FORMALISING NON-FUNCTIONAL REQUIREMENTS EMBEDDED IN USER REQUIREMENTS NOTATION (URN) MODELS

by

CYRILLE DONGMO

submitted in accordance with the requirements

for the degree of

Doctor of Philosophy

in the subject

Computer Science

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF J.A VAN DER POLL

15 November 2016

List of acronyms

CNF-action	Complementary Non-Functional action
CSM	Core Scenario Model
CZT	Community of Z Tools
EPS	Encapsulated Postscript
FOPL	First Order Predicate Logic
GORE	Goal-Oriented Requirements Engineering
GRL	Goal-Oriented Requirements Language
MIF	Maker Interchange Format
MSC	Message Sequence Charts
NF-action	Non-Functional action
NFRs	Non-Functional Requirements
OZ	Object-Z
PRG	Programming Research Group
ProcessNFL	Process-oriented language for describing Non-Functional properties QoS Quality of Service
SDS	Software Development System
SE	Software Engineering
SIG	Softgoals Interdependency Graphs
SLR	Systematic Literature Review
SRS	Software (or System) Requirements Specification
UCM	Use Case Map
UML	Unified Modelling Language
UML-S	Unified Modelling Language for Services
URN	User Requirements Notation
XML	Extensible Mark-up Language
XSD	XML Schema Definition
ZF	Zermelo-Fraenkel

Abstract

The growing need for computer software in different sectors of activity, (health, agriculture, industries, education, aeronautic, science and telecommunication) together with the increasing reliance of the society as a whole on information technology, is placing a heavy and fast growing demand on complex and high quality software systems. In this regard, the anticipation has been on non-functional requirements (NFRs) engineering and formal methods. Despite their common objective, these techniques have in most cases evolved separately.

NFRs engineering proceeds firstly, by deriving measures to evaluate the quality of the constructed software (product-oriented approach), and secondarily by improving the engineering process (process-oriented approach). With the ability to combine the analysis of both functional and non-functional requirements, Goal-Oriented Requirements Engineering (GORE) approaches have become de facto leading requirements engineering methods. They propose through refinement/operationalisation, means to satisfy NFRs encoded in softgoals at an early phase of software development. On the other side, formal methods have kept, so far, their promise to eliminate errors in software artefacts to produce high quality software products and are therefore particularly solicited for safety and mission critical systems for which a single error may cause great loss including human life.

This thesis introduces the concept of Complementary Non-functional action (CNF-action) to extend the analysis and development of NFRs beyond the traditional goals/softgoals analysis, based on refinement/operationalisation, and to propagate the influence of NFRs to other software construction phases. Mechanisms are also developed to integrate the formal technique Z/Object-Z into the standardised User Requirements Notation (URN) to formalise GRL models describing functional and non-functional requirements, to propagate CNF-actions of the formalised NFRs to UCMs maps, to facilitate URN construction process and the quality of URN models.

Keywords: Semi-formal specification techniques, URN, GRL, UCMs, Goal model, NFR, CNF-actions, Formal Methods, Z, Object-Z, Specification Validation, Enterprise organogram, Specification animation, Z/Eves, Four-way framework.

Contents

List of Tables

List of Figures

List of publications

Published articles

1. Exploiting Enterprise Organograms to facilitate goal/requirements elicitation. First published as conference paper in [63] and later selected and republished as journal article in [66].

2. An application of a four-way framework for validating a specification: Animating an Object-Z specification using Prolog. First published as conference paper in [62] and later selected and republished as journal article in [65].

Publications vs contributions

The two published papers represent about 33% of the total number of envisaged contributions (see Section 1.6, p. 10) and about 25% of the effective contributions presented in Chapters 7 and 8. Most of the contributive ideas were already fully developed during the course of this research and are therefore, awaiting to be transformed into publishable conference papers or journal articles. Such publications are planned to follow this thesis.

Acknowledgements

In the first place, I honor and humble myself to the Heavenly Father, without Whom I might have never existed.

I would like to express my deepest recognition to the people closest to me. To my family; the greatest gift from heaven, especially to my parents and dearest wife Solange, thank you for your love, support, and understanding. Thank you to all my kids (Lemek, Loriane, Cyrus and Joshua) who have always been there to remind me that I have no chance to give up because they are watching and copying. Thank you to all my brothers and sisters.

I extend my sincere gratitude to my supervisor, Professor John Andrew Van der Poll, whose advice and guidance justify the achievements of this thesis. Without his help, it would have been hard to bring this work to its present state.

Thank you to Professor Amyot (from the School of Information Technology and Engineering at the University of Ottawa, Canada) and his team for making most of the publications on URN available on their website and for providing the jUCMNav tool for constructing and analysing UCMs and GRL models. Without these, it would have been very hard to draw all the goal models and UCMs diagrams in this thesis.

Cyrille Dongmo
November 2016

Dedication

This thesis is dedicated to the memory of my father, Albert Dongmo (1924-1998), to the memory of my grandfather, Mo'oh Diffo Takem and grand-mother Julienne Jumeta and to the memory of my very dear friend and elder sister, Rachelle Dongmo (1967 - July 2011).

Chapter 1

Introduction

1.1 Background and motivation

There has been a common acceptance of the promises of formal methods[1] to bring more precision into the conceptualisation of software systems, eliminate errors, ambiguous statements and inconsistencies at an early stage of the software development process, and hence, produce quality products [142]. However, these methods have not been widely accepted in industry. For a long period, intensive research has investigated the phenomenon and a number of reasons have been formulated to justify why formal methods have not yet been fully accepted. Amongst others, a steep learning curve, attributed to the limited mathematical skills of software engineers and practitioners has been suggested. Formal methods are said to be hard to develop and assess, and the lack of a comprehensive construction process with appropriate guidelines is cited amongst formal method issues that ought to be addressed (van Lamsweerde [179]). It is also recognised that it is hard to integrate such methods into the existing software development processes and make them operational (Abrial [4]).

Among advocates of formal methods, there are those who believe in the potential of these methods to wholly cover the software production cycle. Arguably, therefore these methods ought to cover the following software development phases:

- The Requirements analysis and specification phase, which are the essential parts of Software Requirements Engineering.

- The design phase, related to the discipline of Software Design and Architecture.

[1]The term formal methods denotes, in a broad sense, mathematical techniques for developing computer-based software and hardware systems. They provide a means to formalise system requirements in a way that mathematical proofs can be used to validate them, as well as, further refined models.

- The implementation and maintenance phases, during which the final software product is produced and rendered operational.

The requirements engineering phase is known to be a hard part, the most critical and risky stage in the process of software development (see Brooks [33], van Lamsweerde [180], Zowghi and Coulin [200]) and therefore, one of the most important research focus areas. Many of the benefits of formal methods are attributed to their ability to anticipate the detection of potential problems in requirements at an early phase of the software process. Research in formal methods, mainly focused on this phase, resulted in developing requirements notation approaches and techniques, as well as associated tools, to formally specify and validate user requirements. Subsequent development phases performed by means of refinement techniques, are intended to progressively transform the abstract formal system specification into operational software entities (see, for example Derrick and Boiten [57]).

Event-B is a typical example of a successful model-based formal approach that provides for a mechanism to formally specify, and progressively refine software system requirements (Butler [41]). A remarkable strength of this method is the availability of tools, such as the Rodin toolset (Abrial et al. [5]), to create abstract models for software system requirements, to gradually refine them into more concrete models, and to ensure, by means of automated proofs, consistency between consecutive refined models. The need for a similar toolset or integrated environment for many other formal methods is now high on the agenda. However, as observed by (Ponsard and Dieul [147]), a major weakness of these methods is the gap between textual or semi-formal requirements and formal models.

Some literature suggests the integration of formal techniques into existing software development processes (e.g., [169], pp. 219-222) and recommends the use of these methods in conjunction with existing ones [31]: "*Thou shalt not abandon thy traditional development methods*". A possible reason for such a recommendation is that stakeholders (other than the technical team) do not generally understand mathematical notations, hence are less involved in the process of formal specification. Techniques more comprehensible by customers are, therefore, needed at the initial phase of requirements engineering. The integration of formal techniques into the existing software development processes has been achieved mainly by combining semi-formal and formal software specification techniques (see, for example Cabral and Sampaio [42], Ponsard and Dieul [147], Wieringa and Dubois [191]). Two main categories of formal specification techniques are:

- Property-oriented: system properties are presented in a declarative way with Algebraic (based on equational axioms), or Axiomatic (based on first-order predicate logic) expressions to describe the data and operations of the system.

- Model-oriented: an abstract model of a system is built, where the static properties of the system are created, as states, by means of set theory and the operations on those states are constructed by means of first-order predicate logic. Two examples are: Vienna Development Method (VDM [75]) and the Z Notation [130, 170], which are traditionally among the successful techniques.

Property-oriented and model-oriented software specification techniques share in common the fact that they are all based on mathematics, use first-order predicate logic to describe the dynamic behavior of systems and therefore, relate static properties to each other. This observation indicates that research results based on one category may be generalised/extended to another.

1.2 Current trends in combining semi-formal and formal techniques

As mentioned earlier, the literature emphasises the need to integrate formal techniques into existing software development processes. Referring to formal methods, Abrial [4, page 766] believes that "*People are quite reluctant to use such methods mostly because it necessitates to modify the development process in a significant fashion. As it is well known, such development processes are hard to develop*". In [169], Sommerville suggests introducing formal methods between the specification and the validation phases of the software development process. Such integration into existing software processes therefore combines semi-formal models with formal ones. Abderrahman et al [3] recognised that the challenge is to define suitable mechanisms to translate semi-formal models into formal ones. Examples are: coupling UML and B [168] that resulted in U2B method, coupling UML and TROLL [80] that resulted in UML-TROLL, coupling UCMs and State Machines [29] that resulted in a new method called UCM-ROOM, and translating UML/Fusion into Object-Z proposed by Bittner and Kammuller [26]. Another case is the integration of UML-B and Object-Z [138], that combines one semi-formal model (UML) and two formal ones: B and Object-Z.

Figure 1.1 is proposed to illustrate a generic strategy often followed to generate a formal specification from a semi-formal model. This will be referred to as the "direct transformation method", which can be summarised as follows:

1. Concepts in a semi-formal model and the target formal notation are identified, analysed and compared.

2. Based on the above concept analysis, a transformation mechanism is derived. For

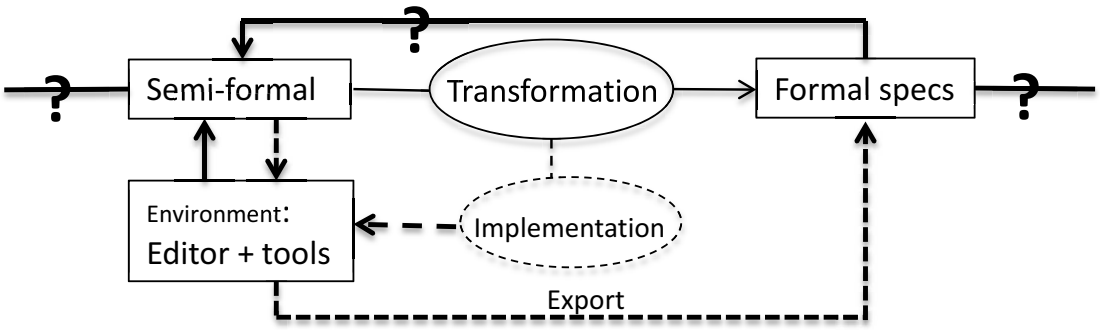

Figure 1.1: Direct transformation strategy

example in the form of guidelines, frameworks, algorithms or heuristics.

3. The transformation mechanism is implemented and (sometimes) integrated into the existing environment of the semi-formal technique. For example, a menu such as Export is added to export a semi-formal model into a formal specification document [167].

The lines or arrows in the figure with question marks, indicate those aspects of the existing transformation strategy that still need research attention. The formalisation process is generally interested in the readily constructed semi-formal model, with little concern as to the construction process of such a model. In that case, the value of formalism is, arguably, restricted, since the quality of the formal specification is relative to the input semi-formal model, and not necessarily to the initial user requirements.

Another issue that needs to be investigated is, for instance, the exploitation, during the transformation process, of the mathematical precision of formal techniques to enhance the initial semi-formal model. Performing such an improvement would render the transformation process interactive and hence, allows for the semi-formal model to act as a graphical interface for the formal techniques.

The last aspect to concentrate on concerns the design and implementation phases. In general, after the formalisation of the semi-formal model, it may be interesting to investigate what design/implementation approach is appropriate to continue with. Should one continue with the traditional design/implementation methods, or follow one of the known formal methods refinement techniques (e.g., Derrick and Boiten [57])?

1.3 Problem statement and purpose of the study

This research investigates the means to provide a comprehensive formal specification construction mechanism using semi-formal methods. For the semi-formal technique, it considers the standardised User Requirements Notation (URN) [1], which encompasses the modelling of functional, as well as, non-functional requirements. On the side of formal methods, the model-based formal notation Z/Object-Z[2] [45, 185] is considered. This work focuses on Object-Z but uses the notation Z/Object-Z very often to remind the reader about the tidy relationship between the two notations and also to justify the frequent use of Z properties when needed.

The choice of a model-based approach for the formal technique was motivated by a number of factors amongst which the followings were most appealing:

- a systematic literature review conducted by Amyot and Mussbacher [14] revealed that the authors' previous work suggesting the use of UCM as an intermediate step to produce a Z/Object-Z specification was (at that time) the first attempt to formalise the part of URN that describes functional requirements. Thus, the need to extend the work to cover GRL models emerged, although more challenging because GRL explicitly includes mechanisms to specify goals describing quality and non-functional requirements whereas, the formal technique does not.

- since URN is model-based, the structural similitude with the formal technique has the advantage that the analysis, for example, of the relationships between the elements of both: URN models and those of the chosen model-based formal technique (Z/Object-Z) can be facilitated by the knowledge of their commonalities.

- since formal methods, in general, share in common the use of mathematical expressions to describe properties of the system, a successful formalisation with a model-based approach would provide for insight or building blocks for other type of formal techniques.

- another important reason to work with model-based formal techniques is their explicit recommendation in software requirements engineering (van Lamsweerde [180]). Z/Object-Z being one of the traditionally successful formal specification techniques, we believe its choice to be adequate for this work.

This work therefore, aims to address the complex problem of integrating formal methods into the existing traditional software specification process through the coupling of URN, Z and

[2]Object-Z is an Object-Oriented extension of Z.

Object-Z. The purpose being, for example, to exploit the flexibility of the semi-formal technique (because of its graphical nature, which is inherently more human-oriented) to facilitate the building of the formal specifications, and when possible, use the mathematical precision of the formal technique to improve on the initial semi-formal model. This research could therefore result in facilitating the use of formal specification techniques and consequently encourage greater use of formal methods in industry.

A URN model of a system comprises of two complementary sub-models: a UCM model to describe functional requirements, and a GRL model, which incorporates goals that need to be achieved by the functional requirements, as well as goals describing non-functional requirements. As part of the same project, earlier work by the researchers [61] was conducted to establish the formalisation of UCMs with Z and Object-Z and demonstrated the usefulness of UCMs in the process of constructing Z and Object-Z specifications. The current work therefore, focuses mainly on the GRL modeling technique. The case of the GRL model brings in more complexity because, neither standard Z nor Object-Z, naturally, integrates any mechanism for non-functional requirements. Furthermore, an initial GRL model encompasses both goals describing functional and non-functional requirements.

Some of the research questions used to guide the authors through this research work are presented next.

1.3.1 Research questions

The following research questions (RQs) are formulated to address the main problem stated above. The purpose being to investigate two issues: Firstly, how to integrate formal methods into the existing software specification process by coupling such methods to semi-formal techniques. Secondly, to investigate to what extent the flexibility of semi-formal software specification techniques can help to alleviate the complexity of formal ones and hence, facilitate their practical use.

RQ 1: What lightweight (enterprise) model could there be to facilitate the process of goals and/or requirements elicitation at an initial phase of requirements elicitation and analysis? Such a model would naturally constitute a starting point for the construction of the system.

RQ 2: To what extent could a URN model construction process take advantage of any lightweight model that would result from the research question RQ 1?

RQ 3: To what extent are goals describing non-functional requirements formalisable?

(a) To what extent can a GRL model, describing both functional and non-functional requirements, serve as input to a formal specification techniques, case of Z/Object-Z?

(b) To what extent is it possible to formalise a goal model describing both functional and non-functional requirements in Z and Object-Z?

RQ 4: What would the impact of a semi-formal modelling technique/method for non-functional requirements be on the process of constructing a formal specification? The impact of GRL on a Z/Object-Z specification process will be considered in this work; especially when a GRL model is used as input to Z/Object-Z.

RQ 5: To what extent can a formal specification of a GRL model help to improve on the process and quality of URN models?

Although the full value of a research project like the present one might not be completely realised at this stage, the next section presents its most valuable significance.

1.3.2 Value of the research

The User Requirements Notation (URN) is a flexible technique used to capture, analyse and structure functional and non-functional requirements at an early stage of a software system development cycle. It provides for a rich requirements manipulation environment, where GRL and UCM models are constructed flexibly. The chief advantage of this work is to investigate the possibilities to exploit the facilities of such a technique to build an interactive and iterative environment for URN and Z/Object-Z specifications, where URN is used as an interface to facilitate their use in industry.

1.4 Research objectives

As mentioned earlier, the main purpose is to provide a comprehensive mechanism for the building of Object-Z specifications, where the URN notation is used at an early stage of the specification process to capture and structure the project statement. Research objectives are therefore to:

RObj 1: Investigate existing enterprise models/architectures to determine a synthetic model or approach that may facilitate a preliminary identification of appropriate information sources pertaining to scope definition and user requirements at an early phase of goals/requirements elicitation.

RObj 2: Determine/derive from the existing processes, a suitable URN modelling mechanism that may take advantage of the above-mentioned model or approach and set the foundation for the formalisation and validation of further design models.

RObj 3: Analyse the impact of non-functional requirements on the software development process and resulting products to determine/derive means to formally specify those requirements together with functional requirements embedded in GRL models.

RObj 4: Demonstrate that a GRL model with goals describing functional requirements as well as quality and non-functional requirements of a system can indeed be formalised by providing a formalisation mechanism.

RObj 5: Establish the usefulness of the proposed GRL formalisation mechanism by applying it to a reasonable size case study to produce a formal specification of the GRL model of the case study.

RObj 6: Establish the impact of using URN in the process of generating Z/Object-Z specifications of non-functional requirements. This would be by investigating the presence or absence in the formal specification of the GRL model of the case study, of commonly accepted or selected qualities of a good formal specification.

Some delineations and limitations are presented next.

1.5 Delineations, limitations and assumptions

The following are to be considered.

1.5.1 The scope

As mentioned in the previous sections, this research is about coupling semi-formal and formal requirements analysis, modelling and specification techniques. The case of URN, Z and Object-Z are considered. However, since in previous work, part of the same project, was performed on the coupling of UCM (describing functional requirements), Z, and Object-Z, this research is mainly about the formalisation of (GRL) models, describing functional and non-functional requirements with Object-Z. The scope of the research includes therefore, the:

- Construction process of semi-formal models of URN, focused on GRL. This is worth considering due to the importance the process may have on the constructed model.

- Development of strategies to transform URN models into Z/Object-Z specifications; and if possible, extend the strategies to include other techniques.

- Enhancement of URN models by the formal techniques.

- Generalisation of the results to other semi-formal and/or formal techniques will also be investigated. This is especially about extending the proposed formalisation mechanism to other semi-formal and formal techniques.

1.5.2 Delineations and limitations

Due to time and resources constraints, further development beyond the formal specification of any of the two URN models are beyond the scope of this research work. However, the formal specifications that are created are validated to ensure that they are left in good standards and can therefore be used effectively in subsequent software development phases.

Since the purpose of the study is to investigate the ability to formalise URN models, GRL elements pertaining to the evaluation of the models which are normally useful during the model construction and analysis, are not considered for the formalisation. The formalisation of those elements and the evaluation process may constitute a good (MSc) research topic on its own.

The formalisation of functional requirements, described in UCMs, was performed in an earlier work [60]. Since the Z/Object-Z specifications of UCMs models hence, developed were fully analysed and discussed, whenever they are needed in the current work, they will be used without further validation.

1.5.3 Assumptions

As far as the researchers are aware, based on a systematic literature review on URN [14], this research is the first to embark on combining URN, Z and Object-Z. Thus, from the beginning, it assumes a twofold intermediate processes, during the transformation of a URN model into Z/Object-Z specifications. The first process is to convert UCM diagrams describing functional requirements, into Z and Object-Z [60]. The second process is to formalise GRL diagrams, describing non-functional with Z and Object-Z. The main idea is that, by performing the above two transformations separately, the final Z/Object-Z specification of the initial URN model may be obtained by combining the two Object-Z specifications resulting from the two intermediate transformations.

1.6 Contributions

The potential contribution is to address the traditional problem of the industrial use of formal specification methods and techniques and improve on the quality of the semi-formal processes and models. It aims to exploit the flexibility (and usability) of URN, to address the integration of such methods into existing software development processes, as well as their ease of use, especially in the case of the Z and Object-Z specification techniques. Some of the major contributions of this research are presented next.

1. A proposed lightweight (enterprise) model to facilitate scope definition as well as goals/requirements elicitation and analysis.

2. A proposed goal elicitation process (that may take advantage of the above lightweight model), suitable to generate appropriate URN models, for a given set of stakeholder goals and user requirements.

3. An approach to model and propagate the influence of NFRs beyond the initial elicitation and analysis phase with the proposed concept of Complementary Non-functional actions (CNF-actions).

4. A framework to transform a URN model into an Object-Z specification.

5. A comprehensive construction mechanism for Z/Object-Z, where URN is used at the interface level, to facilitate user requirements elicitation and analysis.

6. An evaluation of the impact of using URN on the quality of a Z/Object-Z specification obtained by formalising a URN model.

7. A systematic literature search to determine the current state of research in NFRs specification, in general, and their formalisation in particular.

To contextualise the importance of the above contributions, further discussion and elaboration of each or a group of these were performed during the research process, in the form of peer-reviewed conference papers or accredited journal articles. Those that were finalised and published are listed on page xv.

The next section presents the list of chapters.

1.7 Thesis layout

1.7.1 List of chapters

Chapter 1 presents the background to the study which is the basic for the motivation. It also includes an overview of the existing approach to combining semi-formal and formal software techniques. The problem statement comprising the research questions and the significance of the work are also included, as well as the objectives, the delineations and limitations, the scope of the research and the thesis layout.

Chapter 2 covers the literature on the User Requirements Notation, namely, URN, the Z notation and its object-oriented version Object-Z. A systematic literature review on non-functional requirements specification and formalisation is also presented.

Chapter 3 discusses the research design and methodology; the approach adopted in this research work to address the research problem, as well as the chosen methods are presented and justified.

Chapter 4 presents an analysis of the URN construction and the impact of non-functional requirements on the software development process, as well as on the final software product. An approach to formalise conceptual elements of GRL is proposed, including elements describing quality and non-functional requirements. A framework proposed to formalise an input GRL model with Z/Object-Z is also discussed.

Chapter 5 investigates the applicability of the formalisation mechanisms developed in Chapter 4 by applying them to a reasonable size case study. An approach suggesting the use of enterprise organograms to facilitate the goal elicitation process [63] is first discussed followed by the GRL modeling of the case study and its formalisation.

Chapter 6 discusses the validation of the Object-Z specification of the case study. An approach proposed to animate an Object-Z specification with Prolog [62] is first presented followed by the overall planning of the validation based on the four-way framework for validating a software specification [61]. The properties against which the specification is to be validated are defined and the validation discussed in details thereafter.

Chapter 7 explores the previous chapters to identify and analyse each contribution. The possibilities to generalise those contributions are discussed as well.

Chapter 8 summarises the main findings, presents the conclusion of the research work and discusses the future research work.

1.7.2 Relationships among the chapters

The tasks and possible outputs presented in each chapter, as well as relationships between the chapters are depicted in Figure 1.2.

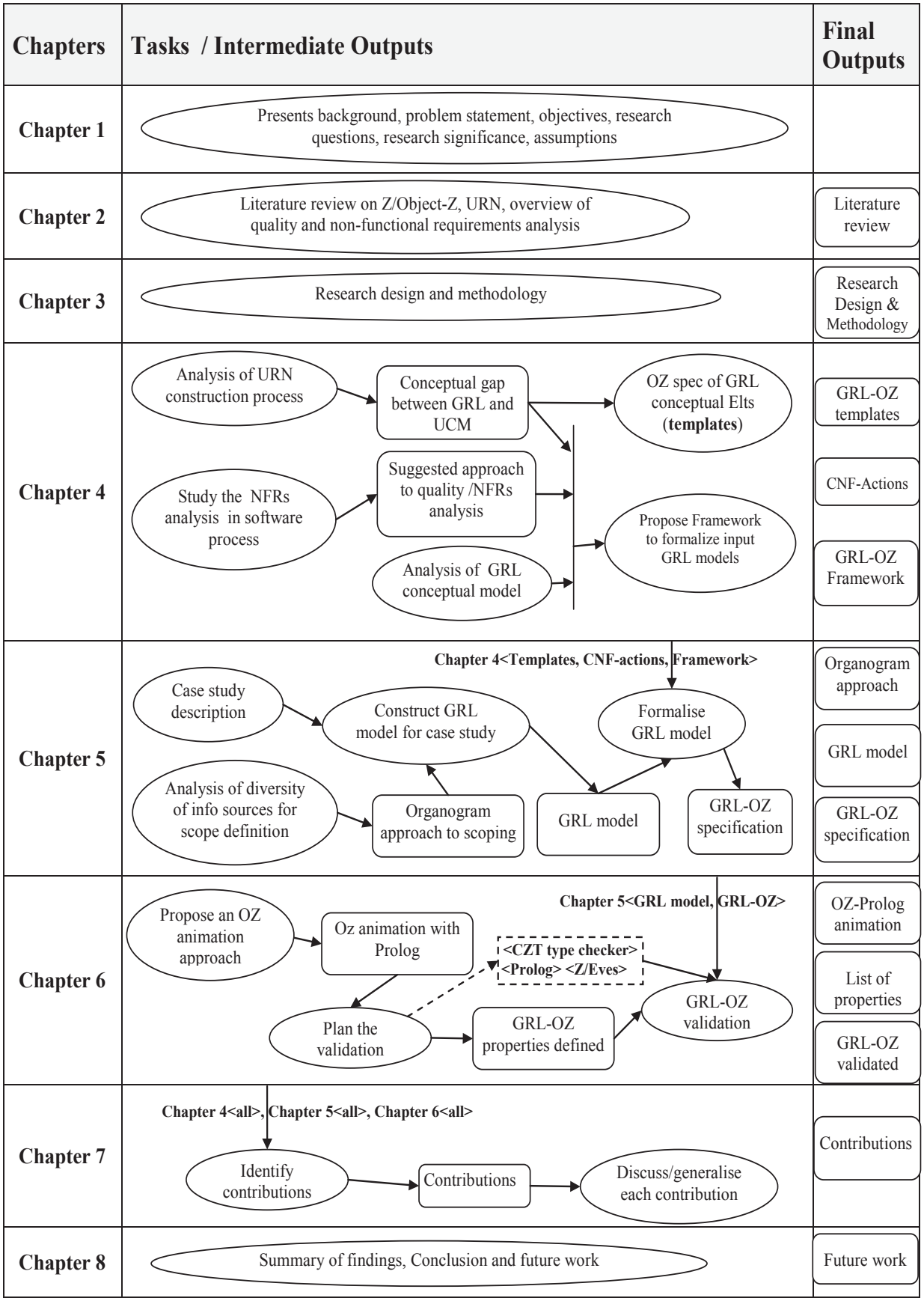

Figure 1.2: Relationships among chapters and main tasks/outputs

Chapter 2

Literature review

The previous chapter elaborated on the motivations and the significance of this research, as well as the layout of this document. This chapter presents an overview of the URN, Z and Object-Z. The result of a systematic literature review pertaining to uncover the current trends in non-functional requirements specification and formalisation is also presented. Much of the definitions on URN elements belonging either to GRL or UCM were taken from (or inspired by) the content of the URN standard document [1]. Since UCM and Z basic notational elements and their semantics have remained unchanged since our last studies [60], some parts of the literature from the studies were reproduced, with and/or without modifications, in the sections covering the literature on UCM (see Section 2.3) and Z/Object-Z (see Sections 2.4 and 2.5) below.

