority are listed in this column, to prevent duplicate information. Note that the list of "conflicting related requirements," in the Contributions column, replaces the *Where* attribute defined in [Brito 03].

Note that not all conflicting requirements created new derived requirements. These are cases where the requirements priority is imposed by the system; therefore, there is no need to make a decision. For example, if the MS is out of coverage (Req-720), it cannot start a call (Req-310). In addition, related requirements that do not conflict (e.g., Req-610 and Req-740) do not impact priority. Because such a relation does not generate new derived requirements, they can (practically) be ignored, but only if first verified that this is indeed the case for this relation.

6.2.3.3 Compose Candidate-Aspects with Concerns

Following is the implementation of the different activities of this step for input requirements.

6.2.3.3.1 Identify how each candidate aspect affects the concerns it cuts

Based on Table 3, the crosscut-operators (composition rules) are added. Based on the operators, the priority was modified in some cases to agree with the operators. These results are summarized in Table 4. Because the priority is also relative to the related crosscutting requirements, *Priority* is called "*Relative Priority*" in Table 4.

Table 4 Requirements Attributes and Prioritization

Requirement	Baseline / Crosscut [Input]	Non-Conflicting Requirements (from Table 1 and Table 2)	Contribution of Conflicting Related Requirements (from Table 1 and Table 2)	Relative Priority [1-6]
Req-250	Base		610 (+ Override), 730 (+ Overlap After)	3
Req-260	Base		610 (+ Override), 740 (+ Override)	2
Req-270	Base		710 (- Override)	2
Req-310	Base		520 (+ Override), 610 (+), 710 (+), 720 (+)	4
Req-320	Base		520 (+ Override), 610(+ Override), 710 (+), 720 (+)	4
Req-330	Base			3
Req-340	Base		540 (+ Override)	4
Req-520	Crosscut		610 (- Override), 710 (- Override), 720 (- Overlap Before or Wrap)	6
Req-540	Crosscut			5
Req-610	Crosscut	740, 730		5
Req-710	Crosscut	720		3
Req-720	Crosscut	710		3
Req-730	Crosscut			3
Req-740	Crosscut	610		2

As seen, only related requirements (that have a negative relation) or conflicting requirements influence the priority. Requirements that are related positively (i.e., requirements that only add to each other and do not generate a derived requirement [see Table 2]), do not influence the priority. The priorities for the requirements were set manually, although in practice, algorithms (that create a partially ordered tree) can be used for this task.

6.2.3.3.2 Identify *Match-Points*

No activity was performed in this work, because the equivalent activity is identifying which requirement crosscuts other requirements. This step has already been performed.

6.2.3.3.3 Identify *Conflicts* between candidate aspects

This analysis has already been performed as part of building Table 3 and Table 4. In this work, the base and crosscut requirements are already input to the process, so it was possible to evaluate both using the same steps. In general, this may not be the case.

6.2.3.3.4 Identify the *Dominant Aspect* based on "Priority"

In this work, this analysis was already performed as part of previous activities.

6.2.3.3.5 Identify Composition Rules

In this last step, the actual composition rules are defined - including which candidate aspects will be used (i.e., which of them will be used as aspects), where, and how.

Based on the crosscutting-operators set for the *contribution* in Table 4, the composition rules are defined. According to [Brito 03], the composition rules format should be something like "Req-xxx <operator> Req-yyy". Since the derived requirements (in this work) are already known, the definition of the composition role also includes the derived requirement (specified in Section 4.6) resulting from the rule. Also included is an explanation of how the derived requirements were derived, using the composition rules. (Note that while preparing this work, some refinements were made to the derived requirements in Section 4.6, because of insights gotten from having to define and use the composition rules.)

In the following cases, although there is a conflict between requirements, in practice the conflict can (should) not happen. Therefore for these cases, no derived functional requirements are needed. Note however; for **robust and safe system** implementation, it may be useful to add such requirements (in case the "impossible" case does happen [because of a software bug, etc.]):

- **Req-710** overrides **Req-320**: This cannot happen because MS in *Normal* mode cannot be in a call while *unregistered*.
- **Req-720** overrides **Req-320**: This cannot happen because MS cannot be in a *call* while *out of coverage*.

Note the influence of requirements priorities to the operation of the composition rule. The higher priority requirement changes the behavior of the lower priority requirement. Therefore, in some cases, the base requirement practically crosscuts the crosscutting-requirement (e.g., Req-250 crosscuts Req-730), or the crosscutting-requirement crosscuts another crosscutting-requirement (e.g., Req-520 crosscuts Req-61).

The above evaluation is summarized in the following table for derived requirements (the description for derived requirements is the same as defined in Section 4.6):

Table 5 Derived Requirements

Base Requirement (and **Priority**)	**Crosscutting Requirement** (and **Priority)**	**Composition Rule**	**Derived Requirement**
Req-250 (3)	Req-610 (5)	Override	Req-1340
Req-250 (3)	Req-730 (3)	Wrap or Overlap After	Req-1120
Req-260 (2)	Req-610 (5)	Override	Req-1330
Req-260 (2)	Req-740 (2)	Override	Req-1130
Req-710 (3)	Req-270 (2)	Override	Req-1110
Req-310 (4)	Req-520 (6)	Override (see also below)	Req-1230
Req-310 (4)	Req-610 (5)	Override	Req-1320
Req-310 (4)	Req-710 (3)	Override	Req-1410
Req-320 (4)	Req-520 (6)	Override	Req-1240
Req-320 (4)	Req-610 (5)	Override	Req-1320
Req-340 (4)	Req-540 (5)	Override	Req-1250
Req-610 (5)	Req-520 (6)	Override Temporarily (see below)	Req-1310
Req-710 (3)	Req-520 (6)	Override	Req-1210
Req-720 (3)	Req-520 (6)	Wrap or Overlap After	Req-1220
Req-1120 (Crosscutting of Derived requirement)	Req-610	Override (see below)	Req-1340

Evaluating the composition rules, used earlier to define the derived requirements, raises some issues:

- In most cases, “override” is often used while “wrap” and “overlap” are barely used. The reason seems to be that crosscutting requirements mainly relate to

"mode" changes, which change the functionality of baseline requirements. "Wrap" and "overlap" are used mainly when additional functionality is required (e.g., checking security and logs).

- Req-1120: this is a delayed activation of Req-250 (registration when getting into coverage). If Req-730 exists, there is no registration when out of coverage. This seems to be a special case for crosscutting a FR. It deserves composition rules like "*delayed after*" and "*when mode change*":

  Req-250 *is delayed after* Req-730, *when the mode changes* to in-coverage.

- Req-1230: Req-520 changes the behavior of Req-310 (from *Normal* to *Emergency* Mode), but does not override it. This may require a composition rule such as "*modify*".

- Req-1340: Req-610 *overrides temporarily* Req-250 until *TXI* is off (entering a TXA Mode). This example implies that the impact of crosscutting functional requirements (to each other) is also state-based (in the sense that it does not change functionality), but defers or moves the functionality to a later (maybe earlier?) state or time. This suggests that composition rules should be enhanced with some kind of temporal rules.

- As also seen in Table 1 and in Table 2, Req-520 and Req-610 override many of the other requirements. In both cases, this is because they both significantly effect the transmission of the MS (which is the primary action in the requirements set defined in this work). (Req-610 prevents Tx and Req-520 allows Tx; when Tx is normally not allowed, it can change its priority.) While Req-610 mainly interacts with baseline requirements, Req-520 also interacts with crosscutting requirements. This is because it was decided (according to stakeholders' needs) that Req-520 has a higher priority than the crosscutting requirements it interacts with.

6.2.4 Applicability of Composition Process for AOR to create the Derived Requirements

The techniques used by this method are very applicable to the type of crosscutting functional requirements defined in this work. Not all of the steps are relevant, because the requirements are already detailed and not given initially as a level of viewpoints, use-cases, or similar level. Because of these reasons, some changes to the steps should be made.

6.3 Theme and Theme/Doc - Finding Aspects in Requirements

The *Theme approach* described in [Baniassad 04a] is a method and set of tools developed for the early identification of aspects in the software development life cycle.

6.3.1 Overview

The *theme* notion represents a system feature. Themes can be either *base themes* (which may share some behavior structure with other base themes), or *crosscutting themes* (*aspects*) which have a behavior that overlays base themes functionality.

The tools (developed to support this approach) include Theme/Doc and Theme/UML. Theme/Doc is used at the requirements level and provides views of the requirements specification text, thus exposing the relationship between behaviors in a system. Theme/UML is used at the design level; it allows a developer to model features and aspects of a system, and specifies how they should be combined. A central idea in the method is that Theme/Doc allows the developer to refine requirements views (in order to reveal which system functionality is crosscutting, and where in the system it crosscuts). Another claim is that the Theme approach helps maintain traceability from requirements to design, because the requirements map directly to Theme/Doc views (which map directly to Theme/UML models). This traceability also provides clues regarding requirements coverage in the design.

The Theme/Doc approach and tool is used to view the relationship between behaviors in requirements documents, and to **identify and isolate aspects** in the requirements. In other words, it is used mainly to identify and separate aspects (or the Separation of Concerns - SoC) from the requirements, but **not to combine aspects with the other requirements**. The approach provides views of requirements specs, and exposes relationships (interactions in the case of aspectual FRs) between behaviors in the system. The method helps to determine which elements of functionality are "base" and which are "aspects."

The Theme/Doc approach assumes that if **two behaviors** are described in the same requirement, they are **related**. According to this approach, there are three ways that behaviors can relate to each other (note that the method refers to identifying *related-behaviors*, and not *related-requirements* which is the main subject of other methods):

- *Erroneously/coincidentally*: In this case, requirements can be re-written so that behaviors are not coupled.
- *Hierarchically*: One behavior is a subset of the other, and there is no crosscutting relationship between them.
- *Crosscutting*: One behavior is an aspect of the other.

The Theme/Doc tool provides views that expose which behaviors are co-located in the requirements, in order to help determine the kind of relationships existing between behaviors.

6.3.2 Theme/Doc Approach Major Steps

The major steps in using the Theme/Doc approach are:

1. Identifying and listing the ***Actions*** used in the requirements. This step is performed manually. Usually the list of actions is pre-defined, based on experience from previous projects. This list of actions is a combination of actions

known from previous projects, and actions identified by reviewing current project requirements and choosing sensible verbs.

2. Creating an ***Action View***. For each action, it is used to show the requirements that use the action. The view also highlights the relationships between actions. This is performed using the Theme/Doc tool to perform a lexical analysis of the requirements text (using the actions list defined earlier).
3. Identifying **crosscutting actions** (aspectual actions) **and entities** being used, and removing non-crosscutting actions. Note that this method identifies crosscutting (aspectual) *actions*, and does not identify aspectual *requirements* (as done by other methods).
4. Creating a **Clipped Action View.** It shows the crosscutting hierarchy between actions. Insights acquired here (regarding the actions) are fed back to enhance the requirements and actions list.

The following steps are performed as part of the design phase, using Theme/UML. These steps are not evaluated in this work, because they are already part of the design phase:

5. Creating a ***Theme View*** to model the *themes* identified in the previous steps, for each of the crosscutting actions.
6. The Theme/UML is used to **incorporate crosscutting actions and identified entities into the design** as classes, methods, etc.
7. The ***Theme Views*** are then ***augmented*** to help verify the design choices made to align with the requirements.

The primary goal of the Theme/Doc approach is, therefore, to **identify which of the *actions* are *themes***. As described earlier, the actions are given as an input to the process. Although it may be possible to automatically identify actions from requirements, it was found that using actions as input is a good starting point for finding themes. Also, requirements that don't seem to include any actions can often be refined to include them. Because a product performs almost the same action for all of its releases, once the action list is defined, it can be reused from release-to-release with relatively minor enhancements.

6.3.3 Input Requirements Analysis using Theme and Theme/Doc

This section evaluates the use of Theme/Doc method to analyze the input requirements defined in Chapter 4. This method can identify aspects from interrelated behaviors of FRs, and not just aspects of a NFR (as most other methods identify). Therefore, it seems highly applicable for the type of requirements defined in this work. In this work, only techniques used for the Theme/Doc tool (for finding aspects in requirements) are used (according to the description by [Baniassad 04b]). The reason being that this work only deals with the requirements phase; it does not continue into the design phase. The Theme/Doc tool itself was not used.

To simplify the evaluation, only the requirements subset (defined in Chapter 4) is used. The subset includes requirements for initiating calls (for starting voice transmission) and emergency mode, or to crosscut the normal mode and calls. The subset does not handle incoming calls, power on/off, registration, and Tx inhibit.

6.3.3.1 Identifying Actions and Entities

The lists of actions and entities are generated by first identifying them in each requirement, and then generating the lists. Note that the lists can be made available from earlier releases of the product (as described earlier).

6.3.3.1.1 Identifying Actions and Entities per requirement

In this step, actions and *entities* are identified per requirement. The following conventions are used to mark actions (by underscore) and *entities (by italics).* In some cases, the text of the requirements is enhanced to clarify the analysis. In these cases, [additional words] that were added are marked in rectangular brackets. Also in rectangular brackets are comments about these additional words [comments about added words].

Req-310: **Pressing *PTT** [button]* **in *Idle* mode** shall initiate a request for an outgoing group call, with Normal Priority, to the predefined *Normal* group. If acknowledged by the *system*, *MS* shall toggle to *Call* mode and may start transmitting voice.

[It was decided not to add the word "button," but assume it for referencing to button values as a reference to the button entity.]

Req-320: **Pressing *PTT** [button]* **in *Call* mode** shall cause the MS to ask the system for permission to Tx voice, when no one else is transmitting in the call. The *MS* may start to Tx voice only if allowed by the *system*. The *PTT [button]* [press] shall be ignored when someone else is already transmitting in the call.

[PTT also refers to "press," because the name "Push-To-Talk" should probably add "press" to all references to PTT; PTT is a name, not an action. "PTT ignore" may not be an action, but a constraint. See also similar issues for "always" in Req-520 and Req-710.]

Req-520: **Pressing *PTT** [button]* **in *Emergency* mode** shall **always** allow the *MS* to initiate a call, as soon as possible, to the *Emergency group* with *Emergency priority*.

["Always" is more likely a kind of constraint of a modal operator, which may be the subject of a later work in analyzing requirements that use temporal logic.]

Req-710: **While *MS* is *unregistered***, no system related operation should be allowed by the *MS*.
[In this context, "system related operation" is translated to mean (see list of actions below): initiate call, ask Tx permission, Tx voice. Therefore, it is implied that the requirement refers to these actions.]
[As for "always" in Req-520, it is not clear what is "allowed". It may require the use of modal logic, or should be replaced by a "no call initiation" action.]

Note that the use of modal operators, such as: "always", "allowed" are problematic in this method. The method developed in this work partially solves these issues (see Chapter 7), although further evaluation is needed, such as the use of temporal logic methods.

6.3.3.1.2 Actions Identified

The actions identified in the previous step are listed below. In practice, this list would also be the basis for a pre-defined actions list, used for future releases of the product. Note that in some cases, the use of an action by a requirement is implied; it is not used directly. For example, "Initiate Call" is implied for Req-520 because "Press PTT in emergency mode" implies initiating a call. Another issue is the use of "not."
These phenomena are handled in the DRAS method developed in this work (see Chapter 7).

- **Press PTT** [button]: Req-310, Req-320, Req-520
- **Ignore PTT Press**: Req-320
- **Initiate** [outgoing group] **Call**: Req-310, Req-520 (implied), Req-710 (implied)
- **(toggle) Call Mode** [from Idle] **to Active**: Req-310
- **Ask** [the System for Voice] **Tx Permission**: Req-320, Req-520, Req-710 (implied)
- **(toggle) Call State** [from Rx] **to Tx**: Req-320, Req-520 (implied?)
- **Tx Voice**:: Req-310, Req-320, Req-520 (implied?), Req-710 (implied)

Note that the actions were rephrased (in some cases) to make the specification clearer and more precise. It may be useful to restate the requirements using these actions definitions. This approach will be evaluated as part of the method defined in this work. Also, note that some actions can have shorter definitions; for example: using "press" instead of "press PTT" in this context. However, because such definitions may not be unique (when other requirements are added), it is better to have a specific definition for the actions. On the other hand, it seems that in the TETRA MS context, "button" is redundant for "press" actions. Although in another context, "press PTT button" should have been used.

6.3.3.1.3 *Entities Identified*

The ***entities*** identified are shown below. Because the actions are usually related to specific entities, the actions (related to the entities) are also listed (to provide more information because these actions are often taken from all requirements). Note that the relationship between entities and their actions is similar to the methods defined for classes in the object-oriented method. Also listed are attributes related to the entity.

- ***MS*** entity
  - Actions: Power *On/Off*
  - Attributes: Priority Mode (*Normal/Emergency*), Call Mode (*Idle/Call*), Registration Mode (*Registered/Un-registered*)
- ***MS User*** entity (note that the MS user is not directly mentioned in the requirements, but its existence is implied)
  - Actions: Press <button> (<button> represents any of the MS's buttons)
  - Attributes: none
- ***Call*** entity
  - Actions: Initiation, Receiving
  - Attributes: Call Priority (*Normal/Emergency*), Call Direction (*Incoming/Outgoing*), Call State (*Tx/Rx*)
- ***PTT Button*** entity
  - Actions: Pressed
  - Attributes: none

- ***Emergency Button*** entity
  - Actions: Pressed
  - Attributes: none
- ***System*** entity
  - Actions: none (because the requirements are for the MS only)
  - Attributes: because the requirements are for the MS, all system attributes are reflected in MS related attributes for entities. Therefore, the system entity attributes are the related attributes from the system point of view: Call Mode (*Idle/Call*), Registration Mode (*Registered/Un-registered*), Call Priority (*Normal/Emergency*), System Call Direction (*Incoming/Outgoing*), System Call State (*Tx/Rx*).
    Note that the system Call Direction and Call State are opposite to the MS state (e.g., when the MS transmits, the system receives).

The entities identification is not evaluated any further or used in this work, because they are mainly used for the design phase.

6.3.3.2 Create Actions Views

The Actions View shows how actions are used by requirements. The Theme/Doc tool uses lexical analysis to generate these views. Here, the analysis was done manually. Two types of inputs are used: a list of actions (generated earlier [e.g., for a previous project] or in this case, as part of previous sections) and the list of requirements.

6.3.3.2.1 Actions View (Theme/Doc) – Actions by Requirements

The actions used by each the requirements are:

- Req-310: Press PTT (*Normal* Priority), Initiate Call (*Normal* Priority), Call Mode to Active (*Normal* Priority), Tx Voice.
- Req-320: Press PTT (*Normal* Priority), Ignore PTT Press, Ask Tx Permission (*Normal* Priority), Call State to Tx, Tx Voice
- Req-520: Press PTT (*Emergency* Priority), Initiate Call (*Emergency*), Ask Tx Permission (*Emergency*), Call State to Tx, Tx Voice.

- Req-710: Initiate Call (*Unregistered*), Ask Tx Permission, Tx Voice.

Figure-2 shows the Action view for requirements (the figure was created manually and without the Theme/Doc tool):

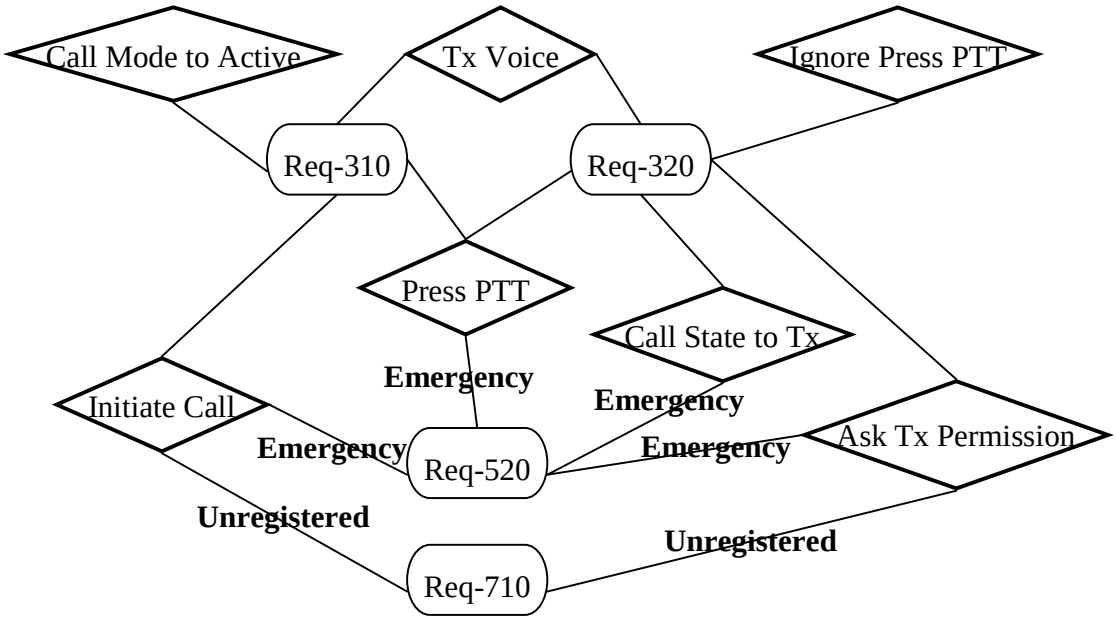

Figure-2 Action View for a Subset of Requirements

Note that attributes were added to some of the actions: Emergency Priority-Mode and Unregistered Registration-Mode. These modes make them crosscutting-actions for other requirements. Also note that adding attributes to the actions is currently not part of the Theme/Doc method, but without this information, the crosscutting nature of some of the requirements is not visible.

As seen in Figure-2, the Action View does not offer much help to identify crosscutting requirements and create derived requirements. The main purpose of Theme/Doc is identifying crosscutting actions. To create derived requirements, crosscutting requirements are the main issue. For this reason, more steps related to this method are not evaluated.

6.3.4 Applicability of Theme/Doc for creating a Derived Requirement

As understood from the previous paragraph, this method is not well suited for creating derived requirements. However, the method is useful for identifying actions and their attributes that make requirements crosscut. This approach is used as part of the DRAS method.

7 The DRAS Methodology

This chapter describes the *DRAS* (*Derived Requirements generation by Actions and States*) methodology for generating DRs from stakeholder requirements, as shown in Fig. 6. The requirements, defined in Chapter 4, are used as an example for this methodology. Prototype implementation scripts for some of the DRAS process steps were developed, and were used to produce some of the tables in this chapter. These scripts perform the following steps:

- **Identifying actions used by requirements** based on the manual actions and entities lists directly used by each action (actions or entities directly used by each requirement).
- **Identifying requirements-action attributes**.
- **Identifying match-points between requirements.**

The input to the scripts is the output of parsing the requirements, and the manually identified actions and attributes. The output of the scripts is the tentative match-points. Because of the fixed priorities per requirement, it is only possible to suggest which requirement is the crosscutting requirement. In general, no part of the requirements text analysis is automated.

7.1 Gathering the Stakeholders' Requirements

In this step, the stakeholders' requirements are gathered and formulated. Different methods may be used to gather and formulate requirements (see [Creveling 03] for details of some of these methods). The requirements defined earlier will be used to explain how the other steps of DRAS work.

7.2 Identifying Actions, Entities and Attributes

The lists of actions, entities, and attributes (mainly modes and states) for the system (TETRA MS in this work) are identified. The attributes are then used to analyze the requirements. The contents of some lists are used for all systems, while the contents of

other lists are specific to the system (TETRA MS, in this work). The following sections describe the different lists by DRAS.

7.2.1 General Lists for all Systems

Action Modifiers

This attribute defines whether the use of an action by a requirement is about restricting its use, or ease of other restrictions for its use. The action modifier is assigned separately per each action used by a requirement. As defined in section 3.3.4, possible values are:

- **NULL**
- **Unconditional**
- **Restrict**

Composition Rules

The way a crosscutting requirement affects the requirement it crosscuts is defined here. This is based on [Brito 03]. As defined in section 3.3.6, possible values are:

- **Overlap Before or After**
- **Override**
- **Wrap** (around)

Relative Priorities

This step defines Relative Priorities between requirements or attributes. It is also based on [Brito 03]. As defined in section 3.3.5, possible values are:

- **"+"**
- **Same**
- **"-"**

7.2.2 General Lists with Specific System Contents

Some of the lists defined in this section conform to ideas found in [Baniassad 04b] for identifying entities and actions in the requirements, and in [Brito 03] for identifying attributes in them.

Entities

This is the list of entities identified in the system. For system requirements of the TETRA MS example, identified entities are:

- Incoming Call
- MS
- Outgoing Call
- PTT
- System

Note that it is possible to add an *actors* list. In this case, MS User may be included in this list as the person who presses the PTT button. (Consequently, "Pressing PTT" could have been written as "MS User presses PTT".) To simplify the example, it was decided not to use the MS User actor in the requirements text.

Actions

Table 6 shows the list of actions as identified in the requirements.

Table 6 Actions List

Action	MS Initiated	Note
Ask Tx Permission	Yes	
Call Ack	No	Ack Initiate Call by the System
Call Mode to Active	Yes	
Call Nack	No	Nack Initiate Call by the System
Call State to Tx	Yes	
De-register	Yes	
Initiate Call	Yes	
Join Incoming Call	Yes	
Power Off	Yes	
Power On	Yes	
Press PTT	Yes	
Receive Incoming Call	No	
Register	Yes	
Tx	Yes	
Tx Control	Yes	
Tx Voice	Yes	

This is the list of functional actions used in the system. For each action, specific system attributes can be added. For the MS, one attribute is defined: whether the action is initiated by the MS, or by the TETRA system. A more general definition would be to define the *Initiating Entity f*or the action.