The overview of the User Requirements Notation is first presented.

2.1 The User Requirements Notation (URN)

2.1.1 Brief description of URN

The User Requirements Notation (URN) is a standardised [1] semi-formal, visual, requirements notation that enables the elicitation, modelling and specification, as well as the analysis and validation of user requirements including stakeholder goals (Amyot and Mussbacher [13, 14]). It comprises two complementary languages: the first is the Goal-oriented Requirement Language (GRL) to describe stakeholder goals, including non-functional requirements (Amyot [10]). The second component of URN is Use Case Maps (UCMs) to describe functional requirements and architecture. URN links are also included to relate GRL elements to those of UCMs.

2.1.2 The URN metamodel

The URN basic structural features in Figure 2.1 describes containers for URN, GRL, and UCM specifications [1]. The URN meta-model, namely, URNspec is the root element of

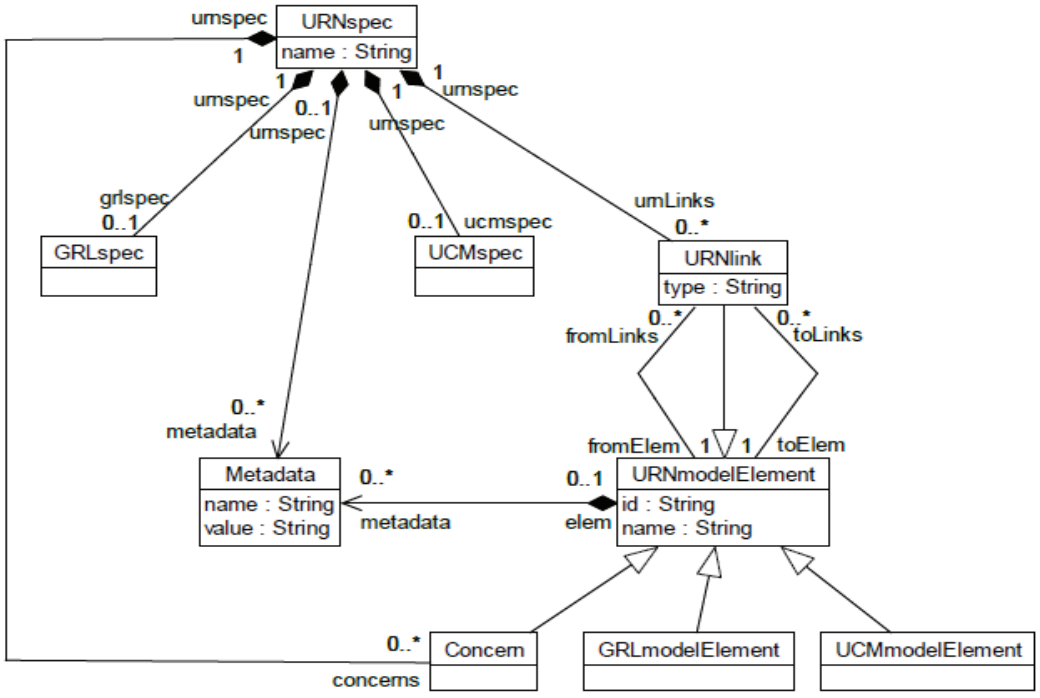

Figure 2.1: URN conceptual model [1]

the URN model or specification. The three other important elements of the specification include:

- GRLspec is the meta-model for GRL specifications which building blocks are GRL model elements (GRLmodelElement).

- UCMspec is the meta-model or container for UCM specifications. Similarly to GRL specifications, the building blocks for a UCM specification are instances of the UCM model elements (UCMmodelElement).

- URNlink is a meta-model or a container for all relationships between URN model elements, especially GRL and UCM model elements. Instances of URN link elements are links connecting URN model elements from one to another.

The basic URN construction approach is presented next.

2.1.3 The URN construction process

As supported by Liu and Yu [110], the basic URN construction methodology relies on a top-down decomposition strategy. The purpose being, for instance, to justify, in a hierarchical structure, the functional and non-functional requirements of a software system. Generally, the higher-level objectives are progressively refined into more fine-grained conceptual elements to form a graph that constitutes a GRL model, which can thereafter be evaluated to establish the level of satisfaction of the initial objectives. Three algorithms have been proposed and implemented for the GRL model evaluation (see Amyot et al. [16]). However, one may use any other appropriate evaluation strategy to identify and analyse the elements of the model.

In the same vein, with UCM, initial requirements in the form of use cases or scenarios, as well as functional requirements from a GRL model are represented with UCM graphical symbols to form the initial UCM model. Depending on the expected level of detail, the initial UCM specification is progressively decomposed/refined until a satisfactory level is obtained.

Goal-Oriented Requirements Engineering techniques are increasingly becoming successful in the software development industry. These techniques provide, amongst others, the means to analyse business objectives (and stakeholder goals), and the traceability of links from high-level strategic objectives to low-level system requirements, as well as support for requirements elaboration, verification/validation, conflict management, negotiation, explanation and evolution as typified by Sen and Hemachandran [162].

URN model specification and evaluation are fully supported by existing tools which are presented next.

2.1.4 The URN tool support: UCMNav and jUCMNav

The UCM Navigator (UCMNav) (Miga [124]) is the oldest tool designed and implemented for UCM; the tool is now obsolete. The more recent and the one which is currently and massively used is jUCMNav (Mussbacher and Amyot [135], Roy et al. [159]).

UCMNav is a graphical software system that helps to create UCMs diagrams. This tool supports most of the features defined in the UCM reference manual (Buhr and Casselman [37]). It maintains binding between plug-ins and stubs, responsibilities to components, sub-components to components etc. It allows users to visit and edit the plug-ins related to stubs at all levels. It loads, exports and imports UCM as XML files. It can also export a UCM di-

agram to formats such as Encapsulated Postscript (EPS), Maker Interchange Format (MIF), and Computer Graphics Meta-file (CGM). As reported by Jason [96], the main drawback of this tool is that it is hard to install and maintain.

The jUCMNav tool is a user-friendly graphical editor under the Java-based open-source Eclipse platform. As an improved version of UCMNav, it provides more functionality including a support for Goal-oriented Requirement Language(GRL). Its export-import possibilities are various, and include the generation from an input URN specification (including GRL and/or UCM models) of different types of files such as XML files, MSC (Message Sequence Charts) files, and the CSM (Core Scenario Model) files.

One of the strength of jUCMNav is the ability to serve as an integrated environment for UCM and GRL models. The tool provides for a visual link between the two models and implements strategies for evaluating GRL models and mechanisms for UCM path traversal (see Amyot et al. [17], Mussbacher et al. [133]).

2.1.5 Current approach to formalise URN models

To gain advantage from the precision of formal techniques (while preserving the flexibility of semi-formal notations), various research works focused on combining semi-formal and formal techniques have been carried out (e.g., Abderrahman et al. [3], Dongmo [60]). However, as far as the authors of this work are aware and based on a comprehensive literature survey on URN conducted by Amyot and Mussbacher [14], this study is the first to attempt the formalisation of GRL with Z/Object-Z. With regards to UCMs, noticeable efforts have been done in the same direction.

Heuristics were proposed by van der Poll et al. [178] to combine UCMs and formal methods purposing to represent and check the validity of scenarios as user requirements. The idea of translating UCMs to communicating state machines was also put forth by Bordeleau and Buhr [29] followed, for example by similar techniques developed to derive message sequence charts (MSC) from UCMs by Miga et al. [125]. To gain more benefits from the validation and testing power of the formal notation LOTOS, an approach was developed by Amyot et al. [18] to translate UCM scenarios into high-level LOTOS specifications. In the same vein, Mokhati and Menassel [129] proposed the formalisation of UCMs in Maude.

During the initial phase of this research work consisting to couple URN, Z and Object-Z, an approach to transform a UCM into Z then, Object-Z was proposed, as well as a framework to

evaluate the impact of UCM on the final Object-Z specification [60]. Such a transformation benefits from the advantage that UCM, Z, and Object-Z collectively address the functional requirements of a system. The present research further advances this idea by coupling GRL and Z/Object-Z to address both functional and non-functional system requirements at an earlier stage of requirements analysis. Z/Object-Z focuses on functional requirements, and does not naturally incorporate mechanisms for NFRs.

An overview of the GRL notation is presented next.

2.2 The GRL notation

2.2.1 The GRL representation

As part of the User Requirements Notation, and to our knowledge, GRL is the first standard goal-oriented requirements language [1]. It is a visual modelling notation that aims to address the "Why" of a system at the requirements level, using concepts such as actors, goals, softgoals, tasks, beliefs and resources to conceptualise requirements artifacts and the relationships between those artifacts (URN[1], Roy et al. [159], Yu and Liu [198]). Likewise its counterpart UCM, the GRL notation uses graphical symbols to model conceptual elements, their decomposition and relationships. Figure 2.2 shows an example of a GRL model pertaining to reason about the requirements for patient safety. Two stakeholders are represented: the first is the government who intends to reduce government costs and force health acts. The second is the hospital that intends to improve patient safety and by doing so would consequently contribute positively to improve patient care, satisfy legal requirements, decrease patient care costs and lawsuits, and also to achieve a competitive edge and recognition of excellence. By decreasing lawsuits and patient care costs would positively contribute to decrease the overall costs and hence satisfying the government goal.

The list of notational elements used to construct a GRL model are presented and discussed next.

2.2.2 The GRL concepts and notational elements

The three main categories of concepts in GRL include: actors, intentional elements, and links. A specific graphical element is used to represent each concept.

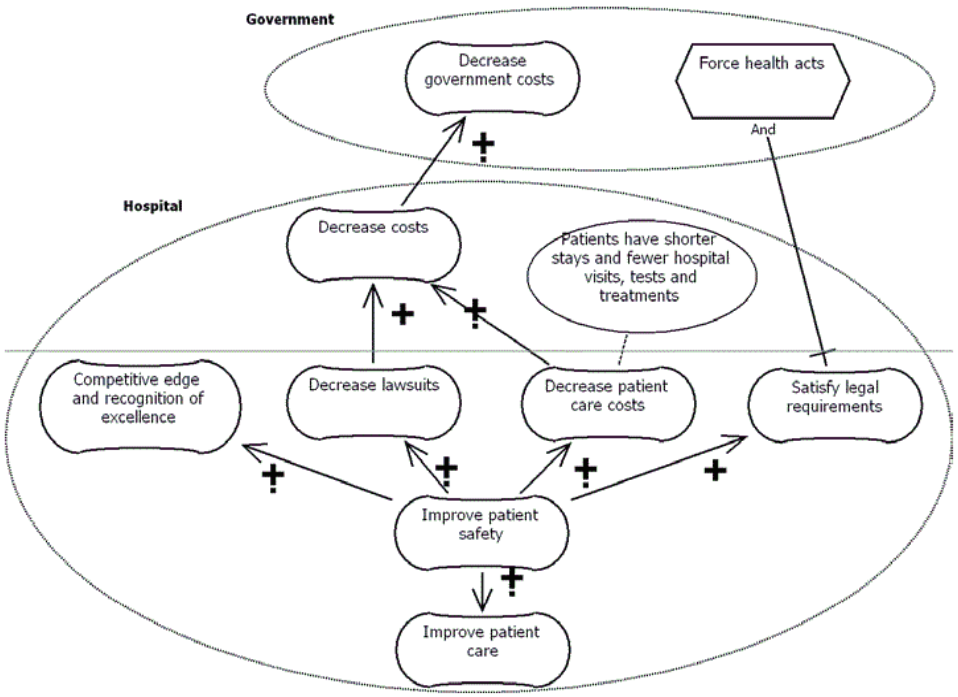

Figure 2.2: Example of a GRL model [24]

GRL Actors

Figure 2.3 shows the graphical representation of an actor. An actor's definition is a concept

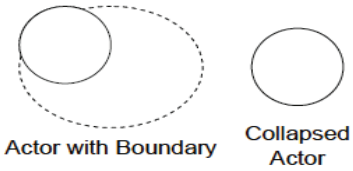

Figure 2.3: GRL actor notation

used to specify active entities that have intentions and the ability to carry out actions to achieve its goals or satisfy its softgoals. An actor may therefore contain intentional elements but, never other actors. In a GRL graph, an actor generally represents a stakeholder or a (component of a) system. As represented in Figure 2.3, its graphical representation can be with boundary or collapsed.

GRL intentional elements

Intentional elements are those addressing the why of the system. The graphical symbols used in GRL to model this category of concepts are depicted in Figure 2.4.

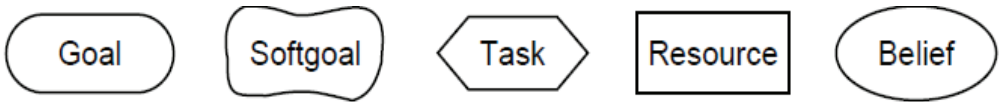

Figure 2.4: Basic GRL notational elements

A (hard) goal is the GRL concept for a condition or state of affairs in the world that the stakeholders would like to achieve. A goal represents either a business goal or system's goal for which alternative solutions can be derived and evaluated. Goals are graphically represented as rounded rectangles with the goal's name inside.

A softgoal is a goal for which there is no clear, objective measure of satisfaction. Unlike (hard) goals, softgoals are generally related to quality and non-functional requirements. The graphical symbol used to represent a softgoal is an irregular curvilinear shape with the softgoal's name inside.

A task specifies a generic activity that is generally derived by decomposition/refinement of other intentional elements to achieve goals or operationalise softgoals. In GRL graphical representation, tasks are represented as a hexagon with the task name inside.

A resource represents an entity, physical or informational, needed to perform a task, achieve a goal or satisfy a softgoal. The availability of the resource is the most important property that is required. In a GRL model, a resource is graphically represented as a rectangle with the resource's name inside.

A belief is used to represent design rationale. Beliefs make it possible for domain characteristics to be considered and properly reflected in the decision-making process, hence facilitating later review, justification and change of the system, as well as enhancing traceability.

GRL links

As illustrated in Figure 2.5, five different types of links are used in a GRL goal graph to specify the connections between the intentional elements that optionally reside within an actor boundary.

A contribution /correlation link specifies the level of impact the satisfaction of a source intentional element has on the satisfaction of the destination intentional element. The impact of a contribution link can be either qualitative or quantitative. Unlike the contribution link,

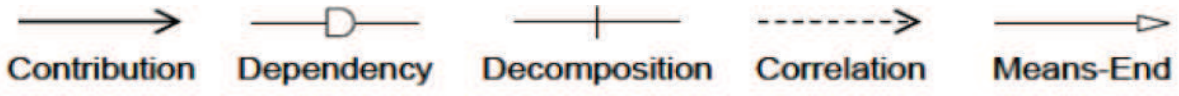

Figure 2.5: List of GRL links

a correlation emphasizes side effects between intentional elements in different categories or actor definitions. The symbols used to annotate each type of a contribution are depicted in Figure 2.6. The impact of a source intentional element on the satisfaction of a destination

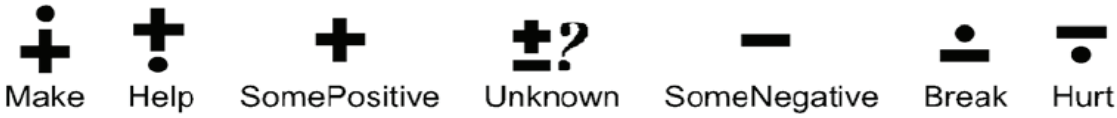

Figure 2.6: GRL contribution types

intentional element is said to be of type `Make`, if the contribution is positive and sufficient, `Help` if the contribution is positive but not sufficient, `SomePositive` if the contribution is positive but the extent of the contribution unknown. The type is `Unknown` if there is some contribution but the extent of the degree of the contribution is unknown, `SomeNegative` when the contribution is negative but the extent of the contribution is unknown. The type is `Break` when the contribution is negative and sufficient whereas, it is `Hurt` if the contribution is negative but not sufficient.

A dependency link specifies a mechanism to reason about the way actors depend on each other to achieve their goals. In a dependency relationship between two actors, an actor definition, namely, depender depends on a destination actor definition, said dependee for an intentional element.

A decomposition link provides the ability to define what source intentional elements need to be satisfied or available in order for a target intentional element to be satisfied. There are three types of decompositions:

- AND decomposition: the satisfaction of each of the sub-intentional[1] element is necessary to achieve the target.
- XOR decomposition: the satisfaction of one and only one of the sub-intentional elements is necessary to achieve the target.

[1]Sub-intentional element refers to a goal, softgoal, task or resource that decomposes a source intentional element

- IOR decomposition: the satisfaction of one of the sub-intentional elements is sufficient to achieve the target, but many sub-intentional elements can be satisfied.

The concept therefore enables the hierarchical decomposition (AND) of a target intentional element by a source element, as well as the description of alternative means (XOR or IOR) of satisfying a target intentional element.

Means-end are links for tasks that achieve goals for which they are alternative solutions.

2.2.3 The GRL model evaluation

A GRL evaluation is one of the most important and automated GRL model analysis processes implemented in jUCMNav [17] to assess how well goals are achieved. The process requires at the inial phase a strategy, which defines initial satisfaction values (qualitative or quantitative) for some intentional elements in the input goal model. The (jUCMNav) software thereafter, executes various evaluation algorithms to propagate the strategy to other elements of the model, including actors.

GRL satisfaction levels

Figure 2.7 shows some qualitative labels that may result from the evaluation. Each label

Figure 2.7: GRL qualitative labels

indicates a level of satisfaction of an intentional element or actor definition. An intentional element is marked with the symbol `Denied` when it is sufficiently dissatisfied, `WeaklyDenied` when it is partially dissatisfied, `WeaklySatisfied` when it is partially satisfied, `Satisfied` when it is sufficiently satisfied, `Conflict` when there are strong arguments both in favour and against its satisfaction, `Unknown` if the satisfaction level is not known, and `None` when the element is neither satisfied nor dissatisfied.

GRL model evaluation makes it possible to flexibly test the impact of different strategies (alternative design decisions) on the entire model during the model analysis.

2.3 Use Case Maps (UCMs)

UCM is a scenario-based, semi-formal requirements notation technique, originally developed by Buhr and his team (Buhr [34], Buhr and Casselman [37]), to bridge the gap between requirements and design. The notation is strengthened by the use of simple graphical elements to describe, in a map-like diagram, service functionalities superimposed on the organisational structure of complex and distributed systems (Amyot et al. [18]).

2.3.1 The UCM model representation

A UCM model presents a holistic view of the system under construction without commitment to details. Use Case Maps accept as inputs user requirements, either expressed in natural language, or transformed into Use Cases as indicated by Amyot [9]. A Use Case may be described as a set of scenarios, which are sequences of actions performed by the system to yield an observable result to its environment (see Booch et al. [28]). An example of a UCM diagram is presented in Figure 2.8. The map specifies a system to handle the return

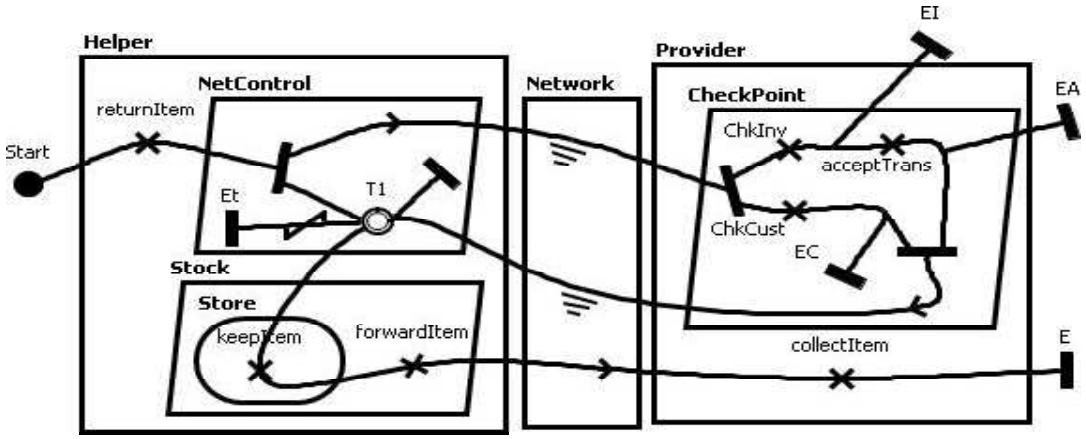

Figure 2.8: An example of a UCM diagram adapted from [60]

of items previously purchased by the customer. The component, named `Helper` models any company where a customer may return a purchased item. The component `Provider` models the company that provided the returned item and the `Network` specifies the means of communication between the helper and the provider companies. The process is triggered whenever a customer wants to return a purchased item at the helper. The helper contacts the provider via the network to check the invoice and customer's information then keeps the item temporary pending the time to forward it to the provider when the transaction is accepted by the provider.

Owing to its flexibility, and the ability to convey different types of information on a single map-like diagram, the UCM notation is applicable (and adaptable) for various purposes, in different domains. Examples include: Object-orientated software design (Buhr [35], Buhr and Casselman [37]), telecommunication networks (Amyot and Andrade [12]), multi-agents software development (Abdelaziz et al. [2], Buhr et al. [38, 39]), Aspect-oriented software development (Mussbacher [134], Mussbacher et al. [136]), Web applications (Kaewkasi and Rivepiboon [100], Liu and Yu [110]), as well as system testing and validation (Amyot et al. [15, 18], Amyot, Daniel and Roy, Jean-François and Weiss, Michael [19]).

2.3.2 The UCM concepts and notational elements

The main concepts in a UCM are: abstract components (to represent the architecture of the system), paths (to represent scenarios), path connectors (to represents scenarios interactions) and path elements including stubs to defer detail of a sub-system to a sub-map called plug-in, waiting places and timers that may be encountered along a path to indicate the place where the progression of a scenario idles until an external event occurs [9].

Abstract components are concepts used to represent the architectural aspect of the system. The five types of UCMs components include: team component, process, object, agent and actor. Path (path segment) and path elements are critical design elements used in UCM to specify scenarios and alternatives as well as scenario execution whereas, path connectors are important artifacts necessary in UCM to represent scenarios'interactions.

Each of the UCM concepts is discussed next starting with abstract components.

2.3.3 UCM abstract components

An abstract component may be viewed as a self-contained operational unit with internal state and links that enable the component to interact with others. Each component is responsible for performing responsibility points located in it and chained with path segments. Different types of components are provided by the UCM notation: Team, Process, Object, Agent and Actor (see Figure 2.9).

Team

A Team component is a generic component allowed to contain any other component type including other teams. In a UCM model, a team component is graphically represented by a labelled rectangle.

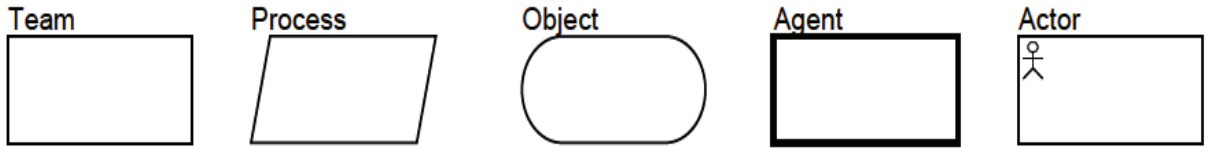

Figure 2.9: Abstract components

Process

A Process is an autonomous, active component, that may operate concurrently with other processes. A process may contain passive components, those that do not have control over the responsibilities that they perform, such as Objects. It is represented graphically by a parallelogram.

Object

An Object is a passive component, that supports data or procedural abstraction through an interface. Objects perform their own responsibilities but do not have ultimate control of when they are activated.

Agent

An Agent is an autonomous component, which acts on behalf of other components. As shown in Figure 2.9, an agent component is graphically represented by a rectangle with a thick border labelled at the top left corner.

Actor

An Actor is an external component that describes an entity, either human or artificial, that interacts with the system. In a UCM map, an actor is graphically represented as a rectangle with a stickman icon in its top-left corner.

2.3.4 Basic path notation

A UCM path is a wiggle line to model the execution route for one or more scenarios, as well as alternative scenarios. A UCM Path may be composed of more than one segment, interconnected by means of path connectors. A path element is placed along the path to describe a specific aspect of the system. Figure 2.10 shows an example of basic path notation. It comprises: a Start Point, a Path Segment, a Responsibility Point, and an End Point.

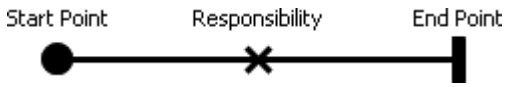

Figure 2.10: Basic UCM path notation

Start point

A start-point specifies the triggering events, necessary, to start the execution of a scenario optionally a precondition. In a UCM model, a start-point is graphically represented by a filled circle labelled with the title / name of the start-point. The execution of a path begins when some triggering events occur with the precondition enabled.

Responsibility

A responsibility point is a UCM concept to specify a generic processing that is to be performed, which can be for example, an operation, a task, an action, a function and so forth. In a UCM model, a Responsibility Point is graphically represented by a cross on the execution path labelled with the title/name of the responsibility point.

End Point

An end-point models a set of resulting events and an optional post-condition that terminates the execution of a scenario along the path that represents it. The symbol used to represent the concept in a UCM specification is a perpendicular bar labelled with the title/name of the end-point.

Path Segment

A UCM path segment is represented graphically by a continuous line with any possible and unambiguous shape. It may sometimes be useful to indicate the direction of a path segment (e.g., with arrow), but in general it is not necessary. A path segment is used to express an ordered sequence of UCM elements that require to be executed.

Use Case Maps provide the concept of path connectors to describe alternative use cases, and parallel executions of scenarios.