Modes and States

This is a list of possible modes and states, related to the entities used by the requirements. Each mode or state is also tagged as crosscutting or not. Non-crosscutting modes or states are used later in the analysis to identify aspectual requirements and their effect on other requirements. Table 7 is the list for the input requirements.

Table 7 Modes and States List

Mode	Crosscutting
Coverage_Mode	Yes
Registration_Mode	Yes
MS_Call_Mode	No
Call_State	No
Power_State	Yes
MS_Priority	Yes
Call_Priority	Yes
Tx_Mode	Yes

For each mode or state, the list of possible values is defined in Table 8.

Table 8 Modes and States Values

Mode	Value 1	Value 2	Value 3	Value 4
Coverage_Mode	In	Out		
Registration_Mode	Registered	Unregistered		
MS_Call_Mode	Idle	Call		
Call_State	Rx	Tx	No-Tx	No-Voice
Power_State	Power-Off	Powering-On	Power-On	Powering-Off
MS_Priority	Normal	Emergency		
Call_Priority	Normal	Emergency		
Tx_Mode	Allowed (TXA)	Inhibit (TXI)		

In addition, a table that lists contradicting pairs of modes values is generated. This table (Table 9) is later used to remove possible conflicts between requirements that usually cannot occur in reality. There are two types of such contradictions:

- Values of two different modes (the second row of Table 9 - unregistered MS that is in a call - MS cannot take part in a call without registering to the system first).

Table 9 Contradicting Pairs of Mode/State Values

Mode1	Value1	Mode2	Value2
Coverage	Out	MS_Call_Mode	Call
Registration_Mode	Unregistered	MS_Call_Mode	Call
Power_State	Power Off	Registration_Mode	Registered
Power_State	Poser Off	Coverage	In

7.3 Identifying Correlations between Actions and Entities

In this step, correlations among actions, among entities, and between actions and entities are identified. That is, which actions are used by each entity in the system and therefore, which entities are relevant for each action.

7.3.1 Entities used by Action

For each action, the list of entities used by that action is defined. It is later used to identify correlations between requirements that define functionality based on entities and requirements that define functionality based on related actions.

Table 10 summarize these relations for input requirements. An empty cell indicates that there is no relation, while Yes/No indicates whether an entity is always used by the action. For example, the system is always a part of call initiation, because all call traffic must go through the system. On the other hand, the system will not be involved after pressing PTT, if someone else is already talking.

Table 10 Entities used by Actions

Action	MS	System	PTT	Outgoing Call	Incoming Call
Ask Tx Permission	Yes	Yes			
Call Ack	Yes	Yes		Yes	
Call Mode to Active	Yes	Yes		Yes	Yes
Call Nack	Yes	Yes		Yes	
Call State to Tx	Yes	Yes		Yes	Yes
De-register	Yes	Yes			
Initiate Call	Yes	Yes		Yes	
Join Incoming Call	Yes	Yes			Yes
Power Off	Yes	No			
Power On	Yes	No			
Press PTT	Yes	No	Yes	Yes	
Receive Incoming Call	Yes	Yes			Yes
Register	Yes	Yes			
Tx	Yes	Yes			
Tx Control	Yes	Yes			
Tx Voice	Yes	Yes		Yes	Yes

Note that the MS entity is used by all of the actions, because the requirements are for the MS. Therefore, the MS entity is practically redundant in the analysis of this specific set of requirements.

7.3.2 Actions Directly Implied (used) by Action

For each action, the list of actions that they directly imply (i.e., that are directly used by the action) are defined. These relations between actions are later used to identify all actions that are used by the requirements, whether directly or indirectly.
Table 11 presents these relations. The Must column indicates whether the implied action must be used (e.g., Registration during Power-On is not a must).

Table 11 Actions Implied by Action

Action	Implied_Action	Must	Note
Initiate Call	Call Ack	Yes	Ack by the System
Initiate Call	Call Nack	Yes	Nack by the System
Power On	Register	No	
Power Off	De-register	No	
Register	Tx Control	Yes	
De-register	Tx Control	Yes	
Press PTT	Initiate Call	Yes	
Press PTT	Ask Tx Permission	Yes	
Initiate Call	Call Mode to Active	Yes	
Initiate Call	Tx Control	Yes	
Initiate Call	Ask Tx Permission	Yes	
Receive Incoming Call	Join Incoming Call	Yes	
Join Incoming Call	Call Mode to Active	Yes	
Ask Tx Permission	Tx Control	Yes	
Ask Tx Permission	Call State to Tx	Yes	
Call State to Tx	Tx Voice	Yes	
Tx Control	Tx	Yes	
Tx Voice	Tx	Yes	

7.3.3 Actions used by Action

For each action, the list of actions that it uses (either directly or indirectly) is defined. The list is generated from the list of actions directly implied (Table 11). These relations between actions are later used to correlate between requirements, based on the use of different but related functional actions. For example, Req-610 specifies that when MS is in the *TXI* (Tx Inhibit) mode, it should not transmit. To understand which actions are affected by Req-610, all actions that may result in transmission should be identified. The table is created by recursively identifying the actions that are used by an action, based on the previously identified, implied actions. The initial values are taken from Table 11.

Pseudo code to generate the “Actions Used by Action” table:

```
From each record in “Actions Implied by Action” (Table 11):
```

```
        Create record in “Actions Used by Actions” table with
            Action, Implied Action, Must
    End From
    Repeat until no new record is added (no duplicates):
        Foreach two records in “Actions Used by Actions”,
                were R1.Implied_Action = R2.Action:
            Create record in “Actions Used by Actions” table with
                R1.Action, R2.Implied_Action, Must
        End Foreach
    End Repeat
```

Table 12 summarizes which actions are used by each action.

Table 12 Actions used by Action Summary

	Implied Actions											
Action	**Ask Tx Permission**	**Call Ack**	**Call Mode to Active**	**Call Nack**	**Call State to Tx**	**De-Reg.**	**Initiate Call**	**Join Income Call**	**Reg.**	**Tx**	**Tx Control**	**Tx Voice**
Ask Tx Permission					Y					Y	Y	Y
Call State to Tx										Y		Y
De-register										Y	Y	
Initiate Call	Y	Y	Y	Y	Y					Y	Y	Y
Join Incoming Call			Y									
Power Off						N				N	N	
Power On									N	N	N	
Press PTT	Y	Y	Y	Y	Y		Y			Y	Y	Y
Receive Incoming Call			Y					Y				
Register										Y	Y	
Tx Control										Y		
Tx Voice										Y		

7.4 Identifying Actions and Entities used by the Input Requirements and their Priorities

In this step, the actions and entities that are directly used by input requirements are identified. For each action and entity, the appropriate *modes* and *states* are also identified. In addition, the relative *priorities* of the requirements are set.

Table 13 Input Requirements Split - use of Actions and Entities

Req. Num	Part Num	Sub Requirement	Action Modifier	Action	Entity
Req-250	**1**	On power-on		Power On	
Req-250	**2**	MS shall register to the system		Register	
Req-260	**1**	On power-off		Power Off	
Req-260	**2**	MS shall de-register first from the system, if it is successfully registered		De-register	
Req-270	**1**	MS shall be able to power-off in any state		Power Off	
Req-310	**1**	Pressing PTT in Idle mode shall		Press PTT	
Req-310	**2**	initiate a request for outgoing group call to the system, with Normal Priority to the predefined Normal group.		Initiate Call	
Req-310	**3**	If acknowledged by the system		Call Ack	
Req-310	**4**	MS shall toggle to Call mode		Call Mode to Active	
Req-310	**5**	and may start transmitting voice		Tx Voice	
Req-320	**1**	Pressing PTT in Call mode		Press PTT	
Req-320	**2**	shall cause MS to ask the system for permission to Tx voice, when no one else Tx in the call.		Ask Tx Permission	
Req-320	**3**	If allowed by the system		Call Ack	
Req-320	**4**	MS may start Tx		Tx Voice	
Req-320	**5**	The PTT shall be ignored when someone else already Tx in the call.	Ignore	Press PTT	
Req-330	**1**	When receiving incoming Group call in Idle mode		Receive Incoming Call	
Req-330	**2**	MS shall toggle to Call mode		Call Mode to Active	
Req-330	**3**	and join the call.		Join Incoming Call	
Req-340	**1**	When receiving incoming Group call in Call mode,		Receive Incoming Call	
Req-340	**2**	MS shall internally reject the call.	Ignore	Join Incoming Call	
Req-520	**1**	Pressing PTT in Emergency mode	Uncond.	Press PTT	
Req-520	**2**	shall always allow the MS to initiate a call, as soon as possible, to the Emergency group with Emergency priority.	Uncond.	Initiate Call	
Req-540	**1**	When received Incoming Call with Emergency priority	Uncond.	Receive Incoming Call	
Req-540	**2**	MS shall join the call if not in Emergency call	Uncond.	Join Incoming Call	
Req-610	**1**	When in TXI Mode, MS shall ignore any request to transmit	Ignore	Tx	
Req-710	**1**	While MS is unregistered, no system related operations should be allowed by the MS	Ignore		System
Req-720	**1**	When MS is out of coverage, pressing PTT shall be ignored	Ignore	Press PTT	
Req-730	**1**	When MS power-on while it is out of coverage, it should not try to register	Ignore	Register	
Req-740	**1**	When MS is out of coverage, MS shall not try to transmit	Ignore	Tx	

7.4.1 Split Requirements Text per Action/Entity (Manual)

In this step, each input requirement is split (manually) into parts, based on *actions* and *functional entities* used by it (the Actions and Entities were defined in the lists described earlier). Functional entities are the entities that the requirements directly refer to, as part of specifying the functionality. Each record in the table includes one action or entity. An important attribute (defined per action and entity) is the *action modifier*: *Restrict* or *Unconditional* (see Section 3.3.4). It is later used to identify the implied actions and to help decide whether a requirement crosscuts. Table 13 shows the split of input requirements.

Note that in this set of input requirements, only Req-710 is specified by entity (the "System" entity) and not by action.

7.4.2 Attributes (Modes and States) of Requirements Parts

For each part of a split requirement, the related attributes, modes and states are identified. This helps us to identify the concerns and aspectual parts of the requirements. Table 14 shows the split requirements table with the attributes for the input requirements.

Table 14 Requirements Attributes of Actions and Entities

Req. Num	Part #	Action Modifier	Action	Entity	Coverage	Reg. Mode	MS Call Mode	Call State	Power State	MS Prio.	Call Prio.	Tx Mode	Comments
Req-320	**5**	Ignore	Press PTT		NA	NA	Call	Rx	NA	NA	NA	NA	
Req-250	**1**		Power On		NA	Unreg.	NA	NA	Off	NA	NA	NA	
Req-250	**2**		Register		NA	Unreg.	NA	NA	Power On	NA	NA	NA	
Req-260	**1**		Power Off		NA	NA	NA	NA	On	NA	NA	NA	
Req-260	**2**		De-register		NA	Reg.	NA	NA	Power Off	NA	NA	NA	
Req-270	**1**		Power Off		NA	NA	NA	NA	NA	NA	NA	NA	
Req-310	**1**		Press PTT		NA	NA	Idle	NA	NA	NA	NA	NA	
Req-310	**2**		Initiate Call		NA	NA	Idle	NA	NA	NA	Normal	NA	
Req-310	**3**		Call Ack		NA	NA	Idle	NA	NA	NA	NA	NA	

Req. Num	Part #	Action Modifier	Action	Entity	Coverage	Reg. Mode	MS Call Mode	Call State	Power State	MS Prio.	Call Prio.	Tx Mode	Comments
Req-310	**4**		Call Mode to Active		NA	NA	Call	Not Tx	NA	NA	NA	NA	
Req-310	**5**		Tx Voice		NA	NA	Call	Tx	NA	NA	NA	NA	
Req-320	**1**		Press PTT		NA	NA	Call	NA	NA	NA	NA	NA	
Req-320	**2**		Ask Tx Permission		NA	NA	Call	No Voice	NA	NA	NA	NA	
Req-320	**3**		Call Ack		NA	NA	Call	No Voice	NA	NA	NA	NA	
Req-320	**4**		Tx Voice		NA	NA	Call	Tx	NA	NA	NA	NA	
Req-330	**1**		Receive Incoming Call		NA	NA	Idle	NA	NA	NA	NA	NA	
Req-330	**2**		Call Mode to Active		NA	NA	Call	NA	NA	NA	NA	NA	
Req-330	**3**		Join Incoming Call		NA	NA	Call	NA	NA	NA	NA	NA	
Req-340	**1**		Receive Incoming Call		NA	NA	Call	NA	NA	NA	NA	NA	
Req-340	**2**	Ignore	Join Incoming Call		NA	NA	Call	NA	NA	NA	NA	NA	Reject = Ignore Request
Req-520	**1**	Uncond.	Press PTT		NA	NA	NA	NA	NA	Emergency	NA	NA	
Req-520	**2**	Uncond.	Initiate Call		NA	NA	NA	NA	NA	Emergency	NA	NA	Always = Unconditionally
Req-540	**1**	Uncond.	Receive Incoming Call		NA	NA	NA	NA	NA	NA	Emergency	NA	
Req-610	**1**	Ignore	Tx		NA	NA	NA	NA	NA	NA	NA	TXI	No Tx = Ignore all related activities
Req-710	**1**	Ignore		System	NA	Unreg.	NA	NA	NA	NA	NA	NA	All actions related to System Entity
Req-720	**1**	Ignore	Press PTT		Out	NA	NA	NA	NA	NA	NA	NA	Ignore
Req-730	**1**	Ignore	Register		Out	Unreg.	NA	NA	Power On	NA	NA	NA	No try to Register = Ignore Request to Register
Req-740	**1**	Ignore	Tx		Out	NA	NA	NA	NA	NA	NA	NA	No Tx = Ignore all Requests to Tx
Req-540	**2**	Uncond.	Join Incoming Call		NA	NA	NA	NA	NA	NA	Emergency	NA	

7.4.3 Requirements Priorities

Relative priorities for the requirements are now defined. The relative priorities help to identify which is the crosscutting requirement. Usually, a crosscutting requirement cannot have a priority lower than the requirement it crosscuts.

For the requirements defined in this work, priority levels were set as 1-6 (where 6 is the highest priority). Level 6 is used only for emergency related requirements. Level 5 is used for requirements that are related to forced conditions, such as not being able to transmit to the system when MS is out of coverage.

7.5 Identifying Actions used by the Requirements

In this step, the list of all actions used by the requirements, whether directly or indirectly, are identified. The purpose is to verify that when checking whether a requirement crosscuts another requirement, all the common actions (referred to by the requirements) are taken into account. The action-modifier is the main attribute used to create the list of actions (used by the requirements).

The following steps are performed to generate the list:

1. The list of entities used by the requirements is generated from the Split Requirements table (Table 11), including action-modifier attributes. For the requirements given here, the table includes only one row.

   Pseudo code to generate the “Entities Used by Requirement” table:

```
Foreach record in “Requirements Split” table:
    If Entity is not Null
        Add record to “Entities Used by Requirement” table
            with Requirement_Number, Entity,
                 Action_Modifier, Direct_Use=”Entity”
End Foreach
```

Requirement Number	Entity	Action Modifier	Direct Use
Req-710	System	Restrict	Entity

Table 15 Entities used by Requirements

2. Based on the entities used by each action (Table 10), the list of actions indirectly used by the requirements is generated according to the entities used by the requirements. The attribute “Direct Use” is added with the “Entity” value; this means the requirement is using the action through an entity it uses. Note that since the table below is based on one requirement (Req-710), the “Action Modifier” and “Direct Use” attributes are identical for all of the requirements.
3.

Pseudo code to generate the “Actions Used by Entity” table:

```
Foreach record in “Entities Used by Requirement” as EUR:
    Foreach record in “Actions Related Entities” as ARE,
            with Entity = EUR.Entity:
        Add record to “Action Used by Entity” table with
            EUR.Requirement_Number, ARE.Action,
            EUR.Action_Modifier, EUR.Direct_Use
    End Foreach
End Foreach
```

Requirement_Number	Action	Action_Modifier	Direct_Use
Req-710	Power Off	Ignore	Entity
Req-710	Power On	Ignore	Entity
Req-710	Register	Ignore	Entity
Req-710	Tx	Ignore	Entity
Req-710	De-register	Ignore	Entity
Req-710	Initiate Call	Ignore	Entity
Req-710	Call Mode to Active	Ignore	Entity
Req-710	Receive Incoming Call	Ignore	Entity
Req-710	Ask Tx Permission	Ignore	Entity
Req-710	Tx Control	Ignore	Entity
Req-710	Join Incoming Call	Ignore	Entity
Req-710	Call State to Tx	Ignore	Entity
Req-710	Press PTT	Ignore	Entity
Req-710	Tx Voice	Ignore	Entity

Table 16 Actions per Entity

4. At this step, the list of actions directly used by the requirements is generated, along with the “Direct Use” attribute (where the requirements directly use the actions [only part of the list is shown]).

   Pseudo code to generate the “Actions Used by Requirement” table:

```
Foreach record in “Actions Used by Requirement” as EUR:
    Foreach record in “Actions Related Entities” as ARE,
            with Entity = EUR.Entity:
        Add record to “Action Used by Requirement” with
            Requirement_Number, Action, Action_Modifier,
            Direct_Use = “Yes”
    End Foreach
End Foreach
```

Requirement_Number	Action	Action_Modifier	Direct_Use
Req-320	Press PTT	Ignore	Yes
Req-250	Power On		Yes
Req-250	Register		Yes
Req-260	Power Off		Yes
Req-260	De-register		Yes
Req-270	Power Off		Yes
Req-310	Press PTT		Yes
Req-310	Initiate Call		Yes
Req-310	Call Ack		Yes
Req-310	Call Mode to Active		Yes
Req-310	Tx Voice		Yes
Req-320	Press PTT		Yes
…			

Table 17 Actions Directly used by the Requirements (excerpt)

5. The list of actions used by each requirement according to the entities each requirement uses (which was generated earlier), is now appended to the “Actions used by Requirements” table (see Table 18).

6. For each action *Act* which is *restricted* by the requirement (according to the action-modifier), the use of all actions implying the use of *Act* may also need to be restricted. For example, if the MS should not transmit when transmit is Inhibited (*TXI* mode), all actions that result in transmission may also need to be ignored. Therefore, all of these implied actions are added to the list. A "Direct Use" attribute for these added actions is set to "Restrict", indicating that they were added because of the "Restrict" attribute of their implying action. The Action Modifier for these actions is set to "Restrict" as the implying action. This is performed using Table 12 (actions used by each action). Pseudo code to add the "Ignored" using actions to "Actions Used by Requirement" table:

```
Foreach record in "Actions Used by Requirement" as AUR,
        with Action_Modifier = "Restrict":
    Foreach record in "Actions Used by Action" as AUA,
            with Action = AUR.Action:
        Add record to "Action Used by Requirement" with
            AUR.Requirement_Number, AUA.Used_By_Action,
            AUR.Action_Modifier, Direct_Use="Restrict"
    End Foreach
End Foreach
```

7. For each action *Act* which is performed *unconditionally* (according to the action-modifier), all actions that are used by *Act* may also have to be allowed. For example, if the MS should be allowed to initiate a call in *Emergency* mode, then it should also be allowed to transmit. Therefore, all actions used by unconditional actions are added to the list. A "Direct Use" attribute for these added actions is set to "Implied", indicating that they were added because of the "Unconditional" attribute of their implying actions. The action-modifier for these actions is set to "Unconditional" as the implying action. This is also performed using Table 12 (the table for actions used by each action).

   Pseudo code to add the "Ignored" using actions to "Actions Used by Requirement" table:

```
Foreach record in "Actions Used by Requirement" as AUR,
```

```
            with Action_Modifier is not “Restrict”:
        Foreach record in “Actions Used by Action” as AUA,
            with Action = AUR.Used_BY_Action:
        Add record to “Action Used by Requirement” with
            AUR.Requirement_Number, AUA. Action,
            AUR.Action_Modifier, Direct_Use=”Implied”
    End Foreach
End Foreach
```

Table 18 Actions Used By Requirements (excerpt)

Requirement_Number	Action	Action_Modifier	Direct_Use
Req-250	Power On		Yes
Req-250	Register		Yes
Req-250	Tx		Forward
Req-250	Tx		Forward
Req-250	Tx Control		Forward
Req-250	Tx Control		Forward
Req-260	Tx Control		Forward
Req-260	Power Off		Yes
Req-260	De-register		Yes
Req-260	Tx		Forward
Req-260	Tx Control		Forward
Req-260	Tx		Forward
Req-270	Power Off		Yes
Req-270	De-register		Forward
……			
Req-730	Power On	Ignore	Ignore
Req-730	Register	Ignore	Yes
Req-740	Power On	Ignore	Ignore
Req-740	Register	Ignore	Ignore
Req-740	Tx Control	Ignore	Ignore
Req-740	Press PTT	Ignore	Ignore
Req-740	Tx	Ignore	Yes
….			

The “Actions used by Requirements” table (Table 18), created by the steps above, shows part of the list for all actions used by the requirements, directly or indirectly. Note that

some rows may be created more than once, as a result of different requirements. These redundancies are removed in later steps.

7.6 Identifying Requirements-Actions Attributes

Now the Actions Used by Requirements table (see Table 18) is extended, by identifying each Requirement-Action pair *Req-Act*, showing in which conditions it is performed according to the requirement. The proper attributes for *Req-Act* (mainly modes and states) are identified based on the attributes of actions and entities table (Table 14). In the case that *Act* is not directly used by *Req*, the action *Act'* that caused *Act* to be related to *Req* and that is directly used by *Req* should first be identified. Then the row of the split table that is relevant to the *Req-Act* should be identified.

Pseudo code for the creation of the “Requirement Attributes” table:

```
For all records with the same Requirement Number
        from “Requirements” table as R
            “Requirements Split” table as RS
            “Actions Used by Requirement” table as AUR
        with the same Requirement Number
        and where
            (AUR.Action=RS.Action And AUR.Direct_Use="Yes")
            Or (RS.Action_Modifier="Unconditionally" And
                AUR.Direct_Use="Implied")
            Or (RS.Action_Modifier="Restrict" And
                AUR.Direct_Use="Restrict")
            Or AUR.Direct_Use="Entity"
    Add record to “Requirements Attributes” table with
        Requirements Number, R.Priority, RS.Action_Modifier,
        AUR.Action, all attributes from RS
End For
```

Following is part of the full Requirement-Action Attributes table for the input requirements (Table 19).

Table 19 Requirement-Actions Attributes (excerpt)

Req. Num	Priority	Action Modifier	Action	Coverage	Reg. Mode	MS Call Mode	Call State	Power State	MS Priority	Call Priority	Tx Mode
Req-250	5		Power On	NA	Unreg.	NA	NA	Off	NA	NA	NA
Req-250	5		Register	NA	Unreg.	NA	NA	Power On	NA	NA	NA
Req-260	5		De-register	NA	Reg.	NA	NA	Power Off	NA	NA	NA
Req-260	5		Power Off	NA	NA	NA	NA	On	NA	NA	NA
Req-270	5		Power Off	NA	NA	NA	NA	NA	NA	NA	NA
Req-310	4		Call Ack	NA	NA	Idle	NA	NA	NA	NA	NA
Req-310	4		Call Mode to Active	NA	NA	Call	Not Tx	NA	NA	NA	NA
Req-310	4		Initiate Call	NA	NA	Idle	NA	NA	NA	Normal	NA
Req-310	4		Press PTT	NA	NA	Idle	NA	NA	NA	NA	NA
Req-310	4		Tx Voice	NA	NA	Call	Tx	NA	NA	NA	NA
Req-320	4		Ask Tx Permission	NA	NA	Call	No Voice	NA	NA	NA	NA
Req-320	4		Call Ack	NA	NA	Call	No Voice	NA	NA	NA	NA
Req-320	4		Press PTT	NA	NA	Call	NA	NA	NA	NA	NA
Req-320	4		Tx Voice	NA	NA	Call	Tx	NA	NA	NA	NA
Req-320	4	Ignore	Press PTT	NA	NA	Call	Rx	NA	NA	NA	NA
Req-330	4		Call Mode to Active	NA	NA	Call	NA	NA	NA	NA	NA
Req-330	4		Join Incoming Call	NA	NA	Call	NA	NA	NA	NA	NA
Req-330	4		Receive Incoming Call	NA	NA	Idle	NA	NA	NA	NA	NA
Req-340	4		Receive Incoming Call	NA	NA	Call	NA	NA	NA	NA	NA
Req-340	4	Ignore	Join Incoming Call	NA	NA	Call	NA	NA	NA	NA	NA
Req-340	4	Ignore	Receive Incoming Call	NA	NA	Call	NA	NA	NA	NA	NA
Req-520	6	Uncond.	Ask Tx Permission	NA	NA	NA	NA	NA	Emergency	NA	NA
Req-520	6	Uncond.	Call Ack	NA	NA	NA	NA	NA	Emergency	NA	NA
……											

7.7 Identifying Match-Points between the Requirements

In this step, crosscutting requirements and the requirements they crosscut are identified. This is performed by identifying the ***match-points*** between requirements, using their common attributes. The common attributes are *actions*, *modes*, *states*, and other

attributes that are common to the requirements ([Brito 03] also adds *candidate aspects* to the match-points). This is based on ideas from [Brito 03] and [Rashid 03]. Also included are *priorities* for the requirements.