2.3.5 Path connectors

A UCM path is the execution route of one or more scenarios, and may be composed of a number of path segments, interconnected by means of path connectors to achieve path coupling, and express interactions between scenarios. Amongst others, path connectors are: OR-forks, OR-joins, AND-forks, and AND-join (see Figure 2.11).

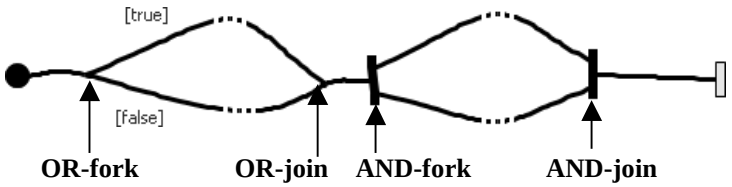

Figure 2.11: Path connectors

OR-fork

An OR-fork splits a path segment into two or more branches. Alternative path segments may be guarded by conditions, depicted inside square brackets.

OR-join

An OR-join is a place on a UCM diagram where two or more path segments merge into a single one. The merging of the path segments does not require any synchronisation or interaction between the incoming paths.

AND-fork

An AND-fork is represented graphically by a vertical ticked bar that splits an incoming path segment into two or more parallel paths. This connector helps to represent the concurrent progression of scenarios along path segments.

AND-join

An AND-join connector collapses two or more parallel paths into a single one. It is represented graphically by a vertical bar.

The AND-fork/join elements provide a strong form of representing inter-scenario synchronisation in which scenarios along different paths are mutually synchronised. The OR-fork/join UCM concept allows for multiple scenarios to progress along a single path segment and be separated independently only where necessary.

Two types of path elements called **stubs** are discussed next.

2.3.6 Stubbing techniques

A UCM provides for the concept of stubs to help sub-divide complex maps into sub-maps. A stub is a mechanism for (paths) abstraction that represents on a UCM diagram, a place where a sub-map is needed, but for which details are referred to elsewhere. It saves as maps

connectors that help to link the execution of a scenario from a map containing the stub (called root-map) to a sub-map called a "plug-in". The two types of stubs are: static-stubs, and dynamic-stubs (see Figure 2.12).

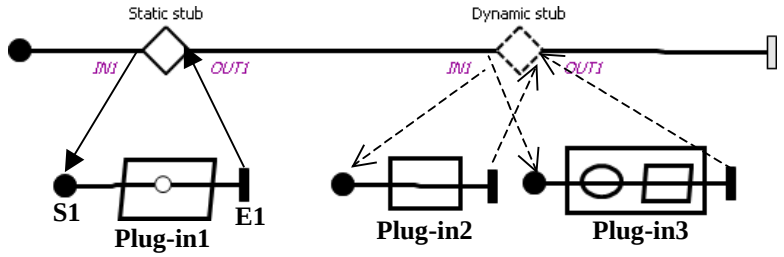

Figure 2.12: An example of a static and a dynamic stub

Static-stub

When only a single sub-map is needed, a static-stub is used. The binding of the plug-in to the root-map is made as follows: the input path segment(s) entering the stub (generally noted INX, where X stands for a referencing number) is (are) associated to the Start point(s) of the plug-in, and the End point(s) of the plug-in is (are) associated to the output path segment(s), leaving the stub (generally noted by $OUTX$ as in Figure 2.12). This association is called a *Binding Relationship*. In the case of the static-stub in Figure 2.12, it is indicated by:$\{\langle IN1, S1\rangle, \langle OUT1, E1\rangle\}$ (Amyot [11]).

Dynamic-stub

A dynamic-stub is used where more than one alternative sub-diagram is needed, for which the binding to a specific diagram is determined during the execution of the scenario being modelled. A selection policy to determine the plug-in to execute is, therefore defined.

Other key notation elements are: Failure-point, Waiting-place and Timer. These are presented next, through the Timeout-recovery mechanism that is provided - by UCM - to model the enhancing of network failures in a network communication.

2.3.7 Timeout-recovery mechanism

Figure 2.13 shows the graphical representation of the three UCM elements: Failure-point, Waiting-place and Timer; it also includes a model for a Timeout-recovery mechanism. In each of the three components in the figure, the path from start-point $S1$ to the end-point $E1$, is called the *main path*. It is the path on which a scenario progresses to reach the

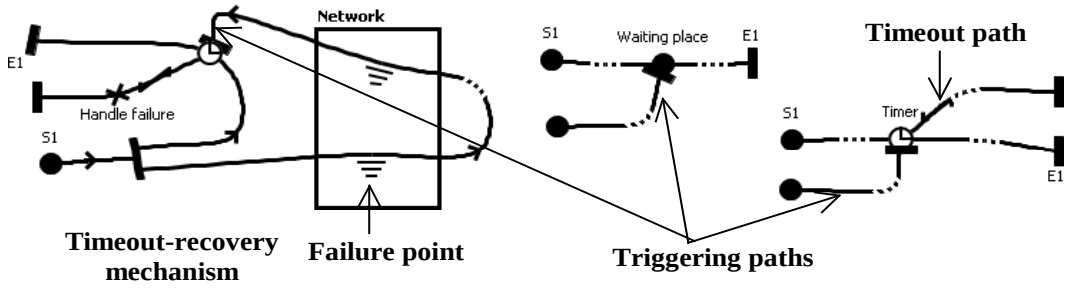

Figure 2.13: Timers

waiting-place or timer. The triggering paths are also indicated. Those are paths along which triggering events occur, to cause a waiting scenario to continue progressing along the main path.

Failure-point

A failure-point indicates a place along a path where the progression of a scenario may stop leaving the system in an incomplete state, possibly jeopardising other paths in execution. For example, a network communication may fail, causing a sent message not to reach its destination or an acknowledgement not to reach the sender.

Waiting-place

A waiting-place indicates a place where a scenario progressing along a main path, may need to pause waiting for an event to occur along the triggering path, before it continues. The triggering path may terminate at the waiting-place or touch it tangentially, and continues. Identifying a path as a main path, or triggering path, is relative, since the same path may play both roles depending on the scenario under consideration.

Timer

A Timer, also known as a timed waiting-place, is just a variation of a waiting-place that uses a time clock to control the occurrence of the triggering event. The timeout path on the diagram in Figure 2.13 is used to model the situation when the waiting time expires before the occurrence of the triggering event.

With the Timeout-Recovery mechanism in Figure 2.13, a message is sent via the network component and concurrently, the Timer is set up to wait for an acknowledgment that may be sent back via the network. If the acknowledgment is not received before timeout, then network communication failure is assumed, and the responsibility point labeled *Handle failure* is performed. Otherwise, the execution continues to the end-point $E1$.

UCM Layering

As advocated by Buhr [36], layering enables a large scale system to be represented by a hierarchy of layers that are described independently, without explicit reference to each other. The concept is that lower layers provide infrastructure for higher ones. It provides a way of describing large scale systems with many levels of infrastructure without becoming bogged down in details of how infrastructure is used.

2.3.8 Extending the original UCM notation

As mentioned earlier, a UCM aims to bridge the gap between user requirements and detailed design (Buhr [34, 35], Buhr and Casselman [37]). Its core notation does not completely cover the notational needs in some specific application domains. Some extensions have been proposed, either to the basic features of a UCM, or to its applicability. Some notational elements and concepts were added to the basic UCM features to support the agent systems (Amyot [9]). As reported by van der Poll et al. [178], UCM support for designing user interfaces is still acknowledged to be insufficient. In this regard, an extension of the basic UCM, that reinforces the exchange of messages between users and the system aimed at allowing the notation to adequately support the user interfaces and usability requirements analysis and modeling was suggested. For a similar reason, a number of heuristics were proposed to facilitate the validation of the three important properties: consistency, completeness and precision.

The following section presents an overview of the Z specification language.

2.4 The Z notation

Z (pronounced 'zed') is a formal specification notation based on the First Order Predicate Logic (FOPL), and a strongly-typed fragment of Zermelo-Fraenkel (ZF) set theory (Bowen [30], Lightfoot [109], Mole [130], O'Regan [142, 142], Spivey [170]). The notation was initiated by Jean-Raymond Abrial in France, and developed at the Programming Research Group (PRG) of Oxford University, in England, since the late 1970s. The main construct in Z is a schema which is built upon basic types and global variables. Z employs schemas to specify the static and dynamic behaviour of systems (Spivey [170]). The generic form of a Z schema is given below.

SchemaName [*ListParameters*]

[*declaration − part*]

[*predicate − part*]

The schema's name is given by *SchemaName*, and a list of parameters to be used within the schema may be specified. The *declaration – part* includes a list of typed variables, called components. Composed types are normally defined from a list of basic types identified during the construction of a specification. The *predicate – part* specifies constraints among components in the declaration, e.g., the state invariant.

2.4.1 Basic types and global sets

The concept of a basic type (also called a given type), is provided in Z, to specify the set of elementary objects, for which details are left unspecified. The list of basic types, for a specification, is enclosed inside square brackets and separated by commas. For example:

$[Customer, Book, Account]$

defines a list of basic types in Z, for which for example, *Customer* specifies the set of all possible customers. Detail information about customers, books and accounts are deferred to the design phase. A basic type may be used anywhere in the specification after its definition (see Bowen [30]).

Similar to basic types, a global variable may be used anywhere in the specification after its definition. The axiomatic definition of a global variable is presented as follows:

declaration part

---

predicate part

For example:

$max : \mathbb{N}$

$max \leq 50$

The concept of **Free types** is also used to list, for a type, the identifiers of its element. The general form is:

$freetype ::= element_1 \mid element_2 \mid ... \mid element_n$

For example, $Response ::= yes \mid no$
The central concept in Z is the Schema introduced next.

2.4.2 Z schemas

The abbreviated notation of the above schema is:

$$SchemaName == [declaration\ part \mid predicate\ part]$$

Two types of schemas are encountered: "state schemas", to describe the static behaviour of a system, and "operation schemas" to describe the dynamic behaviour. For illustration purpose, the Airport example below from Lightfoot [109] is considered:

The air-traffic control of an airport keeps a record of the planes waiting to land and the assignment of planes to gates on the ground.

State schema

In Z, an abstract state, also called a state schema, specifies the static behaviour of a system. For example, with the airport example above, assume the given types:

$$[Plane, Gate]$$

Where *Plane* denotes the set of all possible, uniquely identified planes, and *Gate* the set of all gates at the airport. The state schema is:

Airport

$$waiting : \mathbb{P}\ Plane$$
$$assignment : Gate \rightarrowtail Plane$$

$$waiting \cap (\text{ran}\ assignment) = \varnothing$$

The component *waiting* maintains a list of planes waiting to be assigned to a gate, and *assignment* maps each gate to one, and only one, plane. The predicate part indicates that only planes that have not yet been assigned a gate, are kept in the waiting list. An important aspect of a system state is its inherent variability in time, e.g., when a new plane is assigned a gate, the value of each of the two components *waiting* and *assignment* changes and hence, the state of *Airport*. Z provides an operation called "schema decoration" to describe the change of system states.

Schema decoration

A schema S is decorated by adding a prime to its name (S'). The effect of decorating S, is that all the variables in the declaration and predicate part of S, are also decorated (see Potter et al. [150]). Since an operation performed on a state schema may change the state of the system, an important aspect of schema decoration is to facilitate the specification

state change within the operation schema. The state before and after the operation, are both included in the declaration of an operation schema, and related in the predicate part, to show, for example, how state variables are changed by the operation.

Schema as a type

To define composite (complex) structures, Z allows a schema to be used as a type (Jacky [95], Van der Poll [176]). Such a type is similar to a record type in conventional programming languages such as Pascal. An instance of a schema type is called a binding. Z provides the unary operator θ to reference each binding. For example, an instance of the schema *Airport* is:

$$\langle waiting \Rightarrow \varnothing, assignment \Rightarrow \varnothing \rangle$$

For each abstract state space, a realisable initial state is required.

Initialising the state space

It may be assumed that initially, the list of planes in the waiting list is empty and the list of gates assigned to planes is also empty. Therefore the state of the *Airport* is initially represented as:

InitAirport

$$Airport'$$

$$waiting' = \varnothing \land assignment' = \varnothing$$

Although it is relatively easy to observe that this state is realisable, in general, it is recommended to establish that the initial state is realisable. To this end, the initialisation theorem is used:

$$\vdash Airport' \bullet InitAirport$$

This implies the need to demonstrate that there exists a state $Airport'$ of the state space *Airport*, for which the components $waiting = \varnothing$ and $assignment = \varnothing$.

Partial operation

To illustrate the concept of an operation in Z, consider the following schema that assigns a gate to a plane.

assignGate

$\Delta Airport$
$plane? : Plane$
$gate? : Gate$

$plane? \in waiting$
$assignment' = assignment \cup \{gate? \mapsto plane?)\}$
$waiting' = waiting \setminus \{plane?\}$

The delta (Δ) symbol is used to indicate the state schema that the operation changes. The question mark (?) that follows the two variables *plane*? and *gate*? indicates that those are input variables. An exclamation mark (!) is used to denote an output.

The logical expression $plane? \in waiting$ in the predicate part, constraints the input plane to be taken only from the waiting list. This defines the condition under which the operation becomes applicable, i.e the precondition. The precondition of each operation may be calculated (Woodcock [193]) to determine the circumstances under which an operation is applicable. For example, if the input plane is not in the waiting list, an error is generated and further operations are needed to handle the error. Hence, *assignGate* is said to be a partial operation, since further operations may be needed to specify error conditions.

Error condition

As mentioned in the previous section, if a plane used as input in the operation *assignGate* is not in the waiting list, an error occurs and the following operation is specified for the error case.

unknownPlane

$\Xi Airport$
$plane? : Plane$
$resp! : Response$

$plane? \notin waiting$
$resp! = PLANE\_UNKNOWN$

The symbol Ξ is used to indicate that the operation operates on *Airport* but, does not change its state.

Total operation

In Z, a complete version of the operation that maps each plane to a specific gate may be formed by combining the operation under normal circumstances, and those to handle errors.

$$totalAssignment \mathrel{\widehat=} assignGate \lor unknownPlane$$

The definition of the operation *totalAssignment* is a predicate schema expression that uses the Z disjunction operator $\vee$, to combine two operations. The semantics of this operation is the following: The declaration part of the composed operation, is obtained by merging the declarations of each of the individual operations. The predicates of the individual schemas are disjoined. More schema operators are available to facilitate the construction of predicate schema expressions.

Schema calculus

Amongst others, the following operators are provided in Z: schema inclusion, schema conjunction ($\wedge$), schema negation ($\neg$) and sequential composition ($\fatsemi$), (see Potter et al. [150]).

(a) **Schema inclusion**: This operator allows the name of a schema $S1$, describing an abstract state space to be included in the declaration part of another state space schema $S2$. The declarations of $S1$ are included in those of $S2$, and the predicate of $S1$ is appended (ored) to that of $S2$.

(b) **Schema negation ($\neg$)**: The negation of a schema S, is a schema denoted by $\neg\, S$. It has the same declarations as S, and its predicate, is the negation of the predicate of S.

(c) **Schema conjunction ($\wedge$)**: Let R and S be two schemas, and $P = R \ \wedge \ S$. P is a schema obtained as follows: the declarations of R and S are merged to form that of P and their predicates are conjoined (anded) to form that of P.

(d) **Schema composition ($\fatsemi$)**: Consider an operation C, defined as: $C = A \fatsemi B$ where A and B are two operation schemas. The semantics of C is the following: if the operation A can change the state of the system from S to $S1$, and B from $S1$ to $S2$, then C is an operation that changes the state of the system from S to $S2$.

Some limitations of Z due to schema calculus and the use of schemas as types were analysed by Van der Poll [176]. However, the major disadvantage of using Z for large systems is its inherent lack of object-oriented structures, making it hard to group and manage a rapidly increasing number of schema structures. To this end, the notation was extended to Object-Z to accommodate object-orientation (Carrington and Smith [45], Smith [164]). An overview of Object-Z is presented next.

2.5 The Object-Z notation

Object-Z [151, 163, 165, 185] is a super-set of Z that employs the concept of a class schema to encapsulate traditional Z schemas [173]. The generic structure of an Object-Z class is given next.

ClassName [*generic parameters*]

[*visibility*]
[*inherited classes*]
[*local definitions*]
[*state schema*]
[*initial state schema*]
[*operations*]

The generic parameters list is optional (as well as each component within the class). The visibility list denoted by $\restriction$, restricts access to some components and operations of the class. Similarly, the list of inherited classes is optional. Type and constant definitions may be specified. A class schema may include only one state schema. The components in the state schema may be initialised to some realisable values. Unlike in Z, operations and the state schemas are described within the class. The order in which those components appear is prescriptive.

Z operators are used, within a class, to combine operation schemas for specifying complex activities. Class-level operations are also allowed, for example, to construct complex classes, or to create references to components within a subclass. As mentioned by Taylor et al. [174], Object-Z (see Duke and Rose [68], Duke et al. [69], Smith [163]) is one of the most developed of several Z-like Object-Oriented specification languages. It employs the concept of a class schema to encapsulate Z schemas. A class schema may include only one state schema, which is very similar to Z state schemas, and does not carry a name. The components in the state schema may be initialised to some realisable values. The only initial state is named *INIT*. It includes only instances of the components declared in the state schema. Operations are described in the same vein as in Z, with some differences as indicated next.

2.5.1 Operation schema

The concept of an operation in Object-Z is similar to that of Z. The only difference is that an operation in Object-Z operates on a single state schema. The Delta (Δ) operator lists specific components changed by the operation, whereas the Xi (Ξ) operator is simply discarded in Object-Z. The concepts of partial, total operations and error handling, are not provided since an operation in Object-Z becomes applicable only when the precondition of the operation is satisfied. Most of the Z schema calculus operators (e.g., $\vee$, $\wedge$, $\fatsemi$, *etc.*), are also used in Object-Z. However, the semantics of some of them vary slightly in the context of a class schema. Additional schema operators are also provided. Two examples are: the nondeterministic choice ($[]$) and scope enrichment operators ($\bullet$) (see Duke and Rose [68], Smith [163]).

An example of a class schema to specify the airport example from Section 2.4 is given next. The two components (*waiting* and *assignment*) and the operation *totalAssignment* are made accessible from the system environment.

ClsAirport

$\restriction(waiting, assignment, totalAssignment)$

$[Plane, Gate, Response]$

$waiting : \mathbb{P}\, Plane$

$assignment : Gate \rightarrowtail Plane$

$waiting \cap (\text{ran}\, assignment) = \varnothing$

INIT

$waiting = \varnothing \land assignment = \varnothing$

assignGate

$\Delta(waiting, assignment)$

only the parameters of the Delta operator has changed

unknownPlane

only the Xi operator is removed

$totalAssignment \mathrel{\widehat=}$

$[plane? : Plane, gate? : Gate] \bullet$

$assignGate$

$[]$

$unknownPlane$

The choice operator ([]) is used in the definition of the operation *totalAssignemt*, allowing the system to choose one of the two alternative operations *assignGate* and *unknownPlane* without user intervention. The variables in square brackets are those for which input values are expected from the system environment. The operator $\bullet$ is used to promote[2], when necessary, operations through the selected objects (in square brackets). This operator has the advantage of providing a way to inherit operations from objects of other classes. The concept of inheritance, discussed below, may be introduced in the definition of a class in different ways.

[2] **promotion** allows for the reuse of an operation to specify another one

2.5.2 Inheritance

The concept of inheritance allows for the reuse of features of an inherited class (the superclass) when creating a new class schema (the subclass). As mentioned earlier, Object-Z provides different specification constructs to define the inheritance mechanism, e.g., through class inclusion, by using a class as a type or promoting an operation.

(a) ***Class inclusion***
The name of the inherited class is listed in the declaration of the inheriting class. In that case, the type and constants of both classes are merged as well as their schemas. But, state schemas as well as those that share the same name are joined. The visibility list is not inherited.

(b) ***Class schema as a type***
Consider the following declaration allowed in Object-Z where *ClsAirport* is the class defined earlier:

$$orTambo : ClsAirport$$

This definition specifies the variable *orTambo* as an identifier of an object of the class *ClsAirport*. Object identity is modelled in Object-Z by associating with each class name a countable infinite set of values (Smith [163]). Through the variable *orTambo* and the dot (.) notation, the features of the class *ClsAirport* become accessible to the class in which it is declared. For example, an operation may change the state of the referenced object as follows: $orTambo.waiting' = orTambo.waiting \cup \{plane1\}$, where $plane1$ is of type *Plane*.

(c) ***Operation promotion***
The scope enrichment operator ($\bullet$), the dot and the possibility to use a class as a type in Object-Z provide meaningful ways to specify the reuse of operations. Consider for example the following operation:

$$newAssign \mathrel{\widehat=} [orT? : ClsAirport \mid orT?.waiting \neq \varnothing] \bullet orT?.totalAssignment$$

The operation *newAssign* in a class, is defined by promoting the operation *totalAssignment* of an object of the class *ClsAirport* referenced by orT.

The concept of polymorphism is briefly discussed in the following section.

2.5.3 Polymorphism

In Object-orientation, the concept of polymorphism defines a mechanism which allows a variable to be declared, whose value can be an object from any of a given collection of classes. In Object-Z, polymorphism is introduced with the unary class operator denoted by the symbol $\downarrow$, e.g., the declaration

$$orTambo : \downarrow ClsAirport$$

specifies an object of the class *ClsAirport* or any other class derived from it by inheritance.

2.5.4 Tool support for Z and Object-Z

An important advantage of using Z is the availability of tool support, allowing for the possibility to reason about the properties of the specification (Van der Poll [176]). Z tools include amongst others the following: CadiZ (Toyn and Mcdermid [175]) for formal reasoning, and Fuzz Mike Spivey's type checker for Z. The Community of Z Tools (CZT)(Malik and Utting [117]) are used for type-checking and animating Z. Unlike Z, the tools associated to Object-Z are still limited and many of them operate specifically under Linux. Examples are: the Latex macro OZ.sty (Allen [7]) for editing Z and Object-Z specifications. The Wizard (Johnston [98]) and the Object-Z version of the Community of Z Tools (CZT) (Malik and Utting [117]) for type checking. It has been proposed to encode Object-Z into existing theorem provers (e.g., Smith et al. [166]). A methodology to animate Object-Z specifications using a Z animator (McComb and Smith [119]) and for model-checking Object-Z using Abstract State Machine (ASM) (Winter and Duke [192]) have also been suggested.

2.6 Non-functional requirements specification

We have investigated the literature seeking to discover the current non-functional requirements modeling/specification approaches. A systematic literature review (SLR) process for software engineering, firstly adapted by Kitchenham [105] and later refined by Pickering and Byrne [146], was used to systematically search the literature hoping to identify existing NFRs formal specification approaches.

2.6.1 The research objectives and questions

The main objective of this literature search is to extract from the body of knowledge, existing approach(es) to formally describe non-functional requirements. The focus on formal techniques is due to the intended purpose of this work to contribute in the area of formal methods. From the above objectives, three questions are formulated:

RevQ1 - to what extent have non-functional requirements been informally specified using natural languages or semi-formal languages?

RevQ2 - are there formal specification approaches for non-functional requirements?

RevQ3 - are there existing processes to formalise natural language description or semi-formal specifications of non-functional requirements?

The two main keywords derived (from the review questions) to help build queries to retrieve relevant papers from the selected sources (in Table 2.1) are: *Non-functional requirements* and *Specification*. From these two terms, a simple search string was formed:

```
"non-functional requirement* specification"
```

The above string was adopted as the search string since the result of a pilot search with the string included publications with the various alternative terms clearly related to the research questions: `Non-functional requirements`, `NFR`, `non-functional properties`, `non-behavioral requirements`,`quality attributes / properties`, `constraints`, `goal requirements`, `Formal specification`, `Onthology`, and `architectural design`. The review method is first presented next.

2.6.2 Review approach

This review was conducted as a secondary study to assess NFRs specification approaches. As mentioned above, a systematic literature review process was used.

2.6.3 The search process

Firstly, we performed an automated search of five selected computer science and software engineering databases, and secondly, used the Google scholar database as a means to ensure the comprehensiveness of the first search. The list of databases chosen are presented in Table 2.1. The databases were chosen amongst those mainly encountered in systematic literature reviews in software engineering and information systems.

List of selected databases		
No.	Database	Web address
1-	ACM Digital Library	http://portal.acm.org/
2-	IEEE Xplorer	http://www.ieee.org/web/publications/xplore/
3-	Springer-Link	http://www.springerlink.com/
4-	ScienceDirect - Elsevier	http://www.elsevier.com
5-	Scopus	www.scopus.com

List of selected databases		
No.	Database	Web address

Table 2.1: List of selected databases

The manual search also served to validate the systematic approach since most of the journal and proceedings investigated were already included in previously searched searched databases.

Searching the databases

One of the two strings is used to search a selected database and the result is saved. Then other searches, of the same database, are performed by means of alternatives strings formed using the keywords. The results of these other searches are visually scrutinized to ensure that the relevant papers are included in the study. The filtering of search results was mainly based on the publication date and the focus area. Papers published between the year 2000 and 2015 were considered. Depending on the options available on the selected databases, research areas such as for instance, Computer Science, Software Engineering, and/or some of the above keywords were selected to refine the result.