The table that identifies match-points is in many ways similar to the commonly used traceability matrix (between requirements and their sub-products), used during the development process. A major difference between these tables is that the traceability matrix is mainly used to trace between artifacts from different development phases, while match-points help to create DRs in the same development phase (the requirements phase).

The match-points are mainly identified by modes and states. This is because the requirements defined here are event-based (button press, incoming call, change mode, etc.). They define the MS main function in case these events happen. They hardly include specifications for additional, non-event-based functionality, such as Logging.

Match-point identification is performed in three steps:

1. Identify the list of match-point candidates between requirements, according to the use of common actions and different values for attributes.
2. Remove redundancies that were created by multiple matches between two requirements (mainly caused by several implied actions that match between requirements).
3. Remove match-points that cannot happen in reality (based on Table 9), or do not have at least one different crosscutting mode or State (based on Table 14Table 19).

These steps are described further below.

7.7.1 List of Match-Point Candidates between Requirements

In this step, all requirements using a common action are identified (based on the actions used by the requirements [Table 18] and the requirements attributes [Table 19]). The following criteria are used to match between requirements:

1. The same action is used - based on the actions used by the Requirements table and the Requirements Attributes table.
2. The priority of the crosscutting requirement is at least as high as the priority of the requirement it crosscuts.
3. The action is either directly used by the requirement that is being crosscut, or it is used because of an entity used by the requirement (i.e., Direct-Use is “Yes” or “Entity”). This prevents redundancies because of actions used by that action.
4. The Action Modifier for the crosscutting requirement is not null (i.e., it is “Restrict” or “Unconditional”). Only such requirements can force some functionality on other requirements (i.e., crosscut them).
5. The Action Modifier for the requirement that is being crosscut is different from the Action Modifier for the crosscutting requirement. If both requirements have the same Action Modifier, then they specify related functionality that does not imply changes to one of them.

The match-points list includes the mode or state attributes for the crosscutting requirement. The attributes, if needed, are available from the requirements attributes table (Table 19).

Pseudo code for the creation of the “Candidate Match-Points” table:

```
For each pair of records from “Actions Used by Requirements”
table, with the same Action but that are the result of different
requirements:
       where the priority of the first requirement is lower
                 than the priority of the second record
       and where the action of the first record is directly used
                 by a requirement or is a result of using Entity
       and Action Modifier of both records is not the same

    Add record to “Match by Action” table with
        Action, all attributes of action
            from “Requirements Attributes” table record
```

```
          with same Action
          and with Requirement Number of the second record
Requirement_Number, Priority, Action_Modifier
    from first requirement
Crosscut_Number, Crosscut_Priority,
                 Crosscut_Action_Modifier2
    from second requirement
```

The candidate match-points that have been identified in this step are listed in Table 20.

Table 20 Candidate Match-Points

Req Num	Priority	Action Modifier	Crosscut Num	Crosscut_ Priority	Crosscut Action Modifier	Action	Coverage	Reg. Mode	Power State	MS Prio.	Call Prio.	Tx Mode
Req-250	5		**Req-610**	5	Ignore	Power On	NA	NA	NA	NA	NA	TXI
Req-250	5		**Req-610**	5	Ignore	Register	NA	NA	NA	NA	NA	TXI
Req-250	5		**Req-710**	5	Ignore	Power On	NA	Unreg.	NA	NA	NA	NA
Req-250	5		**Req-710**	5	Ignore	Register	NA	Unreg.	NA	NA	NA	NA
Req-250	5		**Req-730**	5	Ignore	Power On	Out	Unreg.	Power On	NA	NA	NA
Req-250	5		**Req-730**	5	Ignore	Register	Out	Unreg.	Power On	NA	NA	NA
Req-250	5		**Req-740**	5	Ignore	Power On	Out	NA	NA	NA	NA	NA
Req-250	5		**Req-740**	5	Ignore	Register	Out	NA	NA	NA	NA	NA
Req-310	4		**Req-320**	4	Ignore	Press PTT	NA	NA	NA	NA	NA	NA
Req-310	4		**Req-320**	4	Ignore	Press PTT	NA	NA	NA	NA	NA	NA
Req-310	4		**Req-520**	6	Unconditionally	Call Ack	NA	NA	NA	Emergency	NA	NA
Req-310	4		**Req-520**	6	Unconditionally	Call Mode to Active	NA	NA	NA	Emergency	NA	NA
Req-310	4		**Req-520**	6	Unconditionally	Initiate Call	NA	NA	NA	Emergency	NA	NA
Req-310	4		**Req-520**	6	Unconditionally	Press PTT	NA	NA	NA	Emergency	NA	NA
Req-310	4		**Req-520**	6	Unconditionally	Tx Voice	NA	NA	NA	Emergency	NA	NA
Req-330	4		**Req-340**	4	Ignore	Join Incoming Call	NA	NA	NA	NA	NA	NA
Req-330	4		**Req-340**	4	Ignore	Receive Incoming Call	NA	NA	NA	NA	NA	NA
Req-330	4		**Req-340**	4	Ignore	Receive Incoming Call	NA	NA	NA	NA	NA	NA
Req-340	4	Ignore	**Req-540**	6	Unconditionally	Join Incoming Call	NA	NA	NA	NA	Emergency	NA
Req-340	4		**Req-710**	5	Ignore	Receive Incoming Call	NA	Unreg.	NA	NA	NA	NA
Req-610	5	Ignore	**Req-520**	6	Unconditionally	Tx	NA	NA	NA	Emergency	NA	NA

7.7.2 Remove Redundant Entries

In this step, redundant rows generated in the first step are deleted. Redundant rows are created in the first step because two requirements may be matched to several actions that are implied by one action (according to Table 12). For example, requirements that match because of Initiate Call may also be matched because of Tx-Voice (which is implied by call initiation). Redundancy may also occur because a requirement uses different Action Modifiers in different requirement parts (see Table 13).

Pseudo code for removing the redundant records:

```
Delete each record from “Match By Action”
    where a second record exist
        that results from the same Requirement
        and with the same Crosscut Requirement
        and with Action that uses the Action of the first record
End Delete
```

For example, from the following match-points between Req-310 and Req-520, only the “Press PTT” action is kept; all other actions are recursively implied by the Press PTT action, and therefore are removed (the strikethrough actions).

Table 21 Removed Redundant Candidate Match-Points

Req Num	Priority	Action Modifier	Crosscut Num	Crosscut_Priority	Crosscut Action Modifier	Action
~~**Req 310**~~	~~**4**~~		~~**Req 520**~~	~~6~~	~~Uncond.~~	~~Call Ack~~
~~**Req 310**~~	~~**4**~~		~~**Req 520**~~	~~6~~	~~Uncond.~~	~~Call Mode to Active~~
~~**Req 310**~~	~~**4**~~		~~**Req 520**~~	~~6~~	~~Uncond.~~	~~Initiate Call~~
~~**Req 310**~~	~~**4**~~		~~**Req 520**~~	~~6~~	~~Uncond.~~	~~Press PTT~~
~~**Req 310**~~	~~**4**~~		~~**Req 520**~~	~~6~~	~~Uncond.~~	~~Tx Voice~~

7.7.3 Remove Impossible or same Mode/State Match-Points

In this step, match-points that cannot happen in reality (based on Table 9) are removed. For example, the following match-points are removed because the MS cannot be in a Call

(which is part of Req-340 conditions) while unregistered, and because MS cannot be in Idle and Call modes at the same time (Req-310 and Req-320).

Table 22 Removed Impossible Match-Point

Req Num	Priority	Action Modif.	Crosscut Num	Crosscut_Priority	Crosscut Action Modifier	Action	Coverage	Reg. Mode	Power State	MS Priority	Call Prio.	Tx Mode
~~**Req-340**~~	~~4~~		~~**Req-710**~~	~~5~~	~~Restrict~~	~~Receive Incoming Call~~	~~NA~~	~~Unreg.~~	~~NA~~	~~NA~~	~~NA~~	~~NA~~
~~**Req-310**~~	~~4~~		~~**Req-320**~~	~~4~~	~~Restrict~~	~~Press PTT~~	~~NA~~	~~NA~~	~~NA~~	~~NA~~	~~NA~~	~~NA~~

The match-point candidates remaining after this step are regarded as the final list of match-point candidates.

Pseudo code for removing the redundant records according to the above two criteria:

```
Delete each record from “Match By Action”
    where two of its attributes contradict
          according to list of contradicting Mode/State values
                    and the “Requirements Attributes” table
    or where the two crosscutting requirements
              does not have at least one different attribute
              according to the “Requirements Attributes” table
End Delete
```

7.7.4 The Final Match-Point Candidates

Table 23 below is the result of the process for using the input requirements defined in this work. Note that the MS *Call* mode and state attributes are not included; they are not regarded as crosscutting and therefore do not add value to the analysis.

Table 23 Requirements Match-Points

Req Num	Crosscut Num	Action	Priority	Action Modifier	Crosscut Prio.	Crosscut Action Modifier	Coverage	Reg. Mode	Power State	MS Prio.	Call Prior.	Tx Mode
Req-250	**Req-610**	Power On	5		5	Ignore	NA	NA	NA	NA	NA	TXI
Req-250	**Req-730**	Power On	5		5	Ignore	Out	Unreg.	Power On	NA	NA	NA
Req-250	**Req-740**	Power On	5		5	Ignore	Out	NA	NA	NA	NA	NA
Req-260	**Req-610**	Power Off	5		5	Ignore	NA	NA	NA	NA	NA	TXI
Req-260	**Req-710**	Power Off	5		5	Ignore	NA	Unreg.	NA	NA	NA	NA
Req-260	**Req-740**	Power Off	5		5	Ignore	Out	NA	NA	NA	NA	NA
Req-270	**Req-610**	Power Off	5		5	Ignore	NA	NA	NA	NA	NA	TXI
Req-270	**Req-710**	Power Off	5		5	Ignore	NA	Unreg.	NA	NA	NA	NA
Req-270	**Req-740**	Power Off	5		5	Ignore	Out	NA	NA	NA	NA	NA
Req-310	**Req-520**	Press PTT	4		6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-310	**Req-540**	Call Mode to Active	4		6	Uncond.	NA	NA	NA	NA	Emergency	NA
Req-310	**Req-610**	Press PTT	4		5	Ignore	NA	NA	NA	NA	NA	TXI
Req-310	**Req-710**	Press PTT	4		5	Ignore	NA	Unreg.	NA	NA	NA	NA
Req-310	**Req-720**	Press PTT	4		5	Ignore	Out	NA	NA	NA	NA	NA
Req-310	**Req-740**	Press PTT	4		5	Ignore	Out	NA	NA	NA	NA	NA
Req-320	**Req-520**	Press PTT	4		6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-320	**Req-520**	Press PTT	4	Ignore	6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-320	**Req-610**	Press PTT	4		5	Ignore	NA	NA	NA	NA	NA	TXI
Req-330	**Req-520**	Call Mode to Active	4		6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-330	**Req-540**	Join Incoming Call	4		6	Uncond.	NA	NA	NA	NA	Emergency	NA
Req-330	**Req-710**	Receive Incomin	4		5	Ignore	NA	Unreg.	NA	NA	NA	NA

Req Num	Crosscut Num	Action	Priority	Action Modifier	Crosscut Prio.	Crosscut Action Modifier	Coverage	Reg. Mode	Power State	MS Prio.	Call Prior.	Tx Mode
		g Call										
Req-340	**Req-540**	Join Incoming Call	4	Ignore	6	Uncond.	NA	NA	NA	NA	Emergency	NA
Req-610	**Req-520**	Tx	5	Ignore	6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-710	**Req-520**	Press PTT	5	Ignore	6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-710	**Req-540**	Join Incoming Call	5	Ignore	6	Uncond.	NA	NA	NA	NA	Emergency	NA
Req-720	**Req-520**	Press PTT	5	Ignore	6	Uncond.	NA	NA	NA	Emergency	NA	NA
Req-740	**Req-520**	Tx	5	Ignore	6	Uncond.	NA	NA	NA	Emergency	NA	NA

7.8 Evaluating Match-Points

The final evaluation step, before specifying DRs, is to identify which of them should result in a derived requirement. For that, the following attributes are added to each match-point:

- **Crosscutting Attribute**: To identify which of the match-point attributes cause crosscutting between requirements. Usually, this is a mode or state.
- **Contribution of a Crosscutting Requirement to another requirement's functionality**: This indicates whether the function defined by the crosscutting requirement does one of the following: a) conflicts with the function for the requirement it crosscuts ("-"), b) adds to that functionality ("+"), or c) does not affect it ("**None**").
- **Composition Rules** (for the crosscutting requirements to the requirement it cuts): Possible values are: a) **Overlap Before/After**, b) **Override**, or c) **Wrap** (see section 3.3.6).

Table 24 summarizes this evaluation for the identified match-points. For reference, the table also shows the numbers of the resulting DRs, which are identified in a later phase.

Note that the Contribution for the Crosscutting Attribute is: "-" when the Crosscutting Action Modifier is "Restrict", and "+" when it is "Unconditional". Therefore, the contribution attribute is practically redundant.

Table 24 Match-Points Evaluation

Req Num	Action Modif.	Crosscut Req Num	Crosscut Action Modifier	Action	Crosscutting Attributes	Contribution of Crosscut Req (None / + / -)	Composition Rules [Overlap Before / After | Override | Wrap]	Derived Req.
Req-250		**Req-610**	Ignore	Power On	TXI	-	Override (Register)	**Req-1340**
Req-250		**Req-730**	Ignore	Power On	Unregistered, Out, Powering-On	-	Override (Register)	**Req-1120**
Req-250		**Req-740**	Ignore	Power On	Out	-	Override (Register)	**(Join to Req-1120)**
Req-260		**Req-610**	Ignore	Power Off	TXI	-	Override (De-register)	**Req-1330**
Req-260		**Req-710**	Ignore	Power Off	Unregistered	-	NONE (260 already de-register only if registered)	**None**
Req-260		**Req-740**	Ignore	Power Off	Out	-	Override (De-register)	**Req-1130**
Req-270		**Req-610**	Ignore	Power Off	TXI	-	Override (De-register)	**(Join to Req-1130)**
Req-270		**Req-710**	Ignore	Power Off	Unregistered	-	Override (De-register)	**Req-1110**
Req-270		**Req-740**	Ignore	Power Off	Out	-	Override (De-register)	**(New? Join to Req-1130?)**
Req-310		**Req-520**	Unconditionally	Press PTT	Emergency-Mode	-	Override (Emergency Call Priority)	**Req-1230**
Req-310		**Req-540**	Unconditionally	Call Mode to Active	Emergency-Incoming-call	None	NONE (No conflict)	**None**
Req-310		**Req-610**	Ignore	Press PTT	TXI	-	Override (Initiate Call = Ignore PTT)	**Req-1320**
Req-310		**Req-710**	Ignore	Press PTT	Unregistered	-	Override (Initiate Call = Ignore PTT)	**Req-1410**
Req-310		**Req-720**	Ignore	Press PTT	Out	-	NONE (720 already say to ignore PTT and is	**None**

Req Num	Action Modif.	Crosscut Req Num	Crosscut Action Modifier	Action	Crosscutting Attributes	Contribution of Crosscut Req (None / + / -)	Composition Rules [Overlap Before / After | Override | Wrap]	Derived Req.
							the self-derived)	
Req-310		**Req-740**	Ignore	Press PTT	Out	-	Override (Initiate Call = Ignore PTT)	**New**
Req-320		**Req-520**	Unconditionally	Press PTT	Emergency-Mode	+	Override (Initiate Call instead of Tx in call)	**Req-1240**
Req-320	Ignore	**Req-520**	Unconditionally	Press PTT	Emergency-Mode	+	Override (Initiate Call instead of Tx in call)	**Req-1240**
Req-320		**Req-610**	Ignore	Press PTT	TXI	-	Override (Ask Tx=Ignore PTT)	**Req-1320**
Req-330		**Req-520**	Unconditionally	Call Mode to Active	Emergency-Mode	None	NONE (No conflict)	**None**
Req-330		**Req-540**	Unconditionally	Join Incoming Call	Emergency-Incoming-call	None	NONE (in Idle Mode, MS joins the call anyway)	**None**
Req-330		**Req-710**	Ignore	Receive Incoming Call	Unregister	-	Override (Join Call)	**New**
Req-340	Ignore	**Req-540**	Unconditionally	Join Incoming Call	Emergency-Incoming-call	-	Override (Ignore Incoming call - instead, join the new call)	**Req-1250**
Req-610	Ignore	**Req-520**	Unconditionally	Tx	Emergency-Mode	+	<Need to DECIDE whether to allow Emergency Tx in TXI>	**Req-1310**
Req-710	Ignore	**Req-520**	Unconditionally	Press PTT	Emergency-Mode	+	<Need to DECIDE whether to allow Emergency Tx when Unregistered>	**Req-1210**
Req-710	Ignore	**Req-540**	Unconditionally	Join Incoming Call	Emergency-Incoming-call	+	<Need to DECIDE whether to allow to Join Emergency Call when Unregistered>	**New**
Req-720	Ignore	**Req-520**	Unconditionally	Press PTT	Emergency-Mode	+	<Need to decide what to do with Emergency-PTT when Out of Coverage>	**Req-1220**

Req Num	Action Modif.	Crosscut Req Num	Crosscut Action Modifier	Action	Crosscutting Attributes	Contribution of Crosscut Req (None / + / -)	Composition Rules [Overlap Before / After | Override | Wrap]	Derived Req.
Req-740	Ignore	**Req-520**	Unconditionally	Tx	Emergency-Mode	+	<Need to decide what to do with Emergency-PTT when Out of Coverage>	**(Join to Req-1220)**

Here are some notes regarding some of the decisions made regarding the match-points:

- The **Aspectual Requirements,** i.e. the requirements that crosscut other requirements are: Req-520, Req-540, Req-610, Req-710, and Req-720.
- Only **Override Composition-Rule** is practically used, which means that the Composition-Rule attribute may not be useful in context with this methodology. The Modes and States are the main reason for the crosscutting functionality conditions. This is because the requirements defined here are event based, and the Modes and States define the main conditions for the MS functionality in the events. This is probably why the composition rules *wrap* and *overlap* are not used. Yet, using the Composition Rule for this methodology still needs further evaluation.
- In several cases, the **decision regarding the resolution** for the conflict between requirements is **not clear**. This usually happens when an aspectual requirement crosscuts other aspectual requirements.
  For example, should the match-point between aspectual requirements Req-520 and Req-710 allow MS in an emergency to initiate calls, even if it is not registered in the cellular system? Note that for most cellular systems, the resolution for a similar conflict would be that any user can call emergency services.
- The **identified (false) match-points.** Req-310/Req-540 and Req-330/Req-520 could have been removed while preparing the match-point candidates list, by separating the *Call* mode to Active for Incoming [Rx] calls and Active for Outgoing [Tx] calls. In general, during the evaluation process, additional or more accurate definitions of actions and attributes to be defined can be expected. The value of adding these more detailed definitions depends on how many false match-points are saved.

7.9 Generating the Derived Requirements

Derived requirements are generated according to the attributes defined for each match-point. The requirement numbers for each of the DRs are in Table 24. Note that several match-points may result in one derived requirement. For example, Req-1320 is the result of the match-point between Req-310 and Req-610, and also the result of the match-point between Req-320 and Req-610.

The following DRs are the result of the identified match-points (Table 24). In certain cases, a derived requirement is an enhancement to the original requirement and replaces it. The underlined parts are the parts with the added text to the original requirements.

Req-610 and Req-520

Req-1310: **When in *TXI* mode**, MS shall ignore any request to transmit; except when MS is in *Emergency* mode.

[The decision here is to allow transition during Tx-Inhibit (TXI) mode, while in Emergency mode, assuming that the danger of not being able to communicate during emergency is more severe than the danger of transmitting (TXI is usually used in cases where transmitting is problematic; e.g., in hospitals where it may interrupt medical equipment or in oil fields, where it may cause a fire).]

Req-710 and Req-520

Req-1210: **While MS is *unregistered***, no system related operation should be allowed by the MS, except when MS is in *Emergency* mode, where *initiate call* and asking for transmission permission should be allowed.

Req-710 & Req-540

Req-2260: **While MS is *unregistered***, no system related operation should be allowed by the MS, including not joining an incoming emergency call.

[It is assumed that if an MS cannot register to a system, it is not part of the system's active subscribers and therefore will not participate in emergency calls. Note that this requirement can be combined with Req-1210 above. (This requirement was not identified as part of the initial writing of the derived requirements, before performing the process described here.)]

Req-720 & Req-520 and Req-740 & Req-520

Req-1220: **When MS is *out of system coverage***, MS shall not try to transmit. Pressing PTT shall be ignored, unless the MS is in *Emergency* mode, where the MS should initiate *emergency* call as soon as it is in coverage (unless *Emergency* mode is over by that time).

[Req-720 and Req-740 seem to be redundant, "ignoring PTT" as defined by Req-720 is actually implied here by "not try to Tx". This is probably a common case and therefore should not cause any issue during the analysis.]

Req-250 & Req-610

Req-1340: **On power-on**, MS shall register to the system, unless it is in *TX Inhibit* mode (set before the previous power-off).

[The decision here is that the MS should remember its TXI mode during power off/on cycle, and therefore cannot register when powered on.]

Req--250 & Req-730 and Req-250 & Req-740

Req-1120: **On power-on**, when MS is *outside of system coverage*, MS shall register to the system once it is *in coverage*.

Req-260 & Req-610 with Req-270 & Req-610

Req-1330: **On power-off**, the MS shall de-register first from the system, if it is successfully registered and if it is not in *Tx Inhibit* mode.

[Req-270 does not effect the requirement text, but it effects the decision to allow power off without de-registration first.]

Req-260 & Req-260 & Req-740 with Req-270 & Req-740

Req-1130: **On power-off**, the MS shall de-register first from the system, if it is successfully registered and if it is *in coverage*.

[Req-270 does not effect the requirement text, but it effects the decision to allow power off without de-registration first.]

Req-2-270 & Req-710

Req-1110: MS shall be able to power-off in any state even when unregistered.

[Taking into account Req-1130 and Req-1330 above, this requirement becomes redundant. It is given here only for reference to the original list of derived requirements.]

Req-310 and Req-520

Req-1230: **Pressing PTT in *Idle Call* mode and *Emergency* mode** shall *initiate-call* for an outgoing group call, with *emergency* priority to the *emergency* predefined group. If the call-initiation is acknowledged by the system, MS shall toggle to *Call* mode and may start voice transmission.

[This requirement is the equivalent to Req-1230 in the expected Derived Requirements, although Req-1230 is broader, as it includes both Req-320 and Req-310&520 in one requirement. That is, Req-1230 is written in such a way that it can replace Req-310.]

Req-310 & Req-610

Req-1320: **Pressing PTT in *Idle* mode** shall be ignored if MS is in *Tx Inhibit* mode.

Req-310 and Req-710

Req-1410: **Pressing PTT in *Idle Call* mode and *Normal* mode** while MS is *Unregistered* shall be ignored.

[Note that this requirement may be combined with Req-1230 (that is, the result of Req-520 crosscutting Req-310) to one requirement, by adding "even if MS is Unregistered".]

Req-310 and Req-740

Req-2160: **Pressing PTT in *Call* mode while MS is *Out of Coverage*** shall be ignored.

Req-320 and Req-520

Req-1240: **Pressing PTT in *Call* mode in *Emergency* mode** shall cause the MS to ask for voice Tx permission, when no one else is transmitting in the call, with Emergency Call priority, regardless if someone else is transmitting in the call. The PTT shall be ignored when someone else is already transmitting in the call. The MS may start to Tx voice only if the Tx Request was Acknowledged.

Req-320 & Req-610

Req-1320: **Pressing PTT in *Call* mode while MS is in *Tx Inhibit* mode** shall be ignored.

Req-330 & Req-710

Req-2460: **When receiving an incoming group call in *Idle* mode**, MS shall toggle to *Call* mode and join the call, unless the MS is *unregistered*, in which case the call should be ignored.

Req-340 & Req-540

Req-1250: **When receiving an incoming group call with *Emergency* priority while in *Normal* Call mode**, the MS shall internally reject the call without notifying the system (leave the current call and join the Emergency call).

8 Summary and Conclusions

This work presented DRAS, a methodology to help identify and handle crosscutting requirements in the requirements of a system. In many cases, several major problems occur in products because match-points between requirements were not identified. A major goal of AORE methods is to help resolve this issue. Most existing AORE methods concentrate on: handling interactions between requirements during the requirements analysis phase, or during system architecture, software architecture and design phases. However, usually only engineers are familiar with the tools and methods used in these phases. Therefore, in many cases the analysis of crosscutting requirements output is limited only to engineers. For other stakeholders, such as customers and marketing people, it is desirable to state the requirements to the extent possible, in textual format. Another limitation of several existing methods is that they mainly handle NFRs. They either do not handle FRs at all, or do not handle them well. The DRAS methodology was designed to identify and handle crosscutting functional requirements, and to generate textual DRs (which are the result of analyzing crosscutting requirements).