2.6.4 The selection process

Our inclusion and exclusion criteria were derived from the above three review questions, following the approach proposed by Meline [121] and guidelines by Kitchenham [105]. The focus was on any publication in English modeling or specifying non-functional requirements. The hope was to find among the proposed models/specifications those formalising NFRs, especially non-functional requirements described with goal models. The process is summarised in Figure 2.14. Due to the diversities of methods found in the selected publications, we proceeded with their classification into five categories firstly to isolate goal-based models and also to make other techniques, as well as their diversity more apparent. The five modeling categories illustrated in Figure 2.15, with the number of papers in parentheses are: goal-based (14), UML (11), XML (5), Ontology (5) and others (14). The targeted goal-based approaches included Goal-Oriented Requirements Engineering (GORE) methods [6, 13, 32, 47, 81, 131, 132], methods integrating goals into other models[49, 113, 114, 171], and models focused or derived from softgoals (tree) interdependency graph (SIG)[53, 118, 187]. Goal-based approaches were prioritised over any other one. If an article uses, for example, UML (profile) to describe a goal-based technique which is thereafter used to specify NFRs, the article will be classified as goal-based. Articles that exploited the flexibility of UML profile to specify NFRs or those ex-

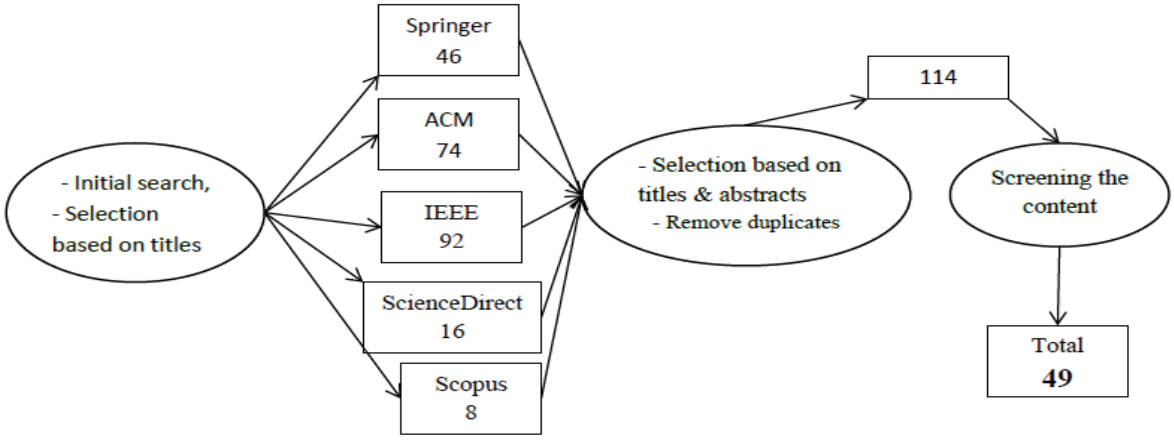

Figure 2.14: Selection of relevant publications

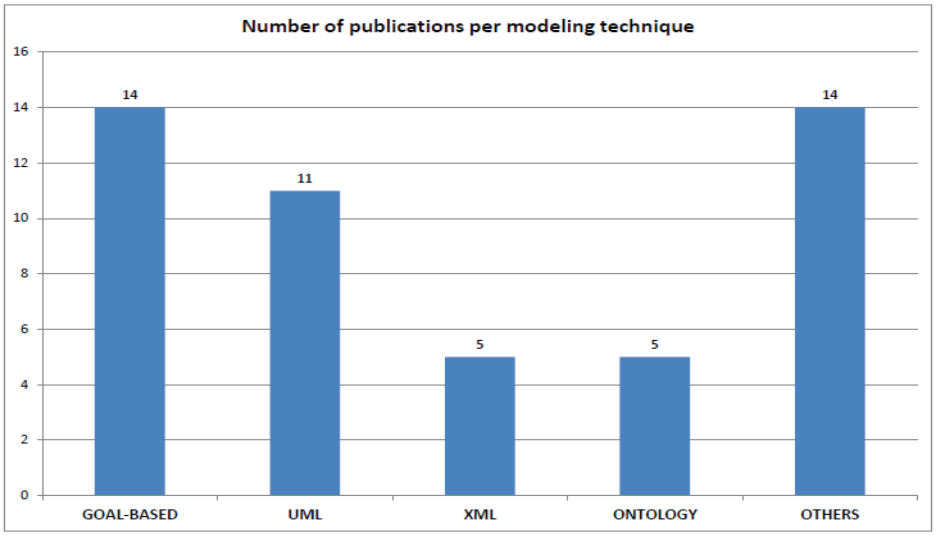

Figure 2.15: Number of publications per specification technique

tending some aspect of UML to integrate or describe NFRs where considered UML-based ([23, 48, 51, 54, 70, 103, 127, 140, 172, 184, 199]). Some papers attempted to specify NFRs directly in XML ([46, 74, 83, 148, 196]) whereas others proposed Ontology for the NFRs ([59, 84, 102, 108, 161]). The rest of the selected papers were classified as Others ([52, 58, 72, 73, 79, 86, 88, 93, 112, 115, 155, 156, 158, 197]). The list of publications per year is presented in Figure A.1, as well as the repartition of categories per year in Figure A.2.

The result of the study is discussed next.

2.6.5 Findings

From this review, it appears that apart from the few publications that propose the ontology for NFRs, the formalisation of NFRs is still limited with the main purpose to validate a specific NFRs model generally expressed in one of the categories presented above: goal-based, UML, XML or others (e.g., Cortesi and Logozzo [52], Hamid and Percebois [86]). Although a large number of publications on NFRs specification clearly indicate the need to integrate NFRs models into the functional ones (Liu et al. [113]), generic modeling approaches or frameworks to develop formal models for NFRs that may serve as inputs to further software development stages are still, in our opinion, an open research area. On this, the study failed to uncover any common NFRs formalisation approach/framework/guidelines that may be adopted to formalise goal models describing NFRs.

Next are the three types of NFRs formalisation encountered in the literature.

Direct NFRs formalisation: researchers propose mathematical expressions to describe a selected sub-set of NFRs sometime, initially represented with some sort of graph or diagrams such as UML diagrams.

NFRs ontology: the emphasis is on creating mathematical models relating NFRs to system domain or other software artifacts (Dobson et al. [59], Guizzardi et al. [84], Kassab et al. [102]). In some cases, the ontology is created for a specific goal-based model (e.g., Sancho et al. [161]).

Formalisation of GORE models: a large number of publications also pointed us to Goal-oriented requirements methods as the way forwards for NFRs analysis from NFRs framework. The development of NFRs beyond GORE methods is still largely open for research.

Research in NFRs is generally either process-oriented (Mylopoulos et al. [137]) or product-oriented. These two complementary research orientations are discussed in the following sections.

2.6.6 Product-oriented NFRs analysis

With this approach, the focus is on the software system as the product of software development and the challenge is twofold: firstly to define the qualities expected from the final product or NFRs that the product should meet; and secondarily, to evaluate the final software system to ensure that it embodies the stated qualities or meets the initial NFRs. This implies deriving means or various quantitative/qualitative models, as well as appropriate metrics to evaluate the software system when it becomes usable (e.g., Vieira et al. [182]).

Product-oriented NFRs analysis techniques are in fact complementary to the process-oriented approach introduced next.

2.6.7 Process-oriented NFRs analysis

The underlying philosophy of this approach is that the quality of process guarantees the quality of product such as based on the natural principle of "Garbage thru garbage out", to avoid having garbage thru, one should get the right process. The goal of the process-oriented analysis is to make NFRs development an integral part of software process whereby they can help to make appropriate design decisions at different stages (Lapouchnian [106]). The challenge is therefore to derive means/techniques/methods to identify, model and manage NFRs.

2.7 Chapter conclusion

This chapter has presented an overview of the URN and Z/Object-Z specification notations used in this work. A systematic literature review on NFRs specification was also conducted to investigate the extent to which formal specification techniques has so far been applied to GORE methods for the analysis of non-functional requirements represented as softgoals. The URN conceptual/metamodel, construction process and tool supports were discussed. The main concepts and notational elements used in each of the two main parts of URN to construct GRL models and UCM specifications were presented. Owing to the tidy relationship between Object-Z and its parent notation Z, both formal notations were presented to make it possible to use Z concepts and properties whenever needed in any discussion regarding Object-Z or part of it.

The study targeting the current state of the formal specification of NFRs is one of the best part of this chapter through which the researcher(s) have gained a better understanding of the practical benefits of conducting the systematic literature review.

The next chapter discusses the research design and methodology adopted in this work.

Chapter 3

Research design and methodology

Chapter 2 presented the essential aspects of the three notations used in this work: URN (UCM and GRL), Z and Object-Z. A systematic literature review was also conducted to determine the current state of NFRs specification and formalisation. This chapter focuses on the overall design approach adopted to address the research problem stated earlier in Chapter 1. Various design techniques were combined to qualitatively examine each research question in particular and thence the problem as a whole.

The approach and techniques used in this work are first presented.

3.1 Design techniques

A variety of research techniques were combined in this work to address the research problem(Hofstee [92], Johann [97], Olivier [141]). These are: literature reviews; models; synthesis of scholarship; case study; arguments based on content analysis and comparative analysis; and algorithms. The general use of each of these techniques is discussed next, while their specific integration into the research process is detailed in Section 3.2.

3.1.1 Literature review

A threefold literature study was conducted in this research work. Firstly to study in detail the URN notation, the standardised semi-formal requirement elicitation, modeling and analysis technique that includes two complementary modeling methods: UCM to address functional requirements and GRL, a GORE method, to address both functional and non-functional requirements. The main objective in studying these notation languages was to acquire an insightful understanding of the methods, the conceptual and notational elements they used to model requirements, as well as their applicability and ability to be extended

or to be coupled with other methods. Since URN is fundamentally a requirements notation language, a detail understanding of its construction process was also important to investigate the source of requirements and/or goals which are (raw) inputs to URN and therefore stimulate thinking about an approach or a model to facilitate the extraction of those requirements from their respective sources within an enterprise and/or to contribute to the system's scope definition.

Secondarily a literature study was conducted on Z and Object-Z, in relation to formal specification techniques. The primary aim was to gain an insightful understanding of the methods, concepts and notational elements used in Object-Z; which are required to anticipate thinking about possible relationships between the formal notation Z/Object-Z and semi-formal URN that constituted the starting point for the coupling of both methods.

Thirdly a systematic literature study was conducted to investigate the literature to uncover the current state of non-functional requirements specification and modeling and more precisely the formalisation of non-functional requirements embedded in goal models. Study which was required to validate the motivation of this work and hence contributing to its worthiness.

3.1.2 Framework and algorithms

As typified by Ramesh et al. [153], the "formulative" research approach, including: formulative-framework, guidelines/standards, model, process, method, algorithm, classification/taxonomy and formulative-concept, is the dominant (represented 79.15% of) research approach used in Computer Science. Although the research conducted by Ramesh et al is about two decades old, the result may not be obsolete because the need for new frameworks and algorithms to address the present and future challenges in Computer Science and Information Systems is still high as it is the case in this research work where a framework to formalise GRL models with Z/Object-Z specifications was developed. From the framework, two algorithms were derived to describe specific formalisation aspects. In the same vein, three algorithms were constructed for the synthetic model to facilitate scope definition and goal/requireements elicitation.

3.1.3 Case study

Two case studies were used in this work: the first and the smaller one, not presented in this document, was used to illustrate the (synthetic) model resulting from the work in step 1 (see fig.3.1), was presented in (Dongmo and Van der Poll [66]). The second and larger case

study, presented in Chapter 5, purposed to illustrate and validate the main ideas proposed in this work and developed either as model, framework or algorithms. The case study was thus used to illustrate the process of elicitation and modelling of user requirements from scratch, including the illustration of the applicability of the proposed synthetic enterprise model and investigating its suitability to guide the design of software systems. It was equally exploited to validate the framework proposed to generate Z/Object-Z specifications from a GRL model, as well as to evaluate the impact of using URN in the process of producing Z/Object-Z specifications, on the quality of the final specifications.

3.1.4 Models

As advocated by Olivier [141], models are essential for their simplicity, comprehensiveness, generality, exactness and clarity. A synthetic model was built to assist with system's scope definition and facilitate the identification of potential sources of goals/requirements within an enterprise. A GRL model for the case study and an Object-Z model/specification for the GRL model were constructed to illustrate the applicability of the synthetic model and to validate proposed GRL formalisation framework.

3.1.5 Synthesis of scholarship

A conceptual analysis of the exiting enterprise models were performed aiming at building a synthetic model to guide the design of software systems. A similar analysis was conducted on existing goal elicitation processes to select or build a suitable approach to construct URN models, more specifically, GRL models. Lastly, a synthetic analysis on the existing processes for transforming semi-formal software system models into formal ones, was conducted, purposing to facilitate the construction of a framework to generate a Z/Object-Z specification, from a GRL model, describing both functional and non-functional requirements.

3.1.6 Arguments based on content analysis

These were used to demonstrate different facts and findings based on two types of analyses: content analysis and comparative analysis.

- Content analysis involves the study of existing models as presented in the literature, as well as the analysis of findings through case studies.
- Comparative analysis is helpful, for example, for synthetic analysis.

The next section describes the methodology wherein the above research tools are combined to address the research problem in this thesis.

3.2 Overall research process

A four steps research strategy was adopted in this work to address the research problem. At each step, one or more design techniques presented above were carefully combined to investigate each of the five research questions discussed in Chapter 1. The solution to the research problem was derived by synthesising the solutions to the five sub-questions which are inherently complementary because each question raises issue(s) about a specific stage in requirements/goals modeling process from scope definition and elicitation, modeling and analysis to the formalisation of the constructed goal models. The process depicted in Figure

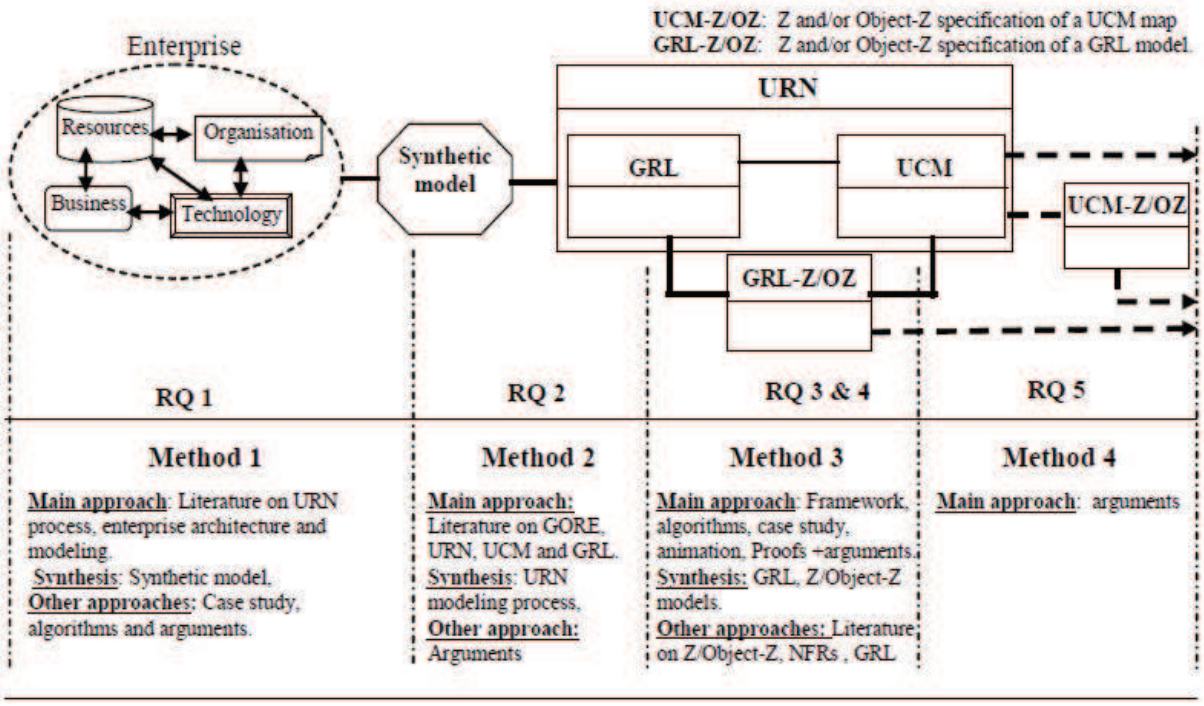

Figure 3.1: Research process

3.1 inductively or analogically (see Johann [97], pp. 176-177, for inductive model-building), suggests a comprehensive software requirements specification process, using Z/Object-Z to bridge the gap between GRL and UCM as an intermediate step, to build a more refined Z/Object-Z specification; which unlike ordinary Z/Object-Z models, integrates both functional and non-functional requirements.

The process is divided into four different steps addressing each, a specific research question for which a method was derived. Method1 purposes to investigate the first research question (RQ 1), Method 2 the second question (RQ 2), Method3, the third and fourth questions (RQ 3 and RQ 4), and Method 4, the last question (RQ 5). Each of these methods is discussed in detail next.

3.2.1 Method 1: Scope definition and requirement elicitation

This method addresses the first research question (RQ 1) with the purpose to determine/build a synthetic enterprise model to guide the construction and validation of a software system for the enterprise. The main design tool is the literature review conducted on existing enterprise models. Seconded by a theoretical analysis to determine a simpler model that included only information needed to guide the modelling, design, and validation of software systems for the enterprise, and a case study used to illustrate such a model (see Dongmo and Van der Poll [66] and Chapter 5). Finally, based on the case study, the quality and the appropriateness of the model was evaluated by means of arguments.

3.2.2 Method 2: URN construction process

This method addresses the second research question (RQ 2) related to user requirements elicitation and modelling with URN. The main design tool is the review of the literature conducted on existing GORE methods in general and particularly on URN and its two components:GRL and UCM. A synthetic analysis of the existing goal elicitation techniques was necessary to help determine or select among the existing ones an appropriate elicitation process. The process was applied to the case study (presented in Chapter 5) to generate a URN model based on the synthetic enterprise model/architecture from Method 1. Then, based on the case study, further discussions about the quality and the applicability of the proposed URN process is conducted to make room for improvement.

3.2.3 Method 3: GRL formalisation and validation of the formal specification

This method is, arguably, the most important. The method was used to investigate the main research questions of this work (RQ 3 and 4). The aim is to build a framework to transform a GRL model, describing non-functional requirements, into Object-Z, and evaluate the impact of the initial model on the resulting Object-Z specifications. The main design tool includes the formulation of a framework and algorithms guided by a literature survey conducted on Z, Object-Z, to determine and study the existing transformation schemes of semi-formal models into Z and Object-Z, and a systematic literature review on existing specification and formalisation approaches of non-functional requirements focusing on goal models including GRL models.

A synthesis of existing approaches for non-functional requirements specification/formalisation was done to inspire and/or guide the development of a framework to formalise GRL mod-

els. The framework was refined into algorithms to describe the formalisation process of the individual components of the input GRL model. The case study, animation, mathematical proof and argumentation were utilised to validate the framework and algorithms, as well as the qualities of the formal model obtained by applying the framework to the case study.

3.2.4 Method 4: Analysis of the formalisation process and the formal specification

This method purposes to investigates the impact of the formal specification of GRL models on the quality of URN models, specifically on UCMs (RQ 5). The method fully relies on arguments based on the analysis of the formalisation process and the validation of the resulting formal model, as well as an example used to illustrate the construction of a UCM map from a formal specification of the GRL model of the case study.

3.3 Chapter conclusion

This chapter has presented the fourfold research approach used in this work to investigate the research problem. At each of the four steps, a method combing two or more of the chosen design techniques is used to address one of the five sub-questions (or sub-problems) derived from the main problem. The design techniques include: literature reviews, framework and algorithms, case studies, models, synthesis of scholarship and arguments.

The following chapter focuses on the analysis of the URN construction, NFRs, as well as the formalisation of GRL models.

Chapter 4

Formalising GRL models with Z/Object-Z

In the previous chapters an overall research design including four methods adopted to address each research question was discussed. In particular, Method3 aimed to develop a framework to formalise GRL models and validate the resulting formal specification. This chapter focuses on this method. It proposes an Object-Z specification of a GRL model focusing on goals describing non-functional requirements. Before identifying GRL elements that need to be formalised, an analysis of the relationship between the GRL and UCM is first presented.

4.1 Relationship between GRL and UCM

Initially, our perception of GRL and UCM was that they were two complementary approaches used at the same abstraction level to elicit, analyse and model user requirements. However, it also appears that the UCM notation is in fact a refinement of a GRL specification. This aspect is further investigated, in the following sections, by analysing the intertwined construction process of the two techniques.

4.1.1 URN construction process

Figure 4.1 depicts the iterative construction process for URN proposed by Liu and Yu [110]. During the process, GRL and UCM are interactively and iteratively constructed.

The URN input

Initially, information such as: the *problem description*, *Business objectives*, *Use case scenarios* and any other stakeholder intention or useful information are necessarily to ignite the

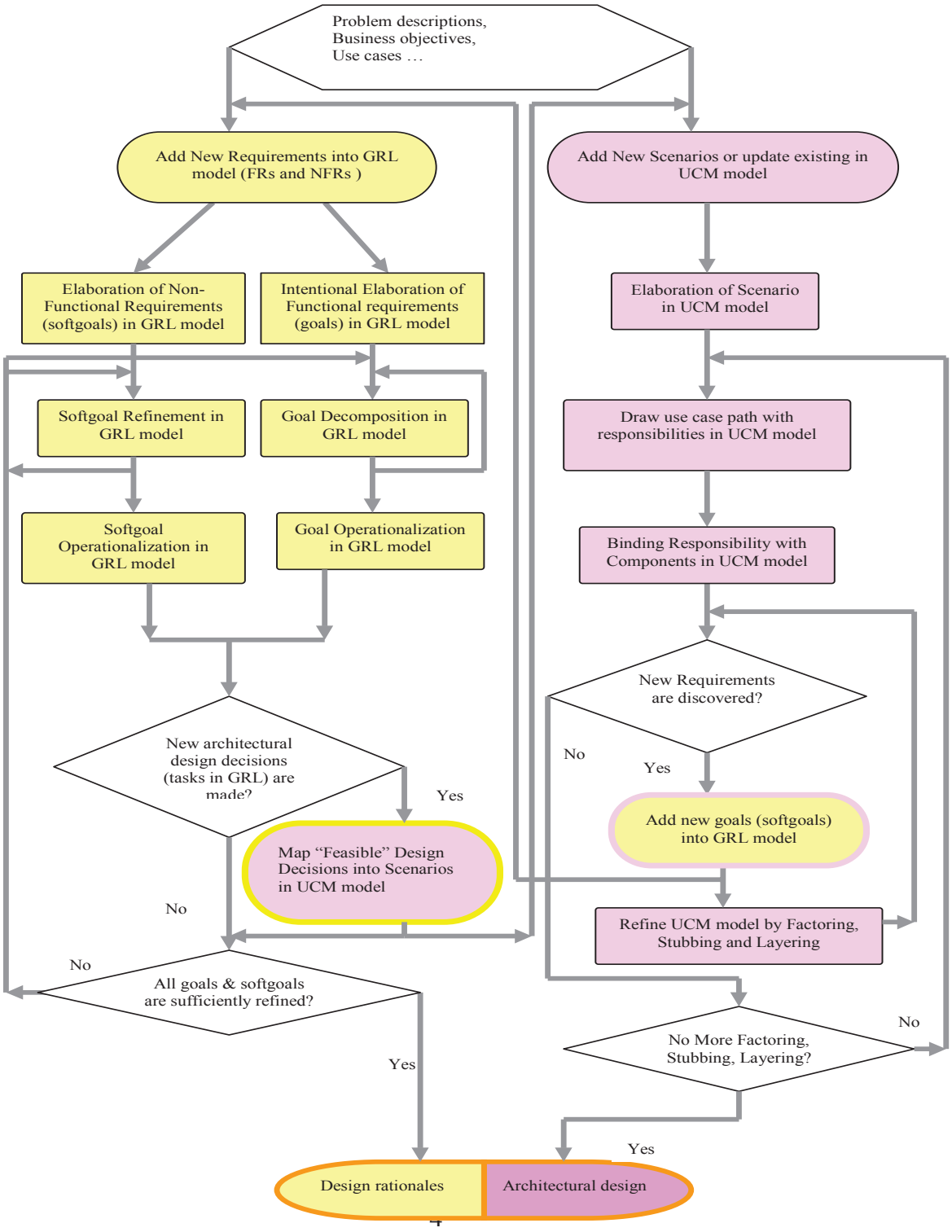

Figure 4.1: URN model construction process [110]

modelling process. However, since URN is a requirements elicitation technique, in practice, more input data are discovered during the development phase.

GRL construction

Any entity with intentions (e.g., stakeholder), also known as a role-player, is likely to be modeled as an "Actor" in GRL. GRL modeling distinguishes two complementary aspects of business objectives or stakeholders intentions: goals describing non-functional requirements (softgoals) and goals describing functional requirements, namely (hard) goals. Those two types of goals are iteratively developed separately but, interactively.

1. Developing softgoals
   Each softgoal is iteratively *refined* and *operationalized* until some concrete design decisions can be made.

   Definition 4.1.1. A softgoal refinement consists in uncovering (hard or soft) subgoals that alternatively or additionally contribute to satisfice the initial softgoal. This is traditionally known as AND/OR decomposition; which is modeled in GRL with decomposition links (URN[1]).

   Definition 4.1.2. A softgoal and/or goal operationalisation is a process that consists to discover operational means and/or resources necessary to satisfice the softgoal or to achieve the goal. Operational means to satisfice a softgoal or to achieve a goal include: Tasks, Resources, and Beliefs (URN[1]).

2. Developing (hard) goals
   As in the case of softgoals, each goal is decomposed and operationalized until some concrete design decisions are obtained.

3. Connection points between softgoals and (hard) goals
   At any point during the refinement of a softgoal, (hard) goals may be introduced causing the move from softgoal to (hard) goal analysis.

From GRL to UCM construction

From Figure 4.1, the shift to UCM construction occurs whenever new architectural design decisions are made. That is after softgoals refinement/goals decomposition and operationalisation are performed. For softgoals (especially), this includes the important phase of evaluation that establishes the degree of achievement of each participant goal based on some initial strategies [1].

UCM construction

UCM modeling is an iterative process whereby scenarios' paths (with responsibility points, connections points, etc.) are constructed and bound to components. With complex systems, the final UCM model is refined. Important phases in this process are:

1. *UCM input:* Inputs to UCM are functional requirements expressed as: scenarios, use cases, operations, etc.

2. *Drawing UCM paths and path elements:* New input scenarios or use cases generally lead to the creation of new paths or paths segment whereas, inputs like operations may lead to the update of existing paths (e.g., add new responsibilities points to a path segment).

3. *Drawing UCM components:* Components are shapes used to represent architectural artifacts such as: processes, systems and sub-systems, etc.

4. *Binding paths elements to components:* The ability to bind paths/path segments and path elements to components render UCM construction flexible so that one may, invariably, start by working on scenarios or architectural design of the system and bind them only when one is satisfied with one or another.

5. *Refining the UCM model:* Techniques such as stubbing and layering are provided in UCM to refine and organise the model until the desired level of detail is obtained.

From UCM to GRL construction

The shift to GRL construction occurs whenever new requirements are discovered during the UCM modeling (fig. 4.1). Updating GRL when a new requirement is discovered, is important to avoid modeling requirements that do not contribute to satisfy or achieve any goal or softgoal in GRL hence, introducing inconsistencies between the two models.

URN construction outputs

The URN construction is completed when two consistent models are produced: GRL and UCM models. The final GRL model is obtained when all goals and softgoals are sufficiently refined; whereas the final UCM map(s) obtained when the map(s) is (are) sufficiently refined.

4.1.2 Important observations: UCM as a GRL refinement

This section presents some observations on GRL, UCM and URN construction that may help realise that a UCM model is actually a GRL refinement. Such a causal relationship can

help to improve the internal URN process as well as the integration of URN into the existing software process. Figure 4.2 presents a summarised version of the URN construction process used for our argumentation. The observations are presented for each of the four major URN phases: the URN inputs, the GRL specification (including an initial UCM sketching), the UCM detailed specification, and the URN outputs: GRL and UCM (final models).

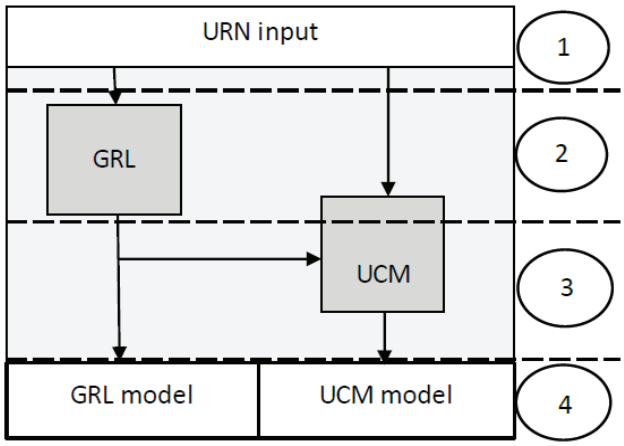

Figure 4.2: Causal relationship between GRL and UCM

Observations on URN inputs

Initially, the bulk of the URN inputs includes business objectives and stakeholders' goals that justify the system to be and constitute the main inputs to GRL. As commonly accepted among researchers and practitioners, a software product is developed and used to support important business goals that justify all other requirements of the system (see Herrmann and Paech [91]). The initial URN inputs, may equally include functional requirements in the form of scenarios, use cases, etc. which are direct inputs to UCM. However, as indicated earlier (From UCM to GRL), for this second class of inputs to be integrated into the final UCM model, the GRL specification must be updated to justify them.

Observations on GRL Construction

As shown in Figure 4.1 and in the sub-section titled "From GRL to UCM", the GRL soft-goals/goals refinement/decomposition and operationalisation produce operational elements (actors, tasks, resources, beliefs) that constitute inputs to UCM. The UCM modeling process transforms those inputs into: architectural artifacts, scenario paths or path segments including: responsibility points, path connectors, etc. and prose description wherever necessary.

Observations on UCM Construction

1. From our previous observations, justifiable inputs to UCM are sourced from GRL goals' analysis. Therefore, although an iterative construction of GRL and UCM models is possible, in practice, the complete development of the UCM model(s) or part of it depends on the partial or total construction of GRL.

2. UCM elements should therefore be traceable from the GRL model. For example, for a UCM component or actor,path or path segment, responsibility, etc. it should be possible to point out a GRL element from which it was generated as illustrated in Table 4.1.

No.	**UCM elements**	**GRL elements**
1-	Components	Actors, softgoals
2-	Paths, Path elements	intentional elements
3-	Architectural patterns	Softgoals

Table 4.1: Traceability between UCM and GRL

3. It is equally important to note that some specific grouping of UCM elements (structuring) including the binding of paths and paths elements to components ought to be justified by the need to satisfy some goals or softgoals from GRL. For example, parallelism may be introduced into a UCM map to achieve performance.

Observations on URN outputs

Although not formally proven, based on the above observations, we believe a UCM model to be a refinement of its counterpart GRL model. Assuming some of the UCM elements cannot be linked to any GRL model elements, one may rightfully ask *why would a software engineer model functionalities that do not support business goals or contribute to satisfy any stakeholder's goal?* In fact, knowing that goals/softgoals decomposition/refinement, operationalisation and evaluation are entirely performed within GRL, amongst others, the following two aspects may require more attention: firstly, verify that the operationalisation of goals, especially softgoals, performed in GRL is sufficient to reasonably achieve them. Secondary, ensure that UCM sufficiently refines GRL. These two aspects are progressively addressed in the following sections.

The next section elaborates on the design gap between UCM and GRL.

4.1.3 Conceptual gap between GRL and UCM

As discussed in Chapter 2, Section 2.1.2, p.16, the meta-class URNlink meta-model (see Figure 2.1) is an important concept that introduces the relationship between GRL and UCM and allow URN modellers to think about the traceability, refinement, composition, etc. between the elements of GRL and UCM models. To the best of the author's knowledge, the implementation of URNlink is still limited to a visual and manual connection between the model elements of GRL and UCM. As discussed in the previous section, this brings forth the importance to investigate the extent to which UCM elements can be traced back to GRL ones.

From URN construction, it appears that each UCM element is linkable to GRL operational element(s) (principally, tasks, resources, beliefs) that constitute solutions to goals/softgoals. This is justifiable since GRL operational elements representing functional requirements are directly specified in UCM; and when a new requirement is discovered in UCM, the GRL model is updated accordingly. However, it is important to notice that, in some cases, the influence of non-functional requirements may be more subtle. Few examples to illustrate:

(a) Parallelism may be introduced into a UCM to address performance, a NFR in GRL.

(b) Stubs or layers may be needed in UCM to respond to a security requirement. For example, to group together those functionalities that require a certain level of security.

(c) Some elements of a UCM may need to be re-arranged in a certain order to facilitate for example the usability of the system.

These examples show how non-functional requirements in GRL can induce actions (performed on UCM) that do not necessary describe functional requirements represented physically in the original GRL model. Additionally, UCM elements such as: forks, stubs or layers used in the above examples, are not (conceptually) different from those created when specifying functional requirements. Thus, the difficulties to trace them back to the original non-functional requirement and consequently the difficulty to validate the final UCM model relatively to GRL. One may also be attempted to question the ability of GORE methods, in general, to fully operationalise non-functional requirements since it is known that the influence of such requirements may involve any of the software development phases. Thus, the need to elaborate on the influence of non-functional requirements in software development introduced in the next section arises.

4.2 The influence of Non-Functional Requirements in software development

As discussed in Sections 2.6.6 and 2.6.7, p.45, two non-functional requirements analysis approaches are generally considered: product-oriented and process-oriented (see Mylopoulos et al. [137]). The product-oriented approach is more interested in finding appropriate metrics to evaluate the quality of the constructed software product. The process-oriented seeks means to make appropriate design decisions based on non-functional requirements.

Since each of the two approaches focuses only on one aspect of software, process-oriented on the process, and product-oriented on the final product; it is also our belief that none of the two NFRs analysis approaches is sufficient to efficiently address NFRs analysis as supported by Mylopoulos et al. [137]. Both approaches are complementary and should therefore be considered together. To describe our proposed approach to the development of non-functional requirements, we present next the concept of `Software Development System (SDS)`.

4.2.1 Software Development System (SDS)

With the term *software development system* (*SDS*), we abstract the different development phases of a software system independently of any specific method. The literature generally presents four fundamental phases for which different authors use different terms to describe (e.g., Burback [40], Otero [143], Sommerville [169]).

- *There are four fundamental phases in most, if not all, software engineering methodologies. These phases are analysis, design, implementation, and testing* (Burback [40]).
- *There are four fundamental process activities that are common to all software processes. These are: Software Specification, Development, Validation, and Evolution* (Sommerville [169]).
- *The fundamental software engineering life cycle phases include requirements, design, construction, test and maintenance* (Otero [143]).

Ideally, the SDS system needs not to be methods dependant. However, due to the diversity of terminology and development phases, to make our proposed idea more visible, the following phases are adapted:

(1) *Requirements* : this phase includes goals and requirements analysis, specification and validation.

(2) *Design* : this is mainly about software specification, modeling [and/or requirements refinement], architectural design and their validation.

(3) *Development* : this is all about implementation and deployment.

(4) *Maintenance* : this phase includes the operational system maintenance and/or evolution.

These phases are depicted in Figure 4.3; which purpose is to illustrate our approach to the integration of non-functional requirements in software development. To simplify the figure, we consider the system at the state where functional and non-functional requirements are identified. In practice, those requirements result from a preliminary work such as goals and/or requirements elicitation and analysis. The inputs to the SDS system may therefore

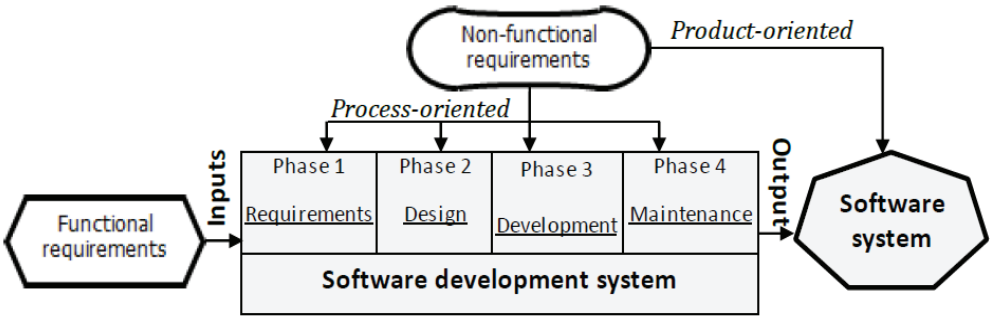

Figure 4.3: Illustrating a software development system

include initial user requirements, stakeholders' goals, functional requirements or some other intermediate model depending on various factors such as the development model, the type of software under construction, as well as the structure and synchronisation of the activities of the four SDS phases. The output is the operational software system.

Definition 4.2.1. An SDS activity refers to any task performed at an SDS phase with the purpose to progressively transform the initial SDS inputs (or an intermediary model) into an operational software. In general, high-level activities are predefined at each phase, structured and/or synchronized to form a process or a method.

Example 4.2.1. At the requirements phase (Phase 1 in Figure 4.3), predefined activities include: feasibility study, requirements elicitation and analysis, requirements specification and requirements validation (Sommerville [169]).

4.2.2 Non-Functional actions: NF-actions

GORE methods generally propose, at an early stage of software development, means to elicit, analyse, decompose/refine, model and evaluate goals, including goals describing non-

functional requirements. Goal refinement is mainly about deriving actions and resources necessary to achieve them. An evaluation of such a refined goal consists in measuring the degree of achievement of the goal based on the (estimated/calculated) quantified or qualified contribution values of derived NF-actions. Figure 4.4 illustrates a systematic process for NFRs that goes beyond the modeling, decomposition and refining of NFRs and adds steps for the formalisation followed by a formal validation (Jung and Lee [99]). Figure 4.4 also

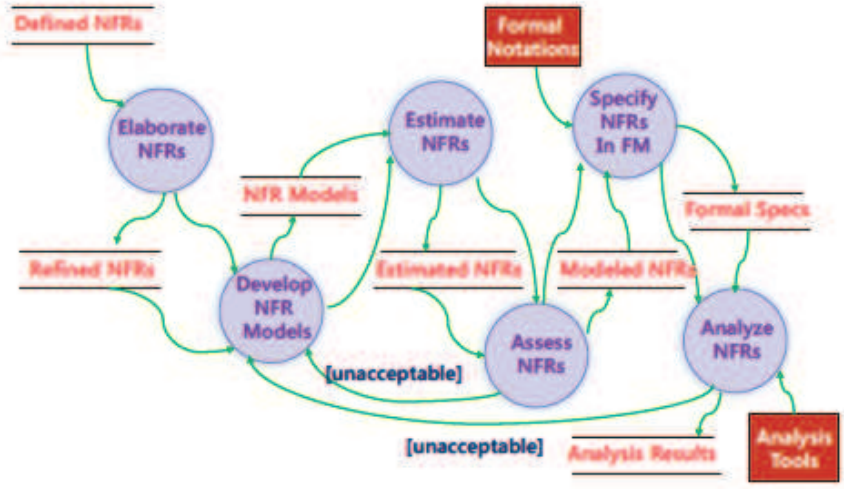

Figure 4.4: Systematic process to Non-Functional Requirements [99]

shows that goals, especially those describing NFRs, are generally developed in isolation; that is, independently of other software development process. Knowing that, beside NF-actions that refine goals, each non-functional requirement may impose at each (SDS) development phase, additional actions and constraints on SDS system's activities, tools and resources, and define metrics to evaluate the quality of an SDS's output (the operational system) expected to have desired qualities (Rosa and Cunha [157]), the analysis of the impact of NFRs on each SDS phase or activity becomes an important activity. An activity that would likely consolidate Cai's idea which emphasises that: *non-functional computing should be developed in parallel with functional computing* [43]. It is also our belief that it would be more interesting to fully integrate the NFRs development into software processes.

4.2.3 Complementary Non-Functional actions: CNF-action

The idea of CNF-actions is founded on the commonly accepted process-oriented's view that the quality of the final product depends on the quality of intermediary models. An CNF-action applies, at each SDS phase, to intermediate models or to the activities of the phase including for example, the selection and structuring of appropriate techniques/methods,

tools and activities purposing to improve the quality of the intermediate models. Thus, since objects on which CNF-actions are applied vary, depend on one another and may not be known before the development reaches a certain point, those actions may not be derived at an early phase when traditional NF-actions are analysed. Examples of such actions include:

- Creating new functional requirements
- Making choice amongst alternatives
- Imposing some constraints on the SDS activities
- Transforming a model from one state to another
- Ordering or restructuring some activities.

Some actions at one stage depend on the precedent actions. For example, the design of parallel algorithms may induce the adoption of multicore systems at the implementation phase to achieve performance.

Creating and performing actions at an SDS development phase aims to help produce intermediate models with an improved quality. For example, at the requirement phase, we may have a requirement model including additional functionalities such as: authentication and encryption. However, such model do not naturally integrate information on the origin of their improvement. Thus the need for a mechanism to propagate such information.

CNF-actions Representation/propagation

CNF-actions can be represented in two possible ways: by progressively upgrading the initial NFRs model with the new action or by creating a new model for CNF-actions.

Upgrading the initial NFRs model: CNF-actions progressively refine non-functional requirements the same way NF-actions do. They may therefore, be integrated into the initial NFRs model as further refinement.

Example 4.2.2. With GRL, a number of alternatives may be considered to upgrade the model with CNF-actions. For example, add one branch from each non-functional requirements to represent CNF-actions derived at each SDS development phase. The GRL model will thus, be progressively updated till the final software product is produced.

The problem with this approach is that, as summarised in Figure 4.4, considering the amount of work that may have been performed on the initial model before the creation of subsequent

CNF-actions, upgrading the model at some stage would cause some activities or processes to be repeated over and over when SDS phases are being developed. Thus, the need for an independent model to represent CNF-actions as alternative solution.

Creating a new model is a continuous process that follows the development of the software until its completion. This process can be perceived as an initial step to a process-oriented approach to non-functional requirement analysis. The new model can be represented in different ways; two examples are given next.

Example 4.2.3. The model representation or construction can use the same refinement method or technique as the one used to refine NFRs with NF-actions. For instance, construct a GRL model where softgoals are gradually refined with CNF-actions.

Example 4.2.4. CNF-actions may also be directly integrated into SDS intermediate models; the same modeling technique is thus used to represent the CNF-actions. For instance, an extra UML class or Object-Z class may be added to a UML model or Object-Z specification to model CNF-actions.

Considering the variety of methods, techniques and tools that ought to be used when developing a software product, the need to propagate actions carried-out from one development stage to another becomes a necessity.

CNF-actions propagation The main reason of this activity is to present to the next SDS development step or phase, a full report of the decisions made in the previous step or phase. The benefit of the report is not only for decision making but also for example, to facilitate traceability, maintain the system consistency and facilitate conflict resolution amongst NFRs.

Although different methods and techniques may be used to model CNF-actions at different levels, for a given project, a common format must be used for the report. Figure 4.5 is an example of Non-Functional actions in ProcessNFL, proposed by Rosa et al. [156]. This

```
action validateAccessAgainstEligibilityRules;
  {
     affected time_performance;
     implemented confidentiality;
     effect time_performance [-1];
  }
```

Figure 4.5: Representing NF-actions in ProcessNFL [156]

notation appropriately describes a NF-action but, may need to be adapted to include all the information required for example, information on SDS development parts affected by CNF-actions. Thus, the need to first identify all the elements necessary for CNF-action's report. The following information must be included: The *SDS phase*, the *NFR*, the *CNF-action*, and the *object(s)* affected by the action. The template should therefore be:

$$< Phase, NFR, CNF - action, Domain, [optional] >$$

Based on the fact that the class of XML languages are already intensively used in software engineering (e.g., Fei and Xiaodong [74], Poon et al. [148]), one may also consider these languages for propagating CNF-actions across different modeling techniques and tools. The fact that URN itself employs XML to share or export models to other techniques is supportive. Since CNF-actions are inherently performed on known objects or entities, the next important aspect to discuss when analysing the influence of a NFR on a system is the CNF-action's domain.

4.2.4 A non-functional requirement's domain

The concept of domain is very generic and may have different meaning in different contexts. For instance, in mathematics we have function domain, domain name in computer network and application domain for software.

Example 4.2.5. In Operating System design, the "domain" of protection is composed of hardware and software objects (see Galvin et al. [77], Ch. 14, pp.613-618).

Since protection is an aspect of the security requirement, which is indeed a non-functional requirement, this example is closed to the context in which the term domain is used in this work.

Definition 4.2.2. A non-functional requirement domain includes any sub set of the SDS system that may be influenced by the non-functional requirement. In other words, any element of the SDS system on which an NFR-action may be performed to satisfice the NFR is included in the NFR domain.

To construct a NFR domain, the followings should therefore be considered: *SDS development activities*, *SDS development methods and techniques*, *SDS development tools* at that phase, as well as, *resources needed*, *the input or initial requirements* or *intermediary models from the previous phase*, and *development or system environment*.

- *SDS development activities:* a non-functional requirement may have twofold influences on the SDS activities. Firstly, constraint the selection of activities amongst alternatives and Secondly, constraint an activity to be performed in a certain way or include some specific functionalities.

- *SDS development methods and techniques:* Similarly to SDS activities, a NFR may force the use of specific methods/techniques and place some constraints on how those techniques will be used.

- *Development tools* may be influenced by a non-functional requirements. For example, a highly secured system may not be developed with unsecured tools. If the portability requirement is to be reinforced, an implementation tool like Java will be appropriate.

- *Resources needed:* to reinforce a NFR, some specific type of resources may be preferred. For example, for security purpose, a wired network may be preferred to a wireless one.

- *Functional requirements* and/or *Intermediate models* from the previous phases are likely to be influenced by NFRs and are therefore part of NFRs domains.

- *Development (and/or the System) environment* includes, for example, operating systems, other software, specialized inputs/output devices such as sensors. Non-functional requirements may constrain the choice of operating systems and the hardware.

Due to the variety of entities that may be part of a domain as listed above, for the sake of commodity and to facilitate operations on domains, we use the concept of "object" to abstract each domain element. Specific attributes and properties of an individual element are fully exploited when the element is considered in isolation.

4.3 An Object-Z specification of URN model elements: Focusing on NFRs in GRL

UCM provides for concepts to efficiently refine and model operational requirements that satisfy goals defined in GRL. Softgoals describing NFRs in a GRL model define requirements that may constrain software architectural design in UCM, especially, the creation and structuring of UCM components. A mechanism was proposed by Dongmo and van der Poll [64] to formalise UCM diagrams with Z and Object-Z. Since part of a GRL model, describing Goals, Tasks, Resources and Beliefs, can be directly refined into UCM, this work focuses on the formalisation of softgoals, their influence on the construction of UCM and their propagation.

4.3.1 GRL Model elements: list of sub-classes

Table 4.2 presents the list of subclasses of the superclass of GRL model elements. The table indicates subclasses that are to be formalised and those that will not.

No.	GRL elements	Type	Included?
1-	StrategiesGroup	Class	No
2-	EvaluationStrategy	Class	No
3-	ContributionContext	Class	No
4-	ContributionContextGroup	Class	No
5-	IndicatorConversion	Class	No
6-	GRLLinkableElement	Meta-class	Yes
7-	ElementLink	Meta-class	Yes

Table 4.2: GRL Model Elements subclasses

The first five classes define GRL elements used to evaluate the level of satisfaction of decomposed/refined goals. Since goals evaluation is part of GRL construction process, these classes are not considered for the formalisation. This work is concerned with the already constructed GRL model or model elements. Other aspects of the model, such as the evaluation, may be considered for formalisation later to bring more details into the construction process. GRLLinkableElement and ElementLink are therefore, the two meta-classes that will be further explored. GRLLinkableElement is an abstraction of the class of actors and intentional elements; whereas ElementLink objects are used to relate GRLLinkableElement from one to another. Table 4.3 recapitulates those Linkable elements that are considered for formalisation.

No.	GRL Linkable elements	Included?
1-	Actor	Yes
2-	Intentional elements	
-	Belief	Yes
-	Resource	Yes
-	Task	Yes
-	Goal	Yes
-	Softgoal	Yes

Table 4.3: List of GRLLinkableElement subclasses

The class **ElementLink** is composed of: Depency, Decomposition, and Contribution shown in Table 4.4. All the objects of theses classes are to be formalised but, not necessary to be

refined.

No.	Element Links	Included?
1-	Dependency	Yes
2-	Decomposition	Yes
-	AND	
-	XOR	
-	IOR	
3-	Contribution	Yes
-	Make	
-	Help	
-	SomePositive	
-	Unknown	
-	SomeNegative	
-	Hurt	
-	Break	

Table 4.4: List of element links

4.3.2 Basic formalisation approach: a top-down approach

GRL, UCM, and URNLink model elements are subclasses of the URN model element class [1]. They therefore, inherit the attributes and characteristics of their superclass URNModelElement. Two important inherited attributes are: unique identifier (id) of the model element and its name. Another inherited characteristic is the named value called Metadata. As shown in Figure 4.6, GRL elements are hierarchically constructed from the superclass GRLModelElement as subclasses that inherits the properties of the superclass. Thus, the choice of a top-down approach as the main formalisation strategy: superclasses are first formalised followed by subclasses. This is important as this work tries to keep the structure of the original GRL model as much as possible.

The identification and analysis of GRL model elements are based on the GRL metamodel, a UML stereotype, from the standardised URN reference, namely, ITU-T Z.151 [1] and presented in Figure 4.6. The three main components of a GRL specification to be specified include: Actor, GRLContainableElement, and ElementLink. Other components, especially those related to the evaluation, are outside the scope of this study.

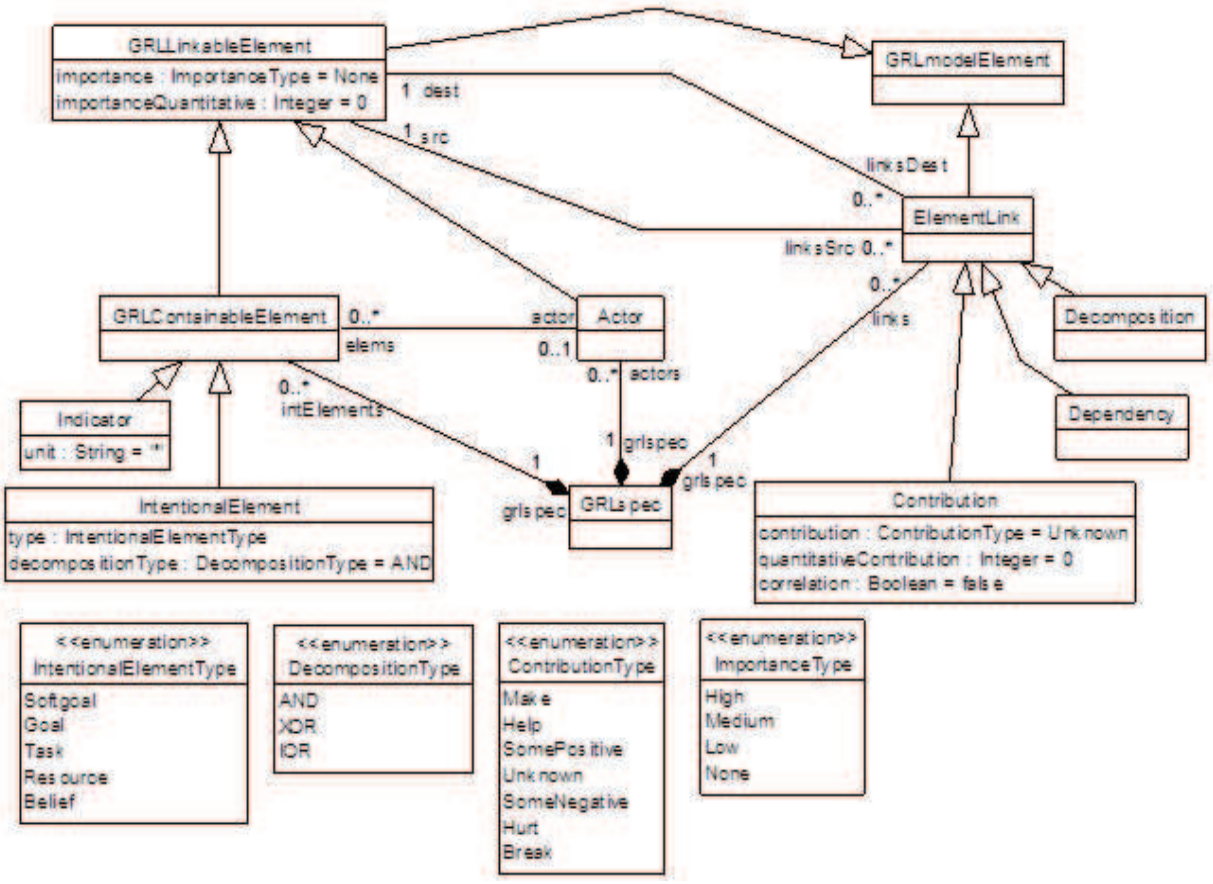

Figure 4.6: GRL elements to be formalised [1]

4.3.3 Formalising GRLModelElement metaclass

A GRLModelElement is an abstract class and operations on this class are performed trough its subclasses.

ClsMetadata

$\lceil(name, value)$
$[Name]$

$name : Name$
$value : String$

ClsGRLModelElement

$\lceil(id, name, mdata)$
$[Identifier]$

$id : Identifier$
$name : String$
$mdata : \mathbb{F}\ ClsMetaData$

INIT

$id = \bot \land \#mdata = 0$

A GRL model element is characterised by: a unique identifier (id), a name, and some metadata. Each subclass of such a class inherits those three properties.

4.3.4 The Object-Z specification of the subclasses of GRLModelElement: LinkableElement and ElementLink

ClsLinkableElement

$\lceil(id, name, mdata, importance, impQualitative)$
$ImportanceType == High \mid Medium \mid Low \mid None$
$ClsGRLModelElement$

$importance : ImportanceType$
$impQualitative : \mathbb{Z}$

The two sub classes of the GRLLinkableElement metaclass are: Actors and GRLContainableElement classes.

4.3.5 The specification of the sub classes of GRLLinkableElements

The Object-Z specification of Actors

An actor represents an entity that has intentions and that carries out actions to achieve its goals.

ClsActor

$\lceil(id, name, mdata, importance, impQualitative)$
ClsLinkableElement

The Object-Z specification of GRLContainableElement

A GRLContainableElement is a GRL model element that can be contained in an actor definition. It is an abstraction of the commonality of intentional elements and indicators.

ClsGRLContainableElt

$\lceil(id, name, mdata, importance, impQualitative, actor)$
ClsLinkableElement

$actor : ClsActor$

INIT

$actor.id = \bot$

Since indicators are not included in the scope of this study, GRLContainableElement and IntentionalElement are the same. There are five types of intentional elements: Belief, Resource, Task, Goal and Softgoal. The first four can be directly refined into functional requirement model with UCM; Whereas softgoal may not since they describe non-functional requirements.