DRAS identifies crosscutting requirements based on the actions they use. It starts with identifying the lists of actions and entities used by the input requirements. The relative priority of each requirement is also identified. Then the list of actions (implied by each action) is defined. This list is later used to identify all of the actions a requirement refers to, directly or indirectly. Generating the list depends on whether the requirement restricts the use of an action, or eases a restriction for its use. If the use of an action *Act* is restricted, the use of all actions that use *Act* (i.e., the implied-actions) is restricted too. If a requirement eases the restrictions for using an action *Act*, the actions list will include actions that are the result of using *Act* (i.e., the implying actions).

For each requirement, the modes and states of the different entities it refers to are also identified. This information is later used to help decide whether functional requirements

crosscut each other, because this usually depends on the modes and states referred to by the requirements.

The actions (and their modifiers), the modes and the states per each requirement are identified. Based on this information, match-points between the requirements are identified. To get to the final list of match-points, the list is further refined to remove redundancies and conflicts that cannot occur in real-life.

The final step of DRAS is to generate DRs, according to the list of match-points between requirements. This process usually requires consulting the stakeholders; because in several cases resolving conflicts are not straight forward, and the stakeholders should decide what direction to take. The requirements, both original and derived, can be reviewed by all stakeholders, making sure that resolutions to conflicts are performed properly.

Using DRAS provides a more reliable method to identify crosscutting functional requirements and the requirements they crosscut. It also helps in deciding what derived requirements should be generated from the crosscutting, because it identifies the match-points between the requirements. Therefore, using DRAS helps complete the requirements definition phase with a better set of requirements.

9 Future Work

Several enhancements are considered for the DRAS methodology, mainly automating the process (making it more robust and easier to use), so that the (tentative) DRs can be generated automatically. That requires the ability to parse and analyze the text and the ability to set the relative priorities between requirements a match-point refers. Note that text analysis should allow identifying actions, even when they are written in different forms. For example, “call initiation” may be written in the requirement “initiate a call”, “start call”, etc.

The DRAS methodology, or part of it, may be integrated with existing requirements management tools (such as DOORS or RequisitePro). This will enhance their functionality and enable an easier definition of requirements (derived from conflicts between other requirements).
Another possible enhancement to such tools is the definition of attributes per requirement, as used in this work (see Table 19 and Table 14). Per requirement, these attributes include the **Actions** used with their **Action Modifiers** and the **Mode/State Attributes**. With proper textual analysis, the requirements management tool may be able to generate these attributes automatically. Using these attributes, the tool can suggest to the user possible crosscutting between the requirements, by implementing similar algorithms to the ones defined for DRAS.

To achieve the above-mentioned enhancements, the requirements management tool should enable the user to define the following required input lists:

- **Action modifiers** (probably a predefined fixed list)
- **Composition rules** (probably a predefined fixed list)
- **Relative priorities** (probably a predefined fixed list)
- **Entities** list
- **Actions** list and **implied actions** list
- **Entities used by actions** list

- **Modes and states** list, their possible values and list of **contradicting mode/state values**.

An additional attribute per requirement may be the **crosscutting type**, which can be one of the following: Baseline, Crosscutting and Derived. This attribute may help users and reviewers better understand the requirements.

Another enhancement can come from handling the generated DRs. This can be supported by **derived requirements traceability** in the requirements management tool. Such traceability can be similar to the traceability used between different development phases: requirements to design, design to code, etc. However, in this case, traceability will be within the same phase. It will enable the user to verify that all identified crosscutting between requirements were handled. Traceability will require a **Derived From** attribute per requirement (listing the requirements that the requirement is derived from).
If the DRs analysis is performed automatically, then attributes identified during the analysis may be added to the DRs traceability entities. These attributes can include Actions and Modes/States that are responsible for the match-point, the involved action modifiers, and identifying non-crosscutting match-points, etc.

DRAS methodology assumes that a match-point it identifies (between functional requirements) means that tentatively one requirement crosscuts the other. That is, one of the requirements is a crosscutting requirement. This assumption was not validated; further work is required to identify whether this is true, or for what cases this is true.

Using natural language processing methods to analyze the requirements (e.g. [Pantel 07], [Lin 07], EA-Miner [Sampaio 2005]), it may be possible to **semi-automate identification of actions** (used by the requirements) and their different attributes. Writing the requirements in some formal form, such as Attempto controlled language [Hoefler 04], can assist this approach. Ideas from AbstFinder [Goldin 97] may also be used to help identify aspects in the specifications text. Mining aspects methods [Loughran 02] and tools may also be used for automatic or semi-automatic retrieval and

identification of aspects. Automatic weaving (composing) of requirements (to generate the DRs) may use methods similar to the ones used by aspect oriented programming (see [Laddad 03]). Tools such as the EasyCRC tool [Raman 07], which automates the processes of finding nouns and synonyms, can be considered for finding actions and related actions in the requirements.

Using **queries to identify crosscutting requirements** and the requirements they crosscut, as defined in the Requirements Description Language (RDL) [Chitchyan 2007], is another possible approach for enhancing DRAS. RDL identifies aspectual (crosscutting) requirements by defining constraint queries about actions and objects used by the requirements. The requirements that the aspectual requirements tentatively crosscut are identified by base queries.

The use of **XML** to internally represent requirements can also be considered. Note that XML cannot be used to represent input and output requirements, because these should be in textual format, so as to be understandable for all stakeholders. XML representation can help automate the creation of DRs. Methods will be needed to translate the textual requirements from text to XML (or other format) and to translate back the XML representation for DRs to textual format. XML is already used for aspect-oriented methods (e.g., the ARCaDe tool [Rashid 03; Katz 04]) to compose requirements, or for supporting aspects plug-ins in software design [Lopes 05]. Concepts from these and other approaches may be reused.

To allow automatic detection of relative priorities between requirements, **priorities may be added per attribute value** (e.g., Normal=1, Emergency=2). In addition to requirements priorities, **t**his can also enable having relative priorities between requirements (i.e., a partially ordered tree of requirements priorities). There will be no absolute priority per requirement, and the relative priority of each pair of requirements should be evaluated separately. In addition, default values per attribute should be defined. This will enable requirements handling, where partial attribute values are

specified (e.g., set call priority default as “Normal”).

Composition rules can be enhanced to improve the automation process. In many cases, current composition-rules values are not useful. Different values for composition rules, which are more suited for generating DRs, may be more useful. One possible approach is to define temporal rules, such as “Override Temporarily”, “Delayed After”, “On Event” (e.g., when mode changes). Enhancements using ideas from LOTOS [Bolognesi 87] and [Brito 04] should also be considered.

Temporal logic may also be used to enhance the method [Manna 92]. **Action Modifiers** identified in DRAS, “Restrict” and “Unconditionally”, seem to be **similar to Temporal Logic Path Quantifiers/Operators A/G** (all paths / always) and A/H (all paths / always in the past). It may be possible to develop a logic based on temporal logic, that will use such action-modifiers and specify (using a formula), the effect of these aspectual action-modifiers on other requirements (e.g., **Emergency -> [A(always) PTT -> Initiate Call”]**). The logic may be defined as an extension to already existing methods which support temporal logic for requirements, such as Formal Tropos [Fuxman 03] or Kaos [Bertrand 98]. Using formal languages that use temporal logic may allow the use of Model Checking methods [Manna 92] to identify crosscutting and conflicting requirements. The ideas suggested by [Katz 04] for the use of temporal logic in the PROBE framework can also be used as input for enhancements.

10 References

[Araujo 05] J. Araujo, E. Baniassad, P. Clements, A. Moreira, A. Rashid, B. Tekinerdogan. Early Aspects: The Current Landscape. Technical Note CMU/SEI-2005-TN-xxx, Technical Report Lancaster University COMP-001-2005, 2005. Available at http://trese.cs.utwente.nl/early-aspects-AOSD2005/Papers/EarlyAspects-LandscapePaper-FirstDraft-2005.pdf (last visited: July 2007).

[Bakker 05] J. Bakker, B. Tekinerdogan, M. Aksit. Characterization of Early Aspects Approaches. Early Aspects Workshop, March 15, 2005. Available at http://trese.cs.utwente.nl/early-aspects-AOSD2005/Papers/11_BakkerBedirAksit_Twente.pdf (last visited: May 2007).

[Baniassad 04a] E. Baniassad S. Clarke. Theme: An approach for aspect-oriented analysis and design. International Conference on Software Engineering, 2004. Available at http://www.cse.cuhk.edu.hk/~elisa/papers/theme.pdf (last visited: July 2007).

[Baniassad 04b] E. Baniassad E., E. Siobhan. Finding Aspects in Requirements with Theme/Doc, Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design 2004. Available at http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/Baniassad-Clarke.pdf (last visited: July 2007).

[Bar-On 07a] D. Bar-On. S. Tyszberowicz. Derived Requirements Generation. Proceedings of 2nd International Workshop on Aspects, Dependencies and Interactions, July 2007. Available at http://www.aosd-europe.net/adi07/papers/baron_adi07.pdf (last visited: September 2007).

[Bar-On 07b] D. Bar-On. S. Tyszberowicz. Derived Requirements Generation - the DRAS methodology. IEEE International Conference on Software – Science, Technology and Engineering, October 2007.

[Bergmans 01] L. Bergmans, A. Aksits. Composing Crosscutting Concerns using Composition Filters. Communications of the ACM, October 2001/Vol. 44, No. 10, pp 51-57. Available at http://doi.acm.org/10.1145/383845.383857 (last visited: May 2007).

[Bertrand 98] P. Bertrand, R. Darimont, E. Delor, P. Massonet, A. van Lamsweerde. GRAIL/KAOS: an Environment for Goal Driven Requirements Engineering. Proceedings 20th International Conference on Software Engineering, April 1998. Available at http://www.info.ucl.ac.be/Research/Publication/1998/icse984p.ps.gz (last visited: August 2007).

[Bolognesi 87] B. Bolognesi, E. Brinksma. Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems, Vol. 14, pp 25-59, 1987. Available at http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=44214 (last visited: April 2007).

[Booch 99] G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[Brito 02] I. Brito, A. Moreira, J. Araujo. A requirements model for quality attributes. 1st International Conference on Aspect-Oriented Software Development, 2002.

Available at http://trese.cs.utwente.nl/AOSD-EarlyAspectsWS/Papers/Brito.pdf (last visited: May 2007).

[Brito 03] I. Brito, A. Moreira. Towards a Composition Process for Aspect-Oriented Requirements. Workshop on Early Aspects: AORE and Architecture Design, March 17 – Boston, USA, 2003. Available at http://www.cs.bilkent.edu.tr/AOSD-EarlyAspects/Papers/BritoMoreira.pdf (last visited: July 2007).

[Brito 04] I. Brito, A. Moreira. Integrating the NFR framework in RE model. Early Aspects Workshop, Lancaster U.K., 2004. Available at http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/BritoMoreira.pdf (last visited: July 2007).

[Chitchyan 05] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia. Survey of Aspect-Oriented Analysis and Design Approaches. AOSD-Europe-ULANX-9. 18 May 2005. Available at http://www.comp.lancs.ac.uk/computing/aod/papers/d11.pdf (last visited: July 2007).

[Chitchyan 07] R. Chitchyan, A Rashid, P. Rayson, R. Waters. Semantics-based composition for aspect-oriented requirements engineering. Proceedings of the 6th International Conference on Aspect-Oriented Software Development (AOSD), pages 36-48, 2007. Available at http://portal.acm.org/ft_gateway.cfm?id=1218569&type=pdf (last visited: August 2007).

[Chung 00] Chung L., Nixon B., Yu, E. and Mylopoulos, J. Non Functional Requirements in Software Engineering. Boston: Kluwer Academic Publishers, 2000.

[Creveling 03] C.M. Creveling, J.L. Slutzky, D. Antis Jr. Design for Six Sigma in Technology and Product Development. Prentice Hall, 2003.

[Filman 05] R.E. Filman, T. Elrad, S. Clarke, M. Aksit. Aspect-Oriented Software Development, Addison-Wesley, 2005.

[Finkelstein 96] A. Finkelstein, I Sommerville. The View Point FAQ. Software Engineering Journal, Vol. 11, pp 2-4, 1996. Available at http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/viewfaq.pdf (last visited: April 2007).

[Fowler 03] M. Fowler, S. Kendall. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 2003.

[Fuxman 03] A. Fuxman, R. Kazhamiakin, M. Pistore, M. Roveri. Formal Tropos: language and semantics. Version 1.0: November 4, 2003. Available at http://dit.unitn.it/~tropos/papers_files/ftsem03.pdf (last visited: July 2008).

[Garcia-Duque 06] J. Garcia-Duque, M. Lopez-Nores, J. J. Pazos-Arias, A. Fernandez-Vilas, R. P. Diaz-Redondo, A. Gil-Solla, M. Ramos-Cabrer, Y. Blanco-Fernandez. Guidelines for the incremental identification of aspects in requirements specifications. Requirements Eng (2006), DOI 10.1007/s00766-006-0028-72006, pp 239-263, Springer-Verlag London Limited 2006. Available at http://www.springerlink.com/index/6N1R150311H22124.pdf (last visited: May 2007).

[Goldin 97] L. Goldin, D. Berry. AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation. Automated Software

Engineering, Vol. 4, pp 375-412,1997. Available at http://www.springerlink.com/index/K68M2902K83526HX.pdf (last visited: July 2007).

[Grundy 99] J. Grundy. Aspect-oriented Requirements Engineering for Component-based Software Systems, Proceedings of RE'99, 7-11 June, Limerick, Ireland, 1999. Available at http://ieeexplore.ieee.org/ (last visited: May 2007)**.**

[Hoefler 04] H. Stefan. The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0. Technical Report ifi-2004.03. Available at http://www.ifi.unizh.ch/attempto/publications/papers/hoefler2004theSyntax.pdf (last visited: July 2007).

[Katz 04] S. Katz, A. Rashid. From aspectual requirements to proof obligations for aspect-oriented systems. In: Proceedings of the 12th IEEE international conference on requirements engineering, Kyoto, pp 48–57, 2004. Available at http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335663 (last visited: July 2007).

[Kovitz 99] B. L. Kovitz. Practical Software Requirements. Manning Publications Co., 1999.

[Laddad 03] R. Laddad. AspectJ in Action Practical Aspect-Oriented Programming. Manning Publications Co., 2003.

[Lieberherr 96] K.J. Lieberherr. Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns. PWS Publishing Company, Boston, 1996. Available at http://www.ccs.neu.edu/research/demeter/book/book-download.html (last visited: May 2007).

[Lin 07] D. Lin, Demos. Available at http://www.cs.ualberta.ca/~lindek/demos.htm (last visited: July 2007).

[Loughran 02] N. Loughran, A. Rashid. Mining Aspects. In Proceedings of Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, April 2002. Available at http://www.comp.lancs.ac.uk/computing/aose/papers/PersAsp_SPL_EarlyAspects2002.pdf (last visited: July 2007).

[Lopes 05] C. V. Lopes, T. C. Ngo. The Aspect Oriented Markup Language and its Support of Aspect Plug-ins. ISR Technical Report # UCI-ISR-04-8, Institute for Software Research, ICS2 210, University of California, October 2004. Available at http://www.isr.uci.edu/tech_reports/abstracts/UCI-ISR-04-8-abs.pdf (last visited: August 2007).

[Manna 92] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer-Verlag, 1992.

[Moreira 02] A. Moreira., J. Araujo, I. Brito. Crosscutting Quality Attributes for Requirements Engineering. 14th International Conference on Software Engineering and Knowledge Engineering (SKE 2002), pp 167-174, ACM Press, Italy, July 2002, http://portal.acm.org/ (last visited: August 2007).

[Mylopoulos 01] J. Mylopoulos at. al. Exploring Alternatives During requirements Analysis. IEEE Software, January/February 2001, pp 2-6. Available at http://ieeexplore.ieee.org/ (last visited: August 2007).

[Nuseibeh 04] B. Nuseibeh. Crosscutting Requirements, AOSD conference 2004 keynote presentation, Available at http://aosd.net/2004/archive/AOSD-FromPromiseToReality.ppt (last visited: August 2007).

[Ossher 00] H. Ossher, P. Tarr. Multi-dimensional separation of concerns and the Hyperspace approach. In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development, Kluwer, 2000. Available at http://www.research.ibm.com/hyperspace/Papers/sac2000.pdf (last visited: May 2007).

[Palmer 02] S.R. Palmer, J.M. Felsing. A Practical Guide to Feature-Driven Development. Prentice Hall, 2002.

[Pantel 07] Z. Pantel, Demos. Available at http://www.isi.edu/~pantel/Content/demos.htm (last visited: July 2007).

[Pang 04] J. Pang, L. Blair. Refining Feature Driven Development – a Methodology for Early Aspects. In Early Aspects Workshop in conjunction with the 3rd International conference on Aspect-Oriented Software Development, 2004. Available at http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/Pang-Blair.pdf (last visited: May 2007).

[Pazos-Arias 01] J.J. Pazos-Arias, J. Garcia-Duque. SCTL-MUS: a formal methodology for software development of distributed systems; a case study. Formal Aspects of Computing, Volume 13, pp 50–91, ISSN 0934-5043, 2001. Available at http://www.springerlink.com/index/X2PUA1WURK6GDTLR.pdf (last visited: May 2007).

[Raman 07] A. Raman, S. Tyszberowicz. The EasyCRC Tool, 2nd international conference on Software Engineering Advances (ICSEA), France, 2007. Available at http://ieeexplore.ieee.org/iel5/4299876/4299877/04299933.pdf?tp=&arnumber=4299933&isnumber=4299877 (last visited: July 2008).

[Rashid 02] A. Rashid, A. Moreira, J. Araujo. Early Aspects: a Model for Aspect-Oriented Engineering. IEEE Joint Conference on Requirements Engineering, pp 199-202, Essen, Germany, September 2002. Available at http://www.comp.lancs.ac.uk/computing/aose/papers/AORE_RE2002.pdf(last visited: May 2007).

[Rashid 03] A. Rashid, Moreira, J. Araujo. Modularization and Composition of Aspectual Requirements, 2nd international conference on AOSD, ACM, pp 11-20, 2003. Available at http://www.comp.lancs.ac.uk/computing/aose/papers/AORE-AOSD2003.pdf (last visited: July 2007).

[Rosenhainer 04] L. Rosenhainer. Identifying Crosscutting Concerns in Requirements Specifications. In Proceedings of OOPSLA Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, pp 49-58, October 2004.

Available at http://trese.cs.utwente.nl/workshops/oopsla-early-aspects-2004/Papers/Rosenhainer.pdf (last visited: May 2007).

[Sampaio 05] A. Sampaio, N. Loughran, A. Rashid, P. Rayson. Mining Aspects in Requirements. Workshop on Early Aspects (held with AOSD), 2005 http://www.comp.lancs.ac.uk/computing/aose/papers/Mining_EA2005.pdf (last visited: July 2007).

[Silva 02] A. Silva. Requirements, Domain Specifications: Viewpoint-based Approach to Requirements Engineering. ICSE'02, pp 94-104, 2002. Available at http://portal.acm.org/citation.cfm?doid=581339.581354 (last visited: August 2007).

[Skotiniotis 04] T. Skotiniotis, D. H. Lorenz. From contracts to aspects and back. Technical Report NU-CCIS-04-05, pp 196-197, College of Computer and Information Science, Northeastern University, Boston, MA 02115, Mar. 2004. Available at http://portal.acm.org/citation.cfm?id=1028747 (last visited: May 2007).

[Sousa 03a] G. Sousa, I. Silva, J. Castro. Adapting the NFR Framework to Aspect-Oriented Requirements Engineering. XVII Brazilian Symposium on Software Engineering, Manaus, Brazil, October, pp 177-192, 2003. Available at http://www.cin.ufpe.br/~ler/publicacoes/pub_2003/SBES03_AdaptingTheNFRFrameworkToAspect.pdf (last visited: August 2007).

[Sousa 03b] G. Sousa, J. Castro. Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle, WER, pp 223-239, 2003. Available at http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER03/georgia_souza.pdf (last visited: August 2007).

[Sousa 04] G. Sousa, S. Soares, P. Castro, B Castro. Separation of Crosscutting Concerns from Requirements to Design: Adapting as Use Case Driven Approach. Early Aspects, 2004. Available at http://www.di.ufpe.br/~scbs/artigos/sousa_soares_borba_castro_earlyAspects2004.pdf (last visited: May 2007).

[Tarr 02] Tarr P., H. Ossher. Hyper/J™: Multi-Dimensional Separation of Concerns for Java™, Proceedings of the 24th International Conference on Software Engineering, pp 24-35, 2002. Available at http://portal.acm.org/citation.cfm?id=581447 (last visited: May 2007).

[TETRA] ETSI EN 300 392-2. Terrestrial Trunked Radio (TETRA); Voice plus Data (V+D); Part 2: Air Interface (AI). Available at http://www.etsi.org (last visited: August 2007).

[Yu 04] Y. Yu, J. C. S. d. P. Leite, J. Mylopoulos. From Goals to Aspects: Discovering Aspects from Requirements Goal Models, presented at International Conference on Requirements Engineering, Kyoto, Japan, pp 38-47, 2004. Available at http://doi.ieeecomputersociety.org/10.1109/ICRE.2004.1335662 (last visited: August 2007).

תקציר

בשלביים המוקדמים של פיתוח מערכת, חלק מדרישות המוצר עשויות להשפיע זו על זו. המשמעות היא שממפרט דרישה אחת יכולים לנבוע שינויים ושיפורים בדרישה אחרת. חלק מהדרישות עלולות אפילו להיות סותרות אחת לשנייה. השפעות וסתירות אלו צריכים להילקח בחשבון ולהיפתר כמה שיותר מוקדם בתהליך הפיתוח, כדי להימנע מתוספת מחיר ועיכובים שהם לרוב התוצאה של זיהוי השפעות כאלו בשלבים מתקדמים של הפיתוח. היכולת לזיהוי השפעות וקונפליקטים בין דרישות מערכת היא לכן חשובה ביותר. זיהוי ברור שלהן בשלב הגדרת הדרישות וניתוחן מאפשר הגדרה של דרישות-נגזרות (derived-requirements) חדשות, או שינוי דרישות קיימות. דרישות אלו מגדירות את המשתמע משילוב המפרטים הדרישות שמשפיעות אחת על השנייה, ופותרות את הקונפליקטים ביניהן.

סוג חשוב של דרישות שמשפיעות על המפרט של דרישות אחרות הן הדרישות הפונקציונאליות החוצות (Crosscutting Functional Requirements). דרישות אלו משנות, או אפילן מחליפות, את המפרט של הדרישות אותן הן חוצות (crosscut), לרוב במצבים מסוימים של פעולת המערכת.

מתודולוגית DRAS (Derived Requirements generation by Actions and States) שפותחה בעבודה זו עוזרת גם בזיהוי דרישות פונקציונאליות שחוצות דרישות אחרות וגם ביצירת הדרישות הנגזרות או בשנויים לדרישות הקיימות. לצורך זיהוי הדרישות החוצות, המתודולוגיה משווה ומתאימה בין פעולות שבשימוש הדרישות ובין מצבי המערכת שאליהן מתייחסות הדרישות. פעולה הינה פונקציונאליות אשר הדרישה הפונקציונאלית מגדירה. שימוש באותה פעולה ע"י שתי דרישות מצביע על אפשרות שאחת מהן חוצה את השנייה. DRAS מטפלת גם בפעולות שבהן יש שימוש כתוצאה מהפעלת הפעולות שאליהן מתייחסות הדרישות באופן ישיר. כדי לקבוע מהן הפעולות האחרות שבהן צריך לטפל כתוצאה מפעולה ספציפית שאליה מתייחסת דרישה מסוימת באופן ישיר, המתודולוגיה בודקת האם הדרישה מגדירה מגבלות לשימוש בפעולה או מבטלת מגבלות לגבי השימוש בפעולה. ע"י כך נקבע האם יילקחו בחשבון פעולות שמשתמשות הפעולה זו או פעולות שפעולה זו משתמשת בהן.

הו הקלט והן הפלט של DRAS הם דרישות טקסטואליות וכל התהליך של המתודולוגיה מתבצע בשלב הגדרת וניתוח הדרישות. דבר זה מאפשר לכל האנשים הנוגעים בדבר להשתתף בתהליך, ולא רק לאנשים הטכניים.

תוכן עניינים

- כללי
  - תקציר
  - סקירה
- סקירת הנדסת דרישות
  - הנדסת דרישות
  - אספקטים-מוקדמים והנדסת-דרישות מונחת אספקטים
- הבעיה וסקירת הפתרון
  - דרישות-חוצות ודרישות נגזרות
  - אספקטים ו DRAS
  - הצגת הבעיה
  - כיצד DRAS פותרת את הבעיה
  - סיכום תרשים תהליך DRAS
  - מבנה העבודה
- רדיו טטרה כמקרה מבחן
- סקירת הנדסת-דרישות מונחת אספקטים (AORE)
- בחינת שיטות קיימות ל והנדסת-דרישות מונחת אספקטים
- תהליך DRAS- יצירת דרישות נגזרות ע"פ פעולות ומצבים
- דיון ומסקנות
- הצעות לכיווני מחקר נוספים
- רשימת מקורות

האוניברסיטה הפתוחה בישראל
המחלקה למתמטיקה ומדעי המחשב

DRAS

יצירת דרישות נגזרות ע״פ פעולות ומצבים

תזה הוגשה בפברואר, 2009

כחלק מהשלמת הדרישות לקראת תאר מוסמך (M.Sc) במדעי המחשב

האוניברסיטה הפתוחה בישראל

ע״י

דוד בר-און

הוכן תחת פיקוחו של ד״ר שמואל טישברוביץ