4.3.6 Formalising the subclasses of ContainableElements: IntentionalElement and Indicators

ClsIntentionalElement

$\lceil(id, name, mdata, importance, impQualitative, actor)$

ClsContainableElement

$(actor \neq \bot \land \forall\, elt : \downarrow ClsIntentionalElement \mid elt.actor.id = actor.id$
$\lor$
$actor = \bot \land \forall\, elt : \downarrow ClsIntentionalElement \mid elt.actor = \bot) \bullet$
$elt.name \neq name$

As indicated in Table 4.3, there are five types of intentional elements: Belief, Resource, Task, Goal and Softgoal. We specify these model elements as subclasses of the abstract class ClsIntentionalElement. Some of the reasons:

- Permits to consider the particularities of each element
- Facilitate the modeling of elements decomposition/refinement with elementlink; making it possible to model the particular type of elements used to refine other elements.
- Improve on the traceability of GRL elements from corresponding UCM by means of URL links.
- Facilitate further analysis and development of a specific type of Intentional element such as softgoals without necessary having to address other types.
- Facilitate the maintenance of intentional elements as a whole since it makes it possible for new types to be easily added to or removed from the model.
- Having an id, makes it easier to link decisions or actions based on a softgoal or non-functional requirement to that particular intentional element.

The Object-Z specification of the intentional element Belief

As a domain characteristic, a belief is modeled in GRL as a text description useful in decision-making process. Considering their importance (for example, facilitating later review, justification and change of the system, and enhancing traceability), and the fact that beliefs are not explicitly integrated into UCMs, they may be added to a Z/Object-Z specification as a given type. However, it is preferred to model it as a class schema.

ClsBelief

ClsIntentionalElement

$description : String$

The Object-Z specification of Resource, Task and (hard) Goals

- **Resource**

  As mentioned above, a resource is modeled as an Object-Z class. One particularity of this intentional element is the availability.

ClsResource

$\upharpoonright(id, name, mdata, importance, impQualitative, actor, available)$

ClsIntentionalElement

$available : Integer$

$available \geq 0$

INIT

$available = 0$

- **Task**

  A task is an abstraction of any action that may be taken to satisfy a goal, softgoal or a high-level task. A task models one or more system functionalities, including data structure that may therefore be specified in a UCM model, which Z and Object-Z formalisation may be derived [64].

ClsTask

$\upharpoonright(id, name, mdata, importance, impQualitative, actor)$

ClsIntentionalElement

  It would be equally important to keep track of the relationship with the goal, softgoal or the higher-level task that it satisfies; this aspect will be further developed when analysing the metaclass ElementLink.

- **Goal**

  Goals (also called hard goals or functional goals) model the system (or business) objectives that can be (functionally) satisfied. In a GRL model, a goal may be decomposed into tasks, (sub) goals, and/or resources that are needed for its achievement. The only intentional element that cannot refine a hard goal is a softgoal.

ClsGoal

$\lceil(id, name, mdata, importance, impQualitative, actor)$
ClsIntentionalElement

Object-Z specification of Softgoals

A softgoal describes a quality or a non-functional requirement. In this work we indifferently called both non-functional requirements. The formalisation of a softgoal will consider three aspects: the softgoal itself as an intentional object, its refinement using GRL links, and its CNF-actions (complementary non-functional actions) at the current stage. The template for each CNFR-action is:

$$< Phase, NFR, CNF - action, Domain, [optional] >$$

where:

- *Phase* refers to the SDS development phase during which the action is performed. In this case, it is the Requirement phase and precisely the UCM construction part of URN construction: Requirement:URN:UCM

- *NFR* refers to the non-functional requirement for which the action is being defined.

- *CNF-action* is the complementary NF-action under consideration.

- *Domain* describes the set of objects on which the action is performed.

- *Optional* is an optional piece of information that may be added to bring more clarity.

A CNF-action as described above may be formalised in different ways. However, since it refines a non-functional requirement just as a task or a (hard) goal does, preference is given to Object-Z class structure to represent it. In the same vein, each of the four components of an CFN-action is defined as an (empty) class in Object-Z. Basic type can also be used however the class concept provides for a better concept for further development of the components.

Example 4.3.1. The class of Phases is defined as:

clsPhase

Other components can be defined similarly. The class of complementary non-functional actions is therefore defined as:

ClsComplementaryAction

$\lceil(id, name, mdata, phase, action, domain, option)$
ClsIntentionalElement

$phase : ClsPhase$
$action : ClsAction$
$domain : ClsDomain$
$option : Description$

The class of CNF-actions, *ClsComplementaryAction*, is defined as a subclass of the class of intentional elements. This makes the association of CNF-actions with other GRL elements, especially softgoals, more flexible because it makes it a linkable element and hence, allowing the designer to link them, if desired, to other linkable elements using element links.

The class of Softgoals is therefore defined as:

ClsSoftGoal

$\lceil(id, name, mdata, importance, impQualitative, actor, cnfactions)$
ClsIntentionalElement

$cnfactions : \mathbb{P}\ ClsComplementaryAction$

To keep the model simpler, the component *cnfactions* is added to capture the set of CNF-actions associated to the softgoal. The advantage of such a model is the facility to associate complementary non-functional action with non-functional requirements. For example, it may be easier to capture those information in a database or knowledge system. However, using GRL links to connect CNF-actions to softgoals would permit continuous update of the GRL model during the complete development process with the benefit of applying existing GRL analysis techniques to CNF-actions as another type of intentional element.

4.3.7 Formalising GRL Links

ElementLink is an important concept used to capture relationships between two GRL linkable elements: one linkable element is known as link source and the other as link destination. Three types of links are put forward: Dependency, Decomposition, and Contribution.The GRL ElementLink model is a subclass of GRLModelElement and thus, inherits properties from GRLModelElement.

ClsElementLink

$\lceil(id, name, mdata, linksrc, linkdest)$
ClsGRLModelElement

$linksrc : \downarrow ClsLinkableElement$
$linkdest : \downarrow ClsLinkableElement$

$linksrc.id \neq linkdest.id$

An instance of the class ClsElementLink includes the two components *linksrc* and *linkdest* representing,respectively, the source and destination of the link.

Dependency

A dependency link describes how a source actor definition (the depender) depends on a destination actor definition (the dependee) for an intentional element or indicator (the dependum).

ClsDependency

$\lceil(id, name, mdata, depender, dependee, dependum, why, how)$
ClsElementLink
$depender \mathrel{\widehat{=}} linksrc$
$dependee \mathrel{\widehat{=}} linkdest$

$dependum : \downarrow ClsIntentionalElement$
$why, how : \downarrow ClsIntentionalElement$

$\{depender, dependee\} \cap ClsBelief = \varnothing$
$depender \notin ClsActor \land dependee \notin ClsActor \Rightarrow$
$\quad depender.actor \neq \bot \lor dependee \neq \bot$
$\quad depender.actor.id \neq dependee.actor.id$

INIT

$why = \bot \land \ how = \bot$

Decomposition:

"The process of subdividing a set of goals into logical subgroups so that system requirements can be more easily understood, defined and specified" Anton [20]. In GRL, it relates the source intentional elements that need to be satisfied or available to a target intentional element to be satisfied. Three decomposition types are used: AND, XOR and IOR.

ClsDecomposition

$\lceil(id, name, mdata, linksrc, linkdest, type)$
$ClsElementLink$
$DecompositionType == AND \mid IOR \mid XOR$

$type : DecompositionType$

$\{linksrc, linkdest\} \cap ClsBelief = \varnothing$
$linksrc \notin ClsActor \land linkdest \notin ClsActor$

INIT

$type = AND$

Contribution

A contribution link defines the level of impact that the satisfaction of a source intentional element or indicator has on the satisfaction of a destination intentional element.

ClsContribution

$\lceil(id, name, mdata, linksrc, linkdest, contribtype, contributionqty, correlation)$
$ClsElementLink$
$ContributionType == Make \mid Help \mid SomePositive \mid Unknown$
$\mid SomeNegative \mid Break \mid Hurt$

$contribtype : ContributionType$
$contributionqty : \mathbb{Z}$
$corrolation : \mathbb{B}$

$\{linksrc, linkdest\} \cap ClsBelief = \varnothing$
$linkdest \notin ClsResource \cup ClsBelief$
$linkdest \notin ClsIndicator$
$linksrc \in ClsIndicator \Rightarrow correlation = \text{false}$
$-100 \leq contributionqty \leq 100$

4.4 Framework for a formal specification of an input GRL model

Two approaches may be considered: interactive transformation and a direct transformation.

Interactive approach: a formal specification is progressively generated when the input GRL model is constructed.

Direct transformation approach: the input is a fully developed GRL model, which is to be formalised. It is the author's belief that if this approach is well-designed/modularised, it can be reused in the design of an interactive approach with little modifications. Thus, the focus in this work on the direct transformation approach. This keeps the formal specification of the GRL model in line with the NFRs systematic process illustrated in Figure 4.4. To perform a direct formal specification of an input GRL model, two important tasks are to be defined: the GRL model traversal mechanism (how are the model elements identified?) and the formalisation process (how are identified elements formalised?).

The GRL model in Figure 4.7 is used in the rest of this chapter to illustrate our formalisation approach. To improve the visibility of their organisation, the management decides to use

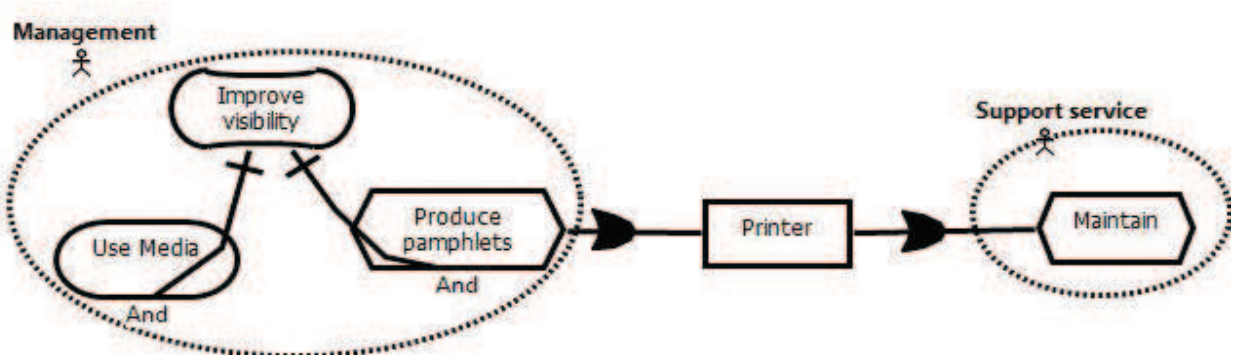

Figure 4.7: Illustrative GRL model

public media and produce pamphlets. To produce the pamphlets, the management depends on the support service to fix their printers.

4.4.1 GRL model traversal mechanism

Figure 4.6 shows that a GRL specification is composed of three sets of elements: a set of containable elements, a set of actors, and a set of element links. The traversal approach adopted in this work is based on the idea to first identify those elements that are needed by other elements. For example, source and destination elements are needed to construct a link. However, the order in which those elements are formalised must also be considered for a proper scanning.

Traverse(Actors, grlspec): this module would scan the input GRL specification (grlspec) to identify all the actors definitions within the specification.

Example 4.4.1. With the GRL model in Figure 4.7, the output would be:
$Actors = \{Management, Supportservice\}$.

Traverse(FreeElements, ActorElements, grlspec) : the module scans the input GRL specification to identify all the intentional elements and/or indicators. The variable **FreeElements** contains intentional elements that are not contained in any actor's definition and **ActorElements** includes those contained in actors definition.

Example 4.4.2. With the GRL model in Figure 4.7, the output would be:
$FreeElements = \{Printer\}$
$ManagementElements = \{Improve\ visibility, Use\ media, Produce\ pamphlets\}$
$SupportServiceElements = \{Maintain\}$

Traverse(Links, grlspec) : the module identifies all the link elements within the input GRL specification.

Example 4.4.3. With the GRL model in Figure 4.7, the output would be:
$Decomposition = \{And(Improve\ visibility, Use\ media), And(Improve\ visibility, Produce\ pamphlets)\}$
$Dependency = \{(Produce\ pamphlets, Printer, Maintain)\}$
$Contribution/Correlation = \{\}$

The formalisation process maybe integrated directly into these traversal modules however, it is preferable to separate the two processes. The scanning of an input GRL model may, for instance, act as a parser to ensure that the model is well-formed and extract the desired information on target elements and keep in output variables: **Actors**, **FreeElements**, **ActorElements** and **Links**. Depending on the physical structure of grlspec, these traversal modules may require simple or complex algorithms to be implemented.

4.4.2 Formalisation approach

Two approaches are used to transform each GRL element into Object-Z:

Instantiation

Goals and softgoals are GRL elements that are (presumably) fully decomposed/refined and analysed within the GORE method that created them. Hence, they are generally not considered for further development in software process; instead, the functional requirements,

including tasks and resources, that operationalise them are. For this reason, it is proposed that for each goal or softgoal in a GRL specification, an instance of the Object-Z abstract class for the softgoals or goals be created to represent the concrete element. In the same vein, the Object-Z specification of all link elements, as well as beliefs, are obtained by instantiating their Object-Z abstract class.

Example 4.4.4. Consider the softgoal *Improve visibility* and the goal *Use media* from the GRL model in Figure 4.7. The abstract class for softgoals and goals, also called in this work templates, developed earlier in Section 4 are, respectively, *ClsSoftgoal* and *ClsGoal*. The Object-Z specification of the two intentional elements are therefore:

$$improvevisib : ClsSoftgoal$$
$$usemedia : ClsGoal$$

The next formalisation approach is based on the concept of inheritance.

Inheritance

In practice, functional requirements resulting from goals and softgoals analysis are represented in GRL as task, resources, etc. These elements are candidates for further development in forthcoming software development phases. For each such element, it is proposed in this work to create a new Object-Z class that inherits the Object-Z abstract (super) class of the type of GRL model element under-consideration.

Example 4.4.5. The actor definition *management* and the resource *Printer* in Figure 4.7, are now used to illustrate this approach. The abstract classes for these two elements are, respectively, *ClsActor* and *ClsResource*. As an actor definition, *management* may have particular properties that other actors do not encompass. More importantly, actors definitions are generally analysed and developed in subsequent development phases. For example, in UCM an actor definition is refined into a UCM component. Thus, the need to keep its specification abstract.

ClsManagement

$$\lceil (department, nfdependers, \ldots)$$
$$ClsActor$$

$$department : String$$
$$nfdependers : \mathbb{P}\ ClsSoftgoal$$

$$\forall\ nfelt \in nfdependers \bullet nfelt.actor.id \neq id$$

In the same vein, a printer as a specific resource may encompass properties that other resources do not have.

ClsPrinter

$\lceil(type, model, nfobjectives, \ldots)$
ClsResource

$type : String$
$model : String$
$nfobjectives : \mathbb{P}\ ClsSoftgoal$

$[Constraints\ on\ properties]$

4.4.3 Creating an Object-Z specification for an input GRL model

For an input GRL specification, the proposed formalisation process is the following:

Elementary transformation/templates: propose an elementary transformation of each GRL model element that can be used as template to formalise concrete elements. During this phase, the order in which the different types of elements are to be formalised is also determined. For example if the template for a containable element inherits properties from an actor, then the formalisation of containable elements should follow that of actors.
Object-Z classes proposed in (Section 4.3.6, P. 74) are here used as templates for Object-Z specification of a GRL model.

Elements identification: Apply the traversal algorithms to the input GRL model to determine all the elements of the specification and group them according to their types.

GRL specification class: create an Object-Z class, namely *ClsGRLSpec*, to keep all elements that are not contained in an actor definition and all the links. Major operations on actors' classes, element links and other specification components should also be defined in this class. Otherwise, it should be possible to access and trigger operations defined in different components of the specification from this class.

ClsGrlSpec

$(free\_elts, actors\_elts, alllinks)$

$free\_elts : \mathbb{P} \downarrow ClsIntentionalElement$
$actors\_elts : \mathbb{P} \downarrow ClsIntentionalElement$
$alllinks : \mathbb{P} \downarrow ClsElementLink$

$\forall\, elt \in free\_elts \bullet elt.actor = \perp$

INIT

$free\_elts = \varnothing \ \wedge \ actors\_elts = \varnothing \wedge \ alllinks = \varnothing$

Transformation of each element: Table 4.5 presents for each type of input element, the type of transformation to be performed on the Object-Z class template.

No.	Element (E)	OZ Template	Formalisation
1	actor	*ClsActors*	create a subclass
2	Sofgoal	ClsSofGoal	instantiate
3	Goal	ClsGoal	instantiate
4	Task	ClsTask	create a subclass
5	Resource	ClsResource	create a subclass
6	Belief	ClsBelief	instantiate
7	Dependency	ClsDependency	instantiate
8	Contribution	ClsContribution	instantiate
9	Decomposition	ClsDecomposition	instantiate

Table 4.5: Summary of transformation per element type

Formalising actors: includes in each new actor class, the component *nfdependers* to keep the set of all softgoals whose achievement depends on the actor. These softgoals will be used to guide the development of the actor in subsequent phases and hence, stimulate the creation of complementary non-functional actions.

Formalising Intentional elements: Five types of intentional elements are considered:

1- ***Belief, goal and softgoal:*** Create, through the class *ClsGRLSpec*, an instance of the element's template and:

- Add to the component named *free_elts* if the element does not belong to any actor.
- Otherwise, add to the component *actors_elts*.

2- ***Resource and task*** : create a new Object-Z class as a subclass of the element's template. The new class must include properties/attributes specific to the input elements, as well as the compulsory component *nfobjectives* to contain all softgoals that need this element to be achieved.

   The softgoals in *nfobjectives* aim to guide the development of the resource or task in subsequent phases and therefore, stimulate the creation of complementary non-functional actions.

Formalising element links: For each element link, create, within the class *ClsGRLSpec*, an instance of the element's template and add the newly created object to the component *alllinks*. As indicated in Table 4.5, three types of links are considered:

1- ***Dependency***: each dependency link is an instance of the class *ClsDependency*.

2- ***Decomposition***: each decomposition link is an instance of the class *ClsDecomposition*. The type of the decomposition (AND, IOR, XOR) is indicated in the variable *type*.

3- ***Contribution/correlation***: a contribution or a correlation is an instance of the class *ClsContribution*. The boolean variable *correlation* is set to true for correlations. The type of the contribution is indicated in the variable *contribtype*.

The framework so far developed is summarised with Algorthm 1.

4.4.4 Updating the specification in the light of element links

An element link connects two linkable elements say A and B. The Object-Z specification of A or B results in either a class schema or an instance of a class. The purpose of this phase is to analyse the impact of such a link on the respective formal specifications of A and B focusing particularly on propagating information on softgoals to the specifications. Each of the three types of element links is considered separately.

The influence of a dependency link

A dependency relationship may involved five entities: (depender, why), dependum, (dependee, how). The depender and dependee are actors definition. The why is an intentional element (or indicator) in the depender that needs the dependum (an intentional element) for its achievement. The intentional element (how) in the dependee is the means by which the dependee provides the dependum to the depender.

The influence of this link element is considered when the "why" of the depender is a softgoal and the Object-Z specification of the dependum or the "how" of the dependee is a class schema. In Object-Z, such a class schema provides for means to achieve the softgoal. Hence, the necessity to include a reference to the Object-Z's object specifying the softgoal in it. The class schema describing the "how" is considered only in the absence of the dependum.

If the dependum (or the "how") is either a softgoal or a goal, then each branch of the refinement (decomposition) tree of the (soft)goal should be followed downward until the first intentional element(s) for which the Object-Z specification is a class schema is reached. The reference to the object specifying the "why" of the depender is added to that/those class schema(s).

If there is neither a dependum nor a "how" then the object's reference should be added to the dependee's class schema.

The presence of the reference in the class schema is to ensure that further development of the class will continue to consider the softgoal that ought to be achieved through for example Complementary Non-Functional actions (CNF-actions).

The influence of a decomposition link

Consider a softgoal A the source of a decomposition link, and B the destination element. If the Object-Z specification of B is a class schema, then add the reference of the object

Algorithm 1: Summary of the transformation process

input : GRLSpec

output: RGE:References to Grl Model Elements, OZS:Object-Z specification

begin

{Create Object-Z templates for each GRL model element}

(see chap. 4, sec.4.3.3, pp.70 - 77)

1 **foreach** *GRL model element* **do**

  if *Object-Z template does not exist* **then**

    Create an Object-Z template for the GRL model element

{apply any GRL traversal algorithm to identify all its elements}

(see chap. 4, sec.4.4.1, pp.78 - 79)

2 **Traverse(GRLSpec)**

  Traverse(Actors,GRLSpec)

  Traverse(FreeElements, ActorElements,GRLSpec)

  Traverse(Links,GRLSpec)

{Create Object-Z class schemas for Actors, Tasks and Resources}

(see chap. 4, sec.4.4.3, pp. 81 - 88)

3 **Formalise the identified actors**

  foreach *actor* **do**

    Create a new class schema that inherits actors' template

    Create components and operations that are needed

4 **Formalise the identified tasks**

  foreach *task* **do**

    Create a new class schema that inherits task' template

    Create components and operations that are required

5 **Formalise the identified resources**

  foreach *resource* **do**

    Create a new class schema that inherits resources' template

    create components and operations that are required

6 **Formalise the system class: GRLSpec**

  (see chap. 4, sec.4.4.3, pp. 81 - 88)

7 **Update the specification in the light of link elements**

  (see chap. 4, sec.4.4.4, pp. 84 - 88)

8 **Finalise the specification**

  (see chap. 4, sec.4.4.4, p.88)

describing A to that class. Otherwise, follow each refinement branch from B until the first task or resource is reached on each branch and add the reference of the object specifying A to the class of that task or resource.

The influence of a contribution link

The analysis of the influence of the contribution link on the formal specification of the source and destination elements involved is very similar to that of the decomposition link. The basic idea is that if a task or a resource contributes to the achievement of a softgoal (non-functional requirement), the formal specification of that task or resource must keep track of the object describing the softgoal. As stated before, the presence of the object's reference in the specification of the task or resource is to serve, for example, as a constrain, a control entity or a guide that stimulates the creation of Complementary Non-Functional actions (CNF-actions) in subsequent development phases of the specification.

Algorithm 2 summarises the influence of link elements. The expression "refinement branch" used in this document is defined as follows.

Definition 4.4.1. A refinement branch from an intentional element say A, is a set of (consecutive) links, composed essentially of decomposition and contribution links, that progressively decompose/refine A in a GRL model.

Algorithm 2: Update the specification in the light of link elements

input : List of links, OZ specification

output: OZ specification updated

begin

This algorithm summarises the framework in (Chapter 4, Section 4.4.4, pp.84 - 88)

List of links = partitions(*Depencies*, *Decompositions*, *Contributions*)

1 **Analyse each dependency link in** *Dependencies*

- Consider the elements of the link: (depender, Why), What, (dependee, How)
- **if** *Why is not a softgoal* **then** Consider first softgoal(s) that refine/decompose to Why
- **let** ozWhy be the Object-Z spec of Why or softgoal that refine/decompose to Why
- **if** *What is given* **then** target := What
- **else if** *How is given* **then** target:= How
- **case** *the Object-Z specification of target is a class schema* **do**
  - Add refWhy to the class
- **case** *target is (soft/hard)goal* **do**
  - **foreach** *refinement branch from target* **do**
    - add ozWhy to the class schema specifying the first task or resource on the branch
- **otherwise do** add ozWhy to the class specifying the dependee

2 **Analyse each decomposition link in** *Decompositions*

- **let** A and B be respectively the source and destination of the link
- **if** *A is not a softgoal* **then** skip
- **let** ozA be the Object-Z specification of A
- **if** *the Object-Z specification of B is a class schema* **then**
  - Add ozA to the class schema {ozA: Object-Z spec of A}
- **else**
  - **foreach** *refinement branch from B* **do**
    - Add ozA to the class schema of the first task or resource on the branch

3 **Analyse each contribution link in** *Contributions*

- **let** B be the source and A the destination of the link
- **if** *A is not a softgoal* **then** skip
- **let** ozA be the Object-Z specification of A
- **if** *the Object-Z specification of B is a class schema* **then**
  - Add ozA to the class schema {ozA: Object-Z spec of A }
- **else**
  - **foreach** *refinement branch from B* **do**
    - Add ozA to the class schema of the first task or resource on the branch

The inputs and outputs of the algorithm are:

- Inputs: the list of link elements in the Table 5.5, and the specification. The input links are supposedly partitioned into three sub-lists including: *Dependencies*, *Decompositions* and *Contributions*.

    1. *Dependencies* contains the list of all the dependency links in the GRL model.
    2. *Decompositions* contains the list of decompositions.
    3. *Contributions* contains the list of contributions.

- Output: the updated specification.

4.4.5 Finalising the specification

This phase is composed of two important tasks: firstly, to check the entire specification and create triggers in the class *ClsGrlSpec*, and secondly, to validate the specification.

1. Perform an overall checkup of the specification and create triggers, where necessary, in the class *ClsGRLSpec* for those operations that need to be accessed from outside the class.

2. The specification validation is a tedious and important activities that will be fully covered in Chapter 6.

The framework so far developed in this chapter is illustrated in Chapter 5 with a reasonable size case study.

4.5 Chapter conclusion

This chapter has presented an analysis of the relationship between GRL and UCM and brought forward some conceptual gap between the two models, that needs to be consolidated in order to improve on URN construction process. An analysis of the influence of NFRs (softgoals) on software development process was also addressed resulting in a suggestion of the concept of Complementary Non-Functional Actions (CNF-actions). This new concept provides for the means to extend the development of NFRs beyond the GRL model hence, making it possible to propagate the influence of a NFR (continue the refinement of a NFR) throughout the entire software development process. The use of CNF-actions would also facilitate backward traceability e.g., from UCM to GRL and provides for means to report on the impact of GRL softgoals on further software development phases.

More importantly, in line with the systematic NFRs analysis process in Figure 4.4, a mechanism to formalise GRL model elements was presented. A framework to generate an Object-Z specification from an input GRL model was hence elaborated allowing for reasoning formally about GRL models.

The case study description and the application of the framework developed in this chapter are presented next.

Chapter 5

Case study

In the previous chapter, a framework to formalise a GRL model was proposed. The Object-Z specification of each GRL model element was developed.

The transformation process requires, for each GRL model element (type), the use of an Object-Z class schema, namely a template, that specifies the element to transform concrete elements into Object-Z. Such templates were equally developed in Chapter 4. This chapter first proposes a technique that exploits an enterprise organogram to facilitate software projet scoping and initial problem analysis at an early phase of software development. Afterward, a GRL model for the case study is constructed followed by the application of the framework, developed in the previous chapter, to derive an Object-Z specification from the GRL model of the case study.

Next is the case study description.

5.1 Problem description

A research department in a higher education institution has a research portfolio that includes a number of research support programmes for researchers and postgraduate students. A call for a programme is prepared by a programme administrator/manager and emailed to the members of the faculties and departments within the institution for a specific period during which applicants may submit their application forms for evaluation.

To apply, an applicant downloads the application form for the opened programme, completes the form and submits, together with the other required documents, to his/her institution/college. Depending on the programme, the applicant sometimes needs to get the application motivated and signed by the supervisor/promoter and the relevant authorities

within the college or the institution before any submission. For the motivation process, the applicant sometimes needs to carry the hard copy of the application from one office to another.

The current system, that is the application process, the management of research support programmes, as well as the evaluation process of the applications submitted for each programme, presents the following difficulties:

p1: **Limited spaces for emails:**
Each email account has a limited storage space; the sending/receiving of emails becomes impossible when the allocated space is saturated.

p2: **Lack of consistency between the application forms**
Each programme has its own application forms with similar fields duplicated on multiple forms. Generally, those forms when completed at different times even by the same person, do not contain the same information.

p3: **Drawing statistics on application data sent via emails is very hard**
Due to the fact that application data need to be downloaded and kept in folders, the difficulty to assemble files from different folders, access their contents and draw statistics arises.

p4: **Stressful application process**
As briefly describes in the previous section, the process from the downloading of an application form until the submission of the completed form is very time consuming. This becomes harder when the applicant must personally go to the different authorities within the concerned institution to get the application motivated and signed.

p5: **The evaluation/motivation process is hard**
This problem is mainly due to the difficulty of instantly accessing application data submitted via emails or by posting the hard copy.

Let $Problems = \{p1, p2, p3, p4, p5\}$ be the set of problems. It is the author's belief that the seriousness of the problems does not necessary imply the need to invest resources into resolving them before proper analysis is conducted and the scope of the system is well defined.

5.2 Problem analysis and scope definition

Keeping in mind that scope definition is one of the harder challenges in requirements engineering (see for example Michael and Kyo [123], Rajagopal et al. [152], Vijayan and Raju

[183]), Goal-Based requirements analysis generally assumes that goals are not, a priori, documented explicitly. It is, therefore, the responsibility of the requirements engineer to explore various sources of information available to identify, create and organise goals (e.g., Anton and Potts [21]). The sources of information to be explored include: stakeholders, policies, transcripts, workflow diagrams, requirements, mission statements, corporate goals and interview facts. Techniques to perform such exploration have been proposed by Regev and Wegmann [154]:

- Understanding stakeholders' problems and negating them.
- Extracting intentional statements from stakeholders: interview transcripts, enterprise policies, enterprise mission statements, enterprise goals, workflow diagrams, and scenarios.
- Asking "How" and "Why" questions about these initially identified goals in order to move about the goal hierarchy.
- Asking "How else" questions to identify alternative means to achieve goals.

The above guidelines are complemented with heuristics to facilitate the identification of goals from a given source, and address separately goals per type. However, with the increasing complexities of enterprises and software systems, the scope definition problem remains. Enterprise modeling/architecting is known to be an appropriate tool for requirements engineering (Kirikova and Bubenko [104]), yet with various models that may be built to represent the different facets of an enterprise (e.g., functional, informational, resource and organisational Delen et al. [56]), the establishment of appropriate guidelines/pointers to reference relevant sources of information (or the specific areas within the enterprise where needed information can be found) remains challenging.

In this work, we propose the use of an organogram to address the above scope delimitation challenges.

5.2.1 The organogram approach to problem analysis and scope definition

Guidelines for constructing the organogram

(1) Create an organogram for the company, based on business objectives, if no organogram is found.

(2) Transform the organogram to add, at each level, one decisional component to each domain (sub-structure) at that level.

(3) Transform the organogram to include, at each level, operational elements to each domain or sub-domain. Such operational elements should be characterised mainly by the service(s) or operations they perform.

(4) At each level in the organogram and for each decisional component, determine the objectives assigned to such component.

(5) For each operational element, when possible, relate the objectives to other elements within the same domain (horizontal relationship) and to the operational component in the immediate higher-level domain (vertical relationship).

Figure 5.1 results from applying the first three steps of the organogram construction guidelines. A rectangle represents a domain or a sub-domain of the organisation. An ellipse inside a rectangle represents a decisional or managerial component/agent whereas, a standalone ellipse represents an operational component. Since an organogram has a directed graph structure, graph concepts are considered for the modeling. This has the advantage that existing graph theory on modeling and graph search may be exploited. The definitions, presented next and further graph theory that may be utilized in later stages are taken from Baase and Van Gelder [22].

Definition 5.2.1. A directed graph is a pair G = (V, E) where V is a set whose elements are called vertices or nodes, and E is a set of ordered pairs of elements of V. Elements of V are called arcs or directed edges.

Let $x \mapsto y \in E$, x is called the tail of the edge or the direct predecessor of y, whereas, y is called the head of the arc or the direct successor of x.

Definition 5.2.2. A sub-graph of a graph G = (V, E) is a graph G' = (V', E') such that $V' \subseteq V$ and $E' \subseteq E$.

Graph traversal: To reach each vertex and each edge of the graph for their processing, a traversal strategy was adapted from the two fundamental graph search mechanisms [22]: breadth-first and depth-first. From a list of sub-objectives of an operational or a decisional node, the adapted strategy searches the entire organogram and produces the list of nodes and associated objectives that need to be considered during Goal/Requirements elicitation phase.

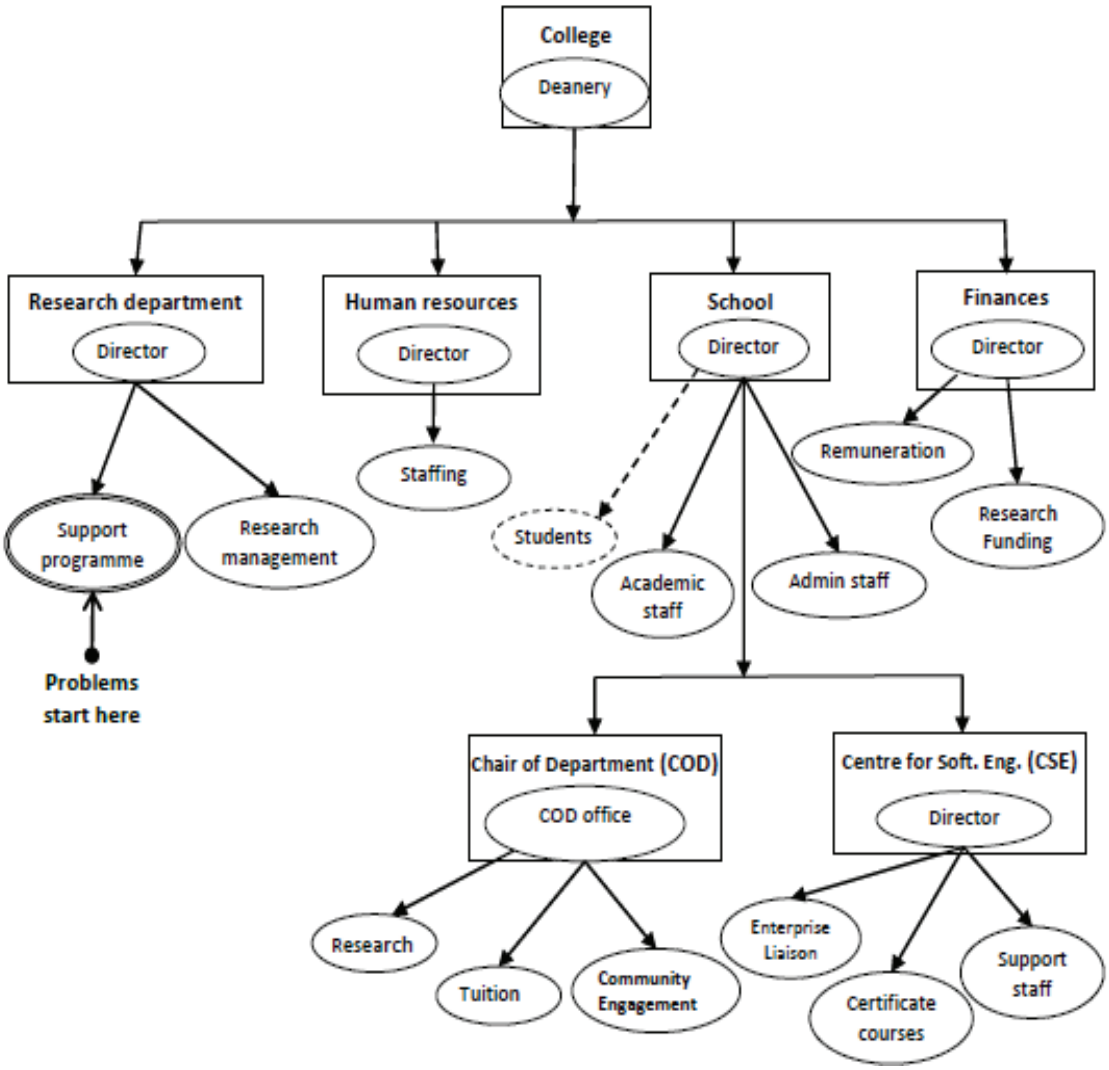

Figure 5.1: College organogram

Algorithm 3: The main algorithm

```
input  : E, CurV, CurObj, CurHrel, CurVrel
output: CurV', CurObj', CurHrel', CurVrel'
begin
    Initialize (CurV, CurObj, CurHrel, CurVrel)
    while there are vertex in CurV not colored black do
        foreach vertex v ∈ CurV do
            if color(v) is white then
                {horizontal processing of v}
                Call Algorithm 4
                color(v) ⟵ grey
                else if color(v) is grey then
                    {vertical processing}
                    Call Algorithm 5
                    color(v) ⟵ black
```

The main algorithm

E is the set of directed edges; CurV contains the currently identified vertices that may be considered during requirements elicitation. CurObj is the set of objectives so far identified and CurHrel and CurVrel represents, respectively, the horizontal and vertical relationships between the currently identified objectives. The initialisation of CurV colors each vertex white. When a vertex is first visited, the horizontal search is performed and the vertex is colored grey.

The horizontal processing algorithm

For a given node, the purpose is to identify those objectives of the node that are in a horizontal relationship with the currently identified objectives.

The vertical processing algorithm

For the input node, the main purpose is to identify direct predecessors and successors of the node which objectives are in vertical relationship with the objectives of the input node.

Algorithm 4: Horizontal search

input : v, CurObj, CurHrel

output: CurObj, CurHrel(updated)

begin

$ListObjectives = \text{ran}\ CurObj$

foreach $Ov \in ran(\{v\} \lhd CurObj)$ **do**

foreach $O \in ListObjectives$ **do**

if $Ov \mapsto O \in hRel$ *or* $O \mapsto Ov \in hRel$ **then**

$CurObj \longleftarrow Ov$

$CurHrel \longleftarrow Ov \mapsto O \mid O \mapsto Ov$

5.2.2 Modeling the organogram as a graph

Before proceeding with the modeling, the construction of the organogram first needs to be completed by applying steps 4 and 5 pertaining to assign business objectives to each operational agent and create both the horizontal and vertical relationships between those objectives.

1. Modeling nodes: a unique identifier is created for each node.

   Example 5.2.1. node identifiers:

NodeId	**Node description**
cd	The office of the dean of the college
crdd	The office of the director of the research department

2. Linking Objectives to nodes: a business objective, as well as, any important element within a company can be modeled similarly to nodes. A unique identifier of an objective is formed by adding to the node's identifier 'ob' followed by an order number.

   Example 5.2.2. two objectives of, respectively, the deanery and the directorate of the research department.

Algorithm 5: Vertical search

input : v, CurObj, CurV, E, CurVrel

output: CurV, CurObj, CurVrel(updated)

begin

$ListObjectives = \text{ran}(\{v\} \lhd nodeObj)$

{analysing the direct predecessor of v }

if *v has direct predecessor w* $(w \mapsto v \in E)$ **then**

if $\exists_1 Ow \mapsto Ov \in vRel \mid Ow : Objective\ of\ w,\ Ov : Objective\ of\ v$ **then**

$w \notin CurV \Rightarrow CurV \longleftarrow w$

foreach $Ow \mapsto Ov \in vRel$ **do**

$CurVrel \longleftarrow Ow \mapsto Ov$

$CurObj \longleftarrow Ow$

{where Ow is an objective of w and Ov an objective of v}

{analysing the direct successors of v}

if *v has at least one direct successor* $s(v \mapsto s \in E)$ **then**

foreach *direct successor s of v* **do**

if $\exists_1 Ov \mapsto Os \in vRel \mid Ov : objective\ of\ v\ and\ Os :\ objective\ of\ s$ **then**

$s \notin CurV \Rightarrow CurV \longleftarrow s$

foreach $Ov \mapsto Os \in vRel$ **do**

$CurObj \longleftarrow Os$

$CurVrel \longleftarrow Ov \mapsto Os$

Identifier	Business objective description
cdob1	Develop research and postgraduate supervision capacity within the institution
crddob1	Timeously report on research funding for different programmes
crddob3	Negotiate research funding and scholarships from industries
crddob4	Attract new postgraduate students

An objective is linked to a node by means of a relation between their identifiers:

$$nodeobj : objectiveId \leftrightarrow nodeId.$$

A complete list of nodes and associated objectives for the case study is presented in Table B.1 of Appendix B.

3. relationships between objectives: Two relationships between business objectives are defined, namely, the horizontal and vertical relationships. The horizontal relationship models the complementarity relationship between the objectives at the same abstraction level, especially the complementarity between the objectives. The vertical relationship links a refined objective to its source.
Horizontal relationship - $Hrel : Objective \rightarrow Objective$
Vertical relationship - $Vrel : Objective \rightarrow Objective$

Example 5.2.3. $crddob3 \mapsto crddob4 \in Hrel$ indicates that objective $crddob4$ is supported or complemented by $crddob3$. In other words, the college efforts to negotiate research funding and scholarships from industries is meant to attract new postgraduate students.

$cdob1 \mapsto crddob1 \in Vrel$ also indicates that objective $crddob1$ was obtained from $cdob1$ by refinement.

The complete lists of vertical relationships and horizontal relationships between business objectives at each node are presented, respectively, in Table B.2 and B.3 of the Appendix B.

The idea so far presented on the modeling and exploitation of enterprise organograms by means of algorithms to address the scope definition challenge in goal/requirements analysis process, was initially published in the proceedings of the Second International Conference On Advances in Computing, Communication and Information Technology CCIT-2014 [63], and later selected and published as a journal article in [66].

5.2.3 The scope definition

The graph model of the case study presented in Section 5.2.1 and Appendix B, was specified in Object-Z and animated with Prolog. The proposed approach to animate an Object-Z specification with Prolog is fully developed in Chapter 6. The Prolog programme used for the case study is presented in Appendix B. The programme implements the three algorithms: Algorithm 1, 2 and 3 to scan, in the light of vertical and horizontal relationships, the organogram in search of candidate domains that maybe included in the scope of goal/requirements analysis.

Since the problems presented in Section 5.1, are rooted from the support programmes department, the search also starts at the node *crdsp*. The initial objectives selected at this node are:

- *crdspob*1: Provide maximum support to applicants.
- *crdspob*2: Increase the number of applicants through advertisement.

```
Amzi! Prolog (IDE Only)
Licensed to
Free Version
Interpreting project: MyProject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: 'operationaleltch5.pro'
Type 'quit.' to end and [Ctrl]-C to stop user input.

?- schema_op([[crdsp], [crdspob1,crdspob2], [], [], []],main).
...
yes
?-
```

The un-formatted version of the output generated by the programme Prolog can be consulted in Section B.0.5, Appendix B in its initial form. The output was re-structured and presented in Table 5.3. Selected nodes are shown in the first column and the associated objectives in the last column. The selected nodes and objectives are justified by the vertical and horizontal relationships contained, respectively, in the second and third columns.

Scope definition (formatted Prolog output)			
Node	**Vertical Relations**	**Horizontal Relations**	**Objectives**
crdsp	crddob2/crdspob2 crddob4/crdspob1, crddob2/crdspob1,	crdspob3/crdspob1	crdspob1, crdspob2, crdspob3.

Scope definition (formatted Prolog output)			
Node	**Vertical Relations**	**Horizontal Relations**	**Objectives**
crdrm	crddob4/crdrmob1		crdrmob1
crdd	cdob2/crddob2, cdob1/crddob4, cdob1/crddob1	crddob3/crddob4	crddob1, crddob2, crddob4, crddob3
cfd	cdob2/cfdob2, cdob2/cfdob1, cdob6/cfdob2 cdob3/cfdob2, cdob3/cfdob1	cfdob2/cfdob1	cfdob2, cfdob1
csd	cdob2/csdob4, cdob2/csdob3 cdob2/csdob1, cdob1/csdob2 cdob5/csdob4, cdob4/csdob4 cdob5/csdob2, cdob4/csdob2	csdob2/csdob1 csdob4/csdob1	csdob1, csdob2, csdob3, csdob4
cscsed	csdob4/cscsedob1	cscsedob1/cscsedob2	cscsedob1
cscd	csdob3/cscdob5, csdob1/cscdob1, csdob2/cscdob3, csdob2/cscdob2	cscdob2/cscdob1 cscdob4/cscdob1, cscdob5/cscdob2	cscdob1, cscdob2, cscdob3, cscdob4, cscdob5
csad	csdob4/csadob3, csdob1/csadob1	csadob2/csadob1, csadob3/csadob1	csadob1, csadob2 csadob3
csas	csdob4/csasob4, csdob4/csasob2, csdob3/csasob3, csdob3/csasob2, csdob1/csasob1, csdob2/csasob3		csasob1, csasob2 csasob3, csasob4
cscdce	cscdob1/cscdceob2		cscdceob2
cscdt	cscdob5/cscdtob2, cscdob2/cscdtob1		cscdtob2, cscdtob1
cscdr	cscdob1/cscdrob1, cscdob2/cscdrob2, cscdob4/cscdrob2		cscdrob1, cscdrob2
cscseel	cscsedob1/cscseelob3 cscsedob1/cscseelob2	cscseelob1/cscseelob3	cscseelob1, cscseelob2 cscseelob3
cfrf	cfdob2/cfrfob2, cfdob2/cfrfob1, cfdob1/cfrfob2,		cfrfob1, cfrfob2
cfr	cfdob2/cfrob1, cfdob1/cfrob1		cfrob1
cd		cdob6/cdob3	cdob1, cdob2, cdob3 cdob4, cdob5, cdob6

Table 5.3: Generated goal/requirements scope

A relation crddob2/crdspob2 in the vertical relationship column is a Prolog implementation of: $crddob2 \mapsto crdspob2 \in Vrel$.

Example 5.2.4. The objective crdspob3 not selected initially was included by Prolog due to the horizontal relationship crdspob3/crdspob1 ($crdspob3 \mapsto crdspob1 \in Hrel$).

Let *selNodes* be the set of selected nodes and objectives in Table 5.3.

$$
\begin{aligned}
&selNodes = \{\\
&\quad crdsp\{ob1, ob2, ob3\}, crdrm\{ob1\}, crdd\{ob1, ob2, ob3, ob4\}, cfd\{ob1, ob2\},\\
&\quad csas\{ob1, ob2, ob3, ob4\}, csd\{ob1, ob2, ob3, ob4\}, csad\{ob1, ob2, ob3\},\\
&\quad cscd\{ob1, ob2, ob3, ob4, ob5\}, cscdr\{ob1, ob2\}, cscdce\{ob2\}, cscdt\{ob1, ob2\},\\
&\quad cscsed\{ob1\}, cscseel\{ob1, ob2, ob3\}, cfr\{ob1\}, cd\{ob1, ob2, ob3, ob4, ob5, ob6\},\\
&\quad cfrf\{ob1, ob2\}\\
&\quad \}
\end{aligned}
$$

An element $crdrm\{ob1\} \in selNodes$ indicates that the node *crdrm* was selected to be included in the scope, as well as its objective $crdrmob1$.

5.2.4 Problem analysis

Analysing the initial problems: $Problems = \{p1, p2, p3, p4, p5\}$, consists to investigate the impact of the problem on *selNodes* with the purpose to eliminate those elements that are not affected and derive appropriate solutions for those elements that are. For example, one may ask what influence would p4 (Stressful application process) have on the achievement of the objective $csdob1$ (increase research output by at least 5%) at *csd* node (the school director's office). Table B.4 in Appendix B illustrates the result of such an analysis.

Example 5.2.5. At the dean's office, a possible impact of P4: stressful application process and P5: A stressful motivation and vetting process may be the following: p4 discourages researchers (whose time is generally limited) and P5 would equally discourages supervisors and managers. Such discouragements would therefore, affect negatively the dean's objective to create and environment that encourages and foster the culture of research within the institution (cdob2).

From the result in Table B.4, the problem domain is obtained by eliminating from *selNodes* nodes and objectives that are not likely to be affected by the initial problems.

$$
\begin{aligned}
&pbDomain = \{\\
&\quad crdsp\{ob1, ob2, ob3\}, crdd\{ob1, ob3\}, cfd\{ob1\},\\
&\quad csas\{ob1, ob2\}, csd\{ob1, ob3, ob4\}, csad\{ob3\},\\
&\quad cscd\{ob1, ob4\}, cscdr\{ob1\}, cscsed\{ob1\},\\
&\quad cscseel\{ob3\}, cfr\{ob1\}, cd\{ob1\}, cfrf\{ob1, ob2\}\\
&\quad \}
\end{aligned}
$$

An interesting aspect of this approach is that, the path representing the propagation of the influence of a specific problem from the initial component to any other one can be re-constructed using the vertical and/or the horizontal relationships.

5.3 The GRL modeling

pbDomain offers an environment to perform traditional goal/requirements analysis process, presented earlier in Section 5.2. Each component of *pbDomain* provides for a location to look for stakeholders, policies, mission statements and more importantly business objectives as well as resources and tasks readily available. Detail understanding of the influence of a problem on one or more components of *pbDomain* would consequently help to derive appropriate solutions in terms of: softgoals, goals, resources, tasks, etc. that should thereafter, be integrated into the goal/requirement model in construction. Assuming the following decisions of the stakeholders:

(1) Resolve the problems due to email space saturation

(2) Facilitate the management of submitted applications

(3) Improve the application and motivation process

(4) Reduce the turnaround time of an application

(5) Ensure consistency between programmes

5.3.1 Solving the problem

The main issue with the limited email size (see Table B.4) is the difficulty to access the submitted applications that in some cases induces more work and causes delays in achieving some business objectives. A possible solution is a well-designed structure to keep data such as database or ontology. Online application forms would equally eliminate the errors due to the use of multiple forms designed for each program; facilitate the application, the evaluation and the motivation processes. It may also reduce the effort to manage the submitted applications and draw statistics. Since the main issues addressed here concerns errors, delays, time, discouragements of applicants and motivators, the chief qualities of the solution would therefore be:

- availability, accessibility and security of data,
- bringing the number of errors in submitted applications close to zero,
- being flexible and user friendly,
- optimising the turnaround time for an application,
- realtime availability of statistics and other processed information.

The GRL construction process presented in Chapter 4, Section 4.1.1, pp.53 - 56 and Figure 4.1, P.54, is followed to construct the GRL model for the case study. The following important aspects are considered: actors (role-players) identification, developing softgoals, and developing hard goals. During softgoals' development, links are also made to hard goals.

5.3.2 Actors identification

The problem domain *pbDomain* is scrutinized to identify all the role-players within the domain. Four actors are therefore identified: applicant, motivator, evaluator, programme administrator and the database server.

Applicant

An applicant is anyone who applies for a research support programme. These include, academic staff members, students as well as any other staff member who does research.

The main concern of the applicant is the improvement of the application process. In GRL construction, the wish to *Improve the application process* constitutes therefore the main intent of the applicant which is progressively refined/decomposed.

Motivator/Evaluator

A motivator is mainly a research student's supervisor or a director. A motivator supports or motivates a researcher's application by providing the researcher with a motivation letter and/or signing the application form. An evaluator is anyone who takes part to a vetting process. Due to the fact that the same people are generally involved in both, the motivation and evaluation, in the GRL model under-construction, any of the terms Motivator and Evaluator invariably refers to both Motivation and Evaluation.

The wish of a motivator is to improve the motivation and evaluation process. In the specification, the motivator's intention to *improve the motivation and evaluation process*, which represents a softgoal in the GRL terminology is progressively refined.

(Support programme) Administrator

A support programme administrator is someone from the *crdsp* department in charge of the management of all support programmes. The programme administrator keeps a list of research support programmes that are regularly updated. Each programme is managed separately since the opening and closing periods differ from one programme to another. When

a programme is opened, manually completed application forms are submitted directly via emails to the programme administrator.

As indicated earlier, the management of applications submitted via emails has been the source of numerous difficulties thus, the wish of the administrator to see those difficulties resolved. In GRL, the concern of the administrator to *resolve email problems* is considered the main softgoal that is progressively refined/decomposed in the GRL model construction process.

Server

A server is a system that maintains the database and provides access to the data. Thus, although a server may not have intentions, it nevertheless provides for services that are needed by other actors and which contribute to the achievement of their objectives. The server maintains a database composed for example, of data on submitted applications, data that facilitate the generation of form sections that are assembled to form an online application form, etc.

5.3.3 The GRL model for the Case study

The model is represented in Figure 5.2.

5.3.4 The GRL model description

An important objective is to improve the application process. This softgoal is decomposed into *flexible application process* and *reducing the overall application time*. Applying and motivating online are believed to contribute to reduce the application time and render the process flexible. Complementarily, being able to *access own submitted applications* for update/consultation would also contribute to render the application process flexible. To perform *online application*, two tasks are required: the first is for *the server to generate application form's sections*, and the second is for the applicant to *complete the application form* and *submit the information online*. It is important to notice that any online activity depends on the availability of the Internet connection, including the online application, as well as the online motivation.

The main concern of the Administrator is to *resolve email space limitation*(softgoal) and hence address the difficulties induced on the management of applications submitted via emails. To achieve this objective, two sub-goals need to be satisfied all together: the first is to *facilitate the management of submitted applications*(softgoal) and the second to *improve*

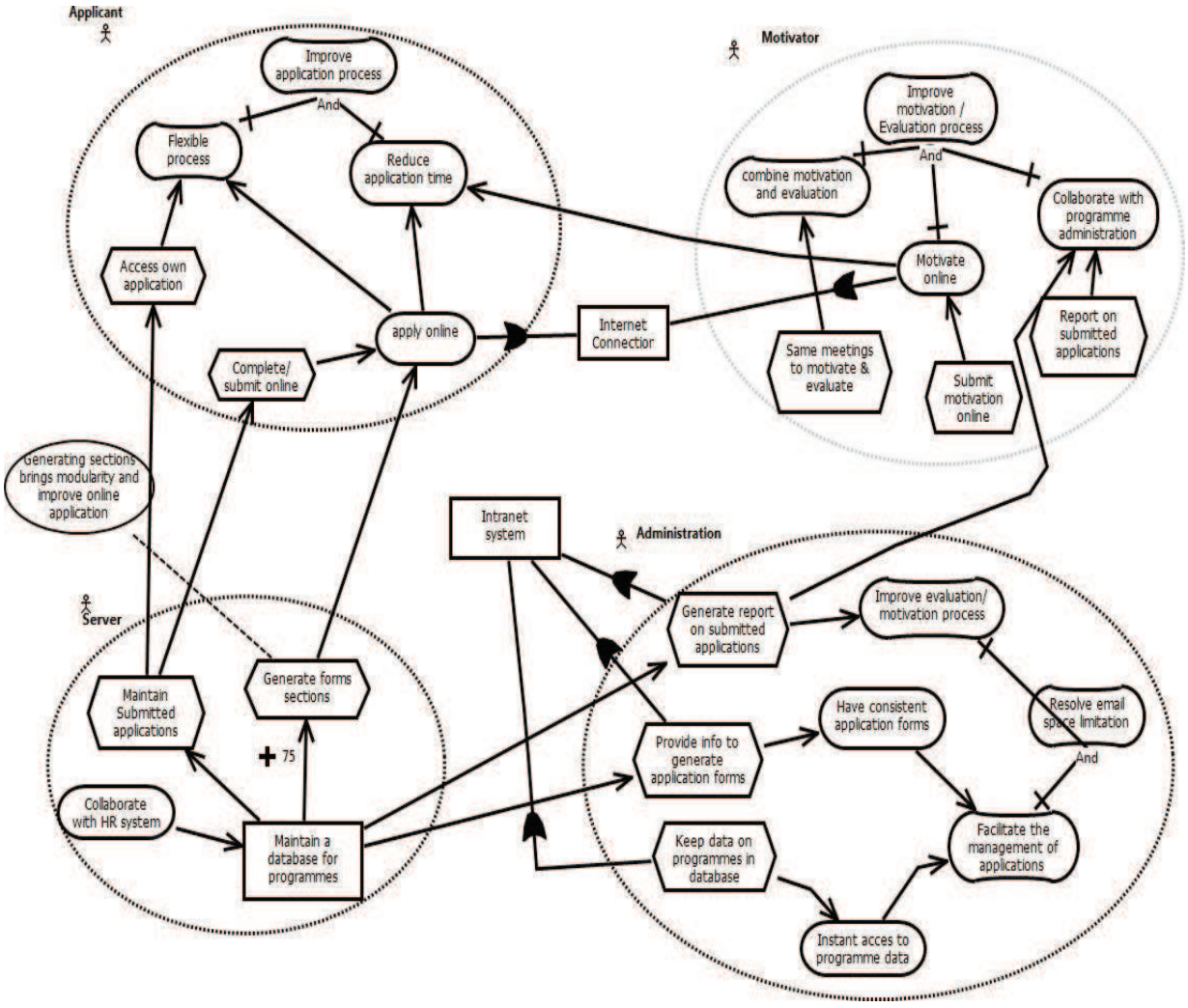

Figure 5.2: Goal model for programmes administration and online application

the evaluation/motivation process(softgoal). To achieve the first one, two other goals must be satisfied: firstly, the administrator must *have instant access to programmes*(hardgoal) and *Have consistent application forms*(hard goal). An instant access to programme data is accomplished by *keeping permanently data on programmes*(Task) and applications in the database. providing the server with *appropriate information on forms' sections*(Task) will help to keep application forms consistent and facilitate their automatic generation by the server. At the Administrator's level, the task that consists to generate reports on submitted applications contribute to *Improve the evaluation and motivation process*.

5.4 Formalising the GRL model of the case study

We now apply the framework for a formal specification of a GRL model, described in (Chapter 4, Section 4.4, p.77), to the GRL model in Figure 5.2. The transformation process summarised in (Chapter 4, Section 4.4.3, p.81) is followed to produce, at each step, the desired outputs.

5.4.1 GRL transformation templates

Templates for each GRL element type were presented in the previous chapter and summarised in (Table 4.5, P.82). We simply use them here to derive the formal specification of the model in Figure 5.2. It is important to note that when a template is used during the formalisation, it becomes part of the specification in construction. For example, defining performance as an instance of the class of softgoals *ClsSoftGoal*, makes the class to be part of the specification in construction.

$$performance : ClsSoftGoal$$

5.4.2 GRL elements identification

As proposed in the framework of Chapter 4, a traversal algorithm is used to scan the input GRL model to determine its elements: actors, elements bound to actors' definitions, free elements (not bound to any actor's definition), and links.

Actors

Traverse(Actors, grlspec): returns the list of all the actors elements in the variable Actors. This will produce the following list:

$$Actors = \{Applicant, Motivator, Administrator, Server\}$$

Intentional elements

Traverse(FreeElements, ActorElements, grlspec): returns the list of all model elements that are not bound to any actor definition in the variable FreeElements. Model elements bound to actors definition are also returned in the variable ActorElements.

$$FreeElements = \{Internet, Intranet, Belief\_01\}$$
$$ActorElements = ApplicantElts \cup MotivatorElts \cup AdministratorElts \cup ServerElts$$

The elements of each actor's definition are presented in Table5.4. Names in the table are obtained by connecting words composing the original name (in the GRL model) one to another in the same order they appear in the original name. Some of the words are first abbreviated.

ActorElements	**ApplicantElts**	**MotivatorElts**	**AdminElts**	**ServerElts**
Softgoals	ImprovAppProc FlexibleProcess	ImprovMotivEvalProc CombineMotivEval	ResolvEmailPb ImprovEvalMotivTime FacilAppManagement	
Goals	ReduceAppTime AppOnline	ColabWithAdmin MotivEvalOnline	HaveConsistForms InstantAccesToPgData	 ColabHRSys
Tasks	AccessOwnApp SubmitAppOnline	MeetingsToMotivEval SubmitMotivEvalOnline ReportOnSubmitedApp	GenStatsOnApp ProvidInfoGenForms KeepPgDataInDb	GenFormSections MaintSubmitedApp MaintDb4SuppPg
Resources	[Intranet] [Internet]	[Intranet] [Internet]	[Intranet]	
Beliefs				GenSecBelief

Table 5.4: Intentional elements of the input GRL model

Links

Traverse(Links, grlspec): returns the list of all the elementlinks in the variable Links.

$$Links = DecompositionElts \cup ContributionElts \cup DependencyElts$$

The elements of each subset of Links are presented in Table5.5 per link type and per actor's definition.

Links	**DecompositionElts**	**ContributionElts**
Applicant	dc1: And (ImprovAppProc,FlexibeProcess) dc2: And(ImprovAppProc, ReduceAppTime)	c1: (AccessOwnApp,FlexibleProcess) c2: (AppOnline,FlexibleProcess) c3: (AppOnline,ReduceAppTime) c4: (SubmitAppOnline, AppOnline)
Motivator	dc3: And(ImprovMotivEvalProc, CombineMotivEval) dc4: And(ImprovMotivEvalProc,MotivEvalOnline) dc5: And(ImproveMotivEvalProc,ColabWithAdmin)	c5: (MeetingsToMotivEval, CombineMotivEval) c6: (submitMotivEvalOnline, MotivEvalOnline) c7: (ReportOnSubmitedApp, ColabWithAdmin)
Administrator	dc6: And(ResolvEmailPb, ImprovEvalMotivTime) dc7: And(ResolvEmailPb, FacilAppManagement)	c8: (InstantAccessToPgData, FacilAppManag) c9: (HaveConsistForms, FacilAppManag) c10: (GenStatsOnApp, ImprovEvalMotivTime) c11: (KeepPgDataInDb, InstantAccessToPgData) c12: (ProvidInfoGenForms, HaveConsistForms)
Server		c13: (MaintDb4SuppPg, GenFormSections) c14: (MaintDb4SuppPg, MaintSubmitedApp) c15: (GenFormSections, AppOnline)

Links	**DecompositionElts**	**ContributionElts**
Links	**DependencyElts**	
	dp1: (Applicant:ApplyOnline,Free:Internet) dp2: (Motivator:MotivEvalOnline,Free:Internet) dp3: (Administrator:GenStatsOnApp, Free:Intranet) dp4: (Administrator:ProvidInfoGenForms, Free:Intranet) dp5: (Administrator:KeepPgDataInDb,Free:Intranet)	

Table 5.5: Link elements in the GRL model for the case study

When an input GRL model has a large number of elements, it is important to structure those elements to facilitate the specification process and ensure that all the elements are considered during the process. Such a planning for the case study is presented next.

5.4.3 Planning the specification

Having identified all the GRL elements, from the input model, that are to be formalised, it is then possible to plan for the list of Object-Z elements that are to be derived and if possible, structure them, in the order in which they will be created. Table 5.6 is an example of such a planning for the case study.

order	Object-Z Elements			GRL Element	
No.	To be Created	Container	Template		Strategy
1	ClsApplicant		ClsActors	Applicant	Inheritance
1	ClsMotivator		ClsActors	Motivator	Inheritance
1	ClsAdministrator		ClsActors	Administrator	Inheritance
1	ClsServer		ClsActors	Server	Inheritance
4	ozImprovAppProc	ClsApplicant	ClsSoftGoal	ImprovAppProc	Instantiate
4	ozFlexibleProcess	ClsApplicant	ClsSoftGoal	FlexibleProcess	Instantiate
4	ozImprovMotivEvalProc	ClsMotivator	ClsSoftGoal	ImprovMotivEvalProc	Instantiate
4	ozCombineMotivEval	ClsMotivator	ClsSoftGoal	CombineMotivEval	Instantiate
4	ozResolvEmailPb	ClsAdministrator	ClsSoftGoal	ResolvEmailPb	Instantiate
4	ozImprovEvalMotivTime	ClsAdministrator	ClsSoftGoal	ImprovEvalMotivTime	Instantiate
4	ozFacilAppManagement	ClsAdministrator	ClsSoftGoal	FacilAppManagement	Instantiate
4	ozReduceAppTime	ClsApplicant	ClsGoal	ReduceAppTime	Instantiate
4	ozAppOnline	ClsApplicant	ClsGoal	AppOnline	Instantiate
4	ozColabWithAdmin	ClsMotivator	ClsGoal	ColabWithAdmin	Instantiate
4	ozMotivEvalOnline	ClsMotivator	ClsGoal	MotivEvalOnline	Instantiate
4	ozHaveConsistForms	ClsAdministrator	ClsGoal	HaveConsistForms	Instantiate
4	ozInstantAccessToPgData	ClsAdministrator	ClsGoal	InstantAccessToPgData	Instantiate
4	ozColabHRSys	ClsServer	ClsGoal	HaveConsistForms	Instantiate
2	ClsAccessOwnApp	ClsApplicant	ClsTask	AccessOwnApp	Inheritance
2	ClsSubmitAppOnline	ClsApplicant	ClsTask	SubmitAppOnline	Inheritance
2	ClsMeetingToMotivEval	ClsMotivator	ClsTask	MeetingsToMotivEval	Inheritance
2	ClsSubmitMotivEvalOnline	ClsMotivator	ClsTask	SubmitMotivEvalOnline	Inheritance

order	Object-Z Elements			GRL Element	
No.	To be Created	Container	Template		Strategy
2	ClsReportOnSubmitedApp	ClsMotivator	ClsTask	ReportOnSubmitedApp	Inheritance
2	ClsGenStatsOnApp	ClsAdministrator	ClsTask	GenStatsOnApp	Inheritance
2	ClsProvideInfoGenForms	ClsAdministrator	ClsTask	ProvideInfoGenForms	Inheritance
2	ClsKeepPgDataInDb	ClsAdministrator	ClsTask	KeepPgDataInDb	Inheritance
2	ClsGenFormSections	ClsServer	ClsTask	GenFormSections	Inheritance
2	ClsMaintSubmitedApp	ClsServer	ClsTask	MaintSubmitedApp	Inheritance
2	ClsMaintDb4SuppPg	ClsServer	ClsTask	MaintDb4SuppPg	Inheritance
3	ClsIntranet		ClsResource	Intranet	Inheritance
3	ClsInternet		ClsResource	Internet	Inheritance
5	ozGenSecBelief		ClsBelief	GenSecBelief	Instantiate

Table 5.6: List of Object-Z Elements to be Created

Without being prescriptive, the numbers in the first column of Table 5.6 suggest the order in which Object-Z element should be created. For example, it is necessary to first create class schemas describing actors' definitions which are containers before creating the elements that they contain.

Link elements are not included in Table 5.6 to keep the table's size manageable. However, it is to be recalled that each link element is formalised by instantiating one of the three templates: *ClsDependency*, *ClsContribution*, and *ClsDecomposition*.

The other important class to be defined is the class *ClsGrlSpec* through which most of the instantiations are performed to create and keep objects describing softgoals, goals, beliefs and link elements.

In total, 59 (fifty nine) Object-Z elements to be created are summarised in Table 5.7.

No.	Element	Number	type object
1	actors	4	class schemas
2	sofgoals	7	objects
3	goals	7	objects
4	tasks	11	class schemas
5	resources	2	class schemas
6	belief	1	object
7	links	26	objects
8	grlspec	1	class schema

No.	Element	Number	type object
Total :		59	

Table 5.7: Summary of Object-Z elements to be created

In view of the relatively large number of class schemas and objects to be created, we have limited the specification to:

- one actor's definition with all the elements that it contains, as well as any link element in which the actor or any of its elements is involved in.
- the two resources *Intranet* and *Internet* that are not bound to any actor.
- the generic class schema named *ClsGrlSpec* that normally defines most of the functionalities of the system, as well as data pertaining to represent the big picture of the model being formalised.

5.4.4 Formalising the identified actors

Since the specification process of actors is the same, we present only one case, that is, the formalisation of the GRL's actor named *Applicant* to illustrate the process. In the light of the framework in Chapter 4, we first explain how different parts of the class schema were created and present resulting Object-Z class schema thereafter with, when necessary, additional description of its contain.

Creating the class schema: *ClsActorApplicant*

The class schema is generated from the template class *ClsActor* by inheritance (see Chapter 4, Section 4.4.2, p. 79). The class name is obtained by adding the class name *Applicant* to that of the Object-Z template *ClsActor*. The new class (subclass) extends the superclass with features specific to the actor under-consideration.

Creating the components of the class

The state of the class includes three type of components:

- *Components inherited from the superclass ClsActor.*

  $id : Identifier$
  $name : String$
  $mdata : \mathbb{F}\ ClsMetaData$
  $importance : ImportanceType$
  $impquantitative : \mathbb{Z}$

id is the GRL identifier of the GRL element being specified and *name* is the name of the element. The class *ClsMetaData* was defined in the previous chapter (see Section 4.3.3, P.70). These variables are inherited by all the class schemas created in this document to specify GRL linkable elements including Actors, Tasks and Resources.

- *Components that are common to all actors class*:
  The following components are included in all such schemas: *softgoals*, *goals*, *beliefs* and *nfdependers*. The first three components specify respectively, the list of softgoals, goals and beliefs bound to the actor. The variable *nfdependers* keeps a record of Object-Z objects references specifying softgoals which achievement depends directly or indirectly on the GRL actor being specified.

  It is important to note that the intentional elements Tasks and Resources bound to an actor's definition are specified as class schemas in Object-Z. Each of such classes includes a variable that keeps a reference to the class schema describing the actor to which the GRL tasks are bound. That is why, unlike *softgoals*, *goals* and *beliefs*, there is no variable in the class schema of the actor to capture Object-Z objects describing tasks and resources.

- *Components specifying characteristics or properties specific to each actor.*
  A class schema describing an actor's definition may include variables to specify properties specific to that particular type of actors. For example, if needed, a full profile of an actor may be included: National Identity Card/book number (applicantid), name, surname, home address, work address, email address, ... To keep it simple, only the *applicantid* is included.

Creating the operations of the class

As shown in Table 5.6, the elements of each of the three components *softgoals*, *goals* and *beliefs* are created by instantiating their respective templates. To performed this, the following operations are created within the actor's class schema:

- *NewSoftGoal*: creates a new object by instantiating the class *ClsSoftGoal* of softgoals, and adds to the variable *softgoals*. The class *ClsSoftGoal* thus, becomes part of the specification.

- *NewGoal*: creates a new object by instantiating the class *ClsGoal* of goals, and adds the object to the variable *goals*. The class *ClsGoal* also known as a template, becomes part of the specification if not yet added.

- *NewBelief*: creates a new object by instantiating the class *ClsBelief*, a generic class specifying GRL elements of type belief and add to the variable *beliefs*. Similarly to the classes *ClsSoftGoal* and *ClsGoal*, the class *ClsBelief* becomes part of the specification.

- *Addnfdepender*: this operation adds an input object of type *ClsSoftGoal* to the variable *nfdependers*.

  An object of the class *ClsSoftGoal* is considered when the class containing the component *nfdependers*, to which it is added, specifies a Grl element that refines (directly or indirectly) the softgoal. The choice of such an object (describing a softgoal) is based on the analysis of the link elements, summarised in Algorithm 2, on page 87.

In general, as recommended in our Object-Z formalisation framework (see Chapter 4, Section 4.4.4, pp.84 - 88) after using a template for the first time in a specification, the template is added to the specification under construction. That is why the following classes used in this section are included to the specification: *ClsSoftGoal*, *ClsGoal*, *ClsBelief*, as well as *ClsActor*.

Class schema representation

ClsActorApplicant

$\lceil(name, softgoals, goals, beliefs, nfdependers)$
ClsActor

$applicantid : Identifier$
$softgoals : \mathbb{P}\ ClsSoftGoal$
$goals : \mathbb{P}\ ClsGoals$
$beliefs : \mathbb{P}\ ClsBelief$
$nfdependers : \mathbb{P}\ ClsSoftgoal$

$\forall\ nfelt \in nfdependers \bullet nfelt.actor.id \neq id$

INIT

$softgoals = \bot \land\ nfdependers = \bot \land\ goals = \bot \land\ beliefs = \bot$
$ClsActor.Init$

NewSoftGoal

$\Delta(softgoals)$
$sgoal? : ClsSoftGoal$

$softgoals' = softgoals \oplus \{sgoal?\}$

NewGoal

$\Delta(goals)$
$goal? : ClsGoal$

$goal?.Init\ \land\ goal?.actor.id = id$
$goals' = goals \oplus \{goal?\}$

NewBelief

$\Delta(beliefs)$
$belief? : ClsBelief$

$belief?.actor.id = id$
$beliefs' = beliefs \oplus \{belief?\}$

Addnfdepender

$\Delta(nfdependers)$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

The initialisation operation

Initially, the four variables are empty. The *applicantid* and other characteristics (if any) such as the id are setup when the class schema is created. The inherited components are

initialised by activating *ClsActor.Init*.

Text description

The operation *NewSoftGoal* initialises the instance *sgoal*? of softgoals' template passed to the operation as the unique input. The identifier of the class *ClsActorApplicant* is passed to the property *sgoal*?.*actor.id* and *sgoal*?, which is in fact a reference to the real object, is added to the variable *softgoals*. The operations so far specified are limited to the essential for illustration purpose. In practice, one would include more operations for example to update the components.

5.4.5 Formalising the tasks: *AccessOwnApp* and *SubmitOnline*

The two tasks bound to the actor's definition *Applicant* that are to be specified are: *Access own application* and *Submit application online*. As recommended by the transformation framework, two class schemas are created: *ClsAccessOwnApp* and *ClsSubmitOnline*.

The two classes play complementary roles; the first class (*ClsAccessOwnApp*) helps the applicant to access its application and hence, allowing the owner to modify the contain. The second class provides the applicant with the possibility to upload the modified application back to its storage place: the database.

The construction process of these classes is the same. Thus, only one case is presented next; the construction of the class schema *ClsAccessOwnApp*.

Creating the class: *ClsAccessOwnApp*

The class schema is generated from the template class *ClsTask* by inheritance (see Chapter 4, Section 4.4.2, p. 79). The new class (subclass) extends the superclass with features specific to the actor under-consideration and the template itself becomes part of the specification.

Creating the state of the class

The state schema of this class is composed the state inherited from the superclass *ClsTask*, as well as variables specifying properties specific to this particular task. The inherited

components are:

$id : Identifier$
$name : String$
$mdata : \mathbb{F}\ ClsMetaData$
$importance : ImportanceType$
$impquantitative : \mathbb{Z}$
$actor : ClsActor$

The only variable specific to the task namely, *apps*, is used to illustrate the idea. The component *apps* keeps the list of all the submitted applications from which one is to be accessed by the applicant.

Creating the operations

Two operations are created: *SelectApp* and *Addnfdepender*.

- *SelectApp* is created to perform the selection of the application
- *Addnfdepender* to update the list of softgoals whose achievement depends on the element being specified.

Graphical representation of the class

The list represents in fact a view, accessible by the applicant, of the applications in a database.

ClsAccessOwnApp

$\lceil(apps, SelectApp)$
$ClsIntentionalElement$
$[Application]$

$apps : \mathbb{P}\ Application$
$nfdependers : \mathbb{P}\ ClsSoftGoal$

SelectApp

$app! : Application$

$apps \neq \varnothing \ \land\ \exists\, app \in apps \bullet app! = app$

Addnfdepender

$\Delta(nfdependers)$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

The variable *apps* contains a list of applications submitted by applicants for different research support programmes. The operation *SelectApp* accesses the list of submitted applications and allowing the applicant to select any of its applications.

5.4.6 Formalising the identified resources

Two resources are to be specified: *ClsIntranet* and *ClsInternet*.

Creating the class: *ClsIntranet*

ClsIntranet

$\lceil(networkadmin)$
$ClsResource$
$[Person]$

$networkadmin : Person$
$nfdependers : \mathbb{P}\ ClsSoftGoal$

Addnfdepender

$\Delta(beliefs)$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

At this stage, it is difficult to decide which aspect of the resource to describe. It seems reasonable that the support programme administrator (dependee actor) would be more interested in knowing who to contact in case the resource (Intranet) is not available: network problem. However, if non-functional requirements like performance and/or security are to be considered, properties such as network type and structure will have to be included in the specification.

Creating the class: *ClsInternet*

ClsInternet

$\lceil(provider)$
ClsResource
$[Internetprovider]$

$provider : Internetprovider$
$nfdependers : \mathbb{P}\ ClsSoftGoal$

Addnfdepender

$\Delta(nfdependers)$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

As in the case of the class *ClsIntranet*, we limit the properties of the Internet to the knowledge of the internet provider. At this stage one would have created class schemas for all the actors, tasks and resources. Next, we engage with the specification of the class *ClsGrlSpec* as recommended in the framework.

5.4.7 Formalising the system class: ClsGrlCaseStudy

The main purpose of this class is to provide for an environment from which other classes can be accessed and manipulated.

Creating the Class schema: *ClsGrlCaseStudy*

The suggested class name contains the prefix *ClsGrlSpec* followed by any suffix that best describes the case at hand. The structure of this class does not differ from one case to another; mainly the content of variables will differ and in some cases more or fewer operations may be specified.

Creating the class components

The essential components of this class are: *actors*, *freeelts*, and *alllinks*. The variable *actors* contains the list of all the references to actors' class schemas, whereas, *freeelts* keeps the list of all the Object-Z classes and subclasses that formalise GRL elements that are not contained in any actor's definition. The variable *alllinks* captures the reference to all the link elements within the specification. Initially, all these tree components are empty. The predicate part of the schema ensures that no element in *freeelts* is bound to an actor's definition.

Creating the operations needed for the class

Three operations are defined to update the variables *actors*, *freeelts* and *alllinks* so far defined; and one operation to bound an element to an actor's class: *AddActor*, *NewFreeelt*, *NewLink* and *BoundIntentionElt*. In practice, many more operations may be specified for example, to remove an object from a given component.

ClsGrlCaseStudy

$\lceil(freeelts, actorselts, alllinks)$

$actors : \mathbb{P}\, Identifier$
$freeelts : \mathbb{P}\downarrow ClsIntentionalElement$
$alllinks : \mathbb{P}\downarrow ClsElementLink$

$\forall\, elt \in free\_elts \bullet elt.actor = \perp$

INIT

$actors = \varnothing \land \ freeelts = \varnothing \ \land \ alllinks = \varnothing$

AddActor $[id]$

$\Delta(actors)$

$actors' = actors \oplus \{id\}$

NewFreeelt

$\Delta(freeelts)$
$elt? : \downarrow ClsIntentionalElement$

$elt?.actor.id = \perp$
$freeelts' = freeelts \oplus \{elt?\}$

NewLink

$\Delta(alllinks)$
$link? : \downarrow ClsElementLink$

$alllinks' = alllinks \oplus \{link?\}$

BoundIntentElt

$elt? \in \{ClsTask, ClsResource\}$
$actor? : Actor$

$elt?.actor = actor?$

Some text description

Since an actor's specification is built outside this class, the operation *AddActor* is meant to add the actor's id to the list *actors*. The identifier assigned to a class schema of an actor is from the original GRL actor's definition. It is the same for all the instances/objects of the

actor's class and thus is more likely to identify the class as a whole than an object reference which identifies a specific object. The operation *NewFreeelt* creates an Object-Z object representing a GRL intentional element from the template and keeps the newly created object's reference in the variable *freeelts*. Before adding any object to the list, the system cautiously checks to ensure that the object does not already exist. Similarly to *NewFreeelt*, *NewLink* creates an Object-Z's object specifying a GRL link and adds the reference of the new object to the variable *alllinks*. *BoundIntentionElt*: bounds a class specifying a task or a resource to an actor's class schema.

5.4.8 Updating the specification in the light of links

The impact of each type of link elements on the Object-Z specification of a GRL model was presented in (Chapter 4, Section 4.4.4, pp.84 - 88) and summarised in Algorithm 2. The list of all the GRL links in the GRL model of the case study was presented in Table 5.5. Each link element in the table is analysed and the specification updated when necessary as the algorithm instructs. In the algorithm, link elements are treated separately according to their types . The analysis of the dependency link is the first, followed by the decomposition links, and finally, the analysis of the contribution links is presented. As one may deduce from the algorithm, the main activity that results from the analysis of the link elements is to add objects representing softgoals to specifics components in class schemas.

Noting that each of the class schemas that needs to be updated already includes the component *nfdependers* to keep such objects, and the operation *Addnfdependers* to add the object into the component, the additional design action that should be taken is to create an interface operation in the class *ClsGrlCaseStudy*:

$$Newnfdependers \mathrel{\widehat=} [newob : ClsSoftGoal, \exists\, myClass] \bullet myClass.Addnfdependers$$

The input to the operation *Newnfdependers* is an object of the class of softgoals. The system selects a class schema and activates the operation *Addnfdependers* of the class to add the new object to the variable *nfdependers* of the selected class.

5.4.9 Finalising the specification

This phase involves mainly the validation of the specification. One may think of performing the last checkup of the specification, create appropriate interface operations when necessary, adjust the visibility of components and operations, review the relationships between different class schemas, and/or proceed with a systematic validation of the whole specification.

The validation of the specification

Considering the inherent difficulties involved in a specification validation, this phase is separately covered in detail in Chapter 6.

5.5 Chapter conclusion

In this chapter, we have proposed an approach that allows an enterprise organogram to be exploited, by means of graph theory and three algorithms, to facilitate the scoping of software projets in an early phase of requirements analysis and elicitation. The idea was developed as a full paper first published in the proceeding of the Second International Conference on Advances In Computing, Communication and Information Technology-CCIT 2014 [63] and later selected for the International Journal of Software Engineering and Research Methodology [66]. To ensure the validity of the suggested approach, a prototype was developed in Prolog to implement the algorithms and successfully applied to the case study.

After the scope definition as described above, problem analysis was performed and an adapted GRL construction process based on the process presented in the previous chapter (Chapter 4, Section 4.1.1, pp.53 - 56 and Figure 4.1, P.54) was followed to build a GRL model for the case study. To formalise the GRL model of the case study, Algorithms 1 and 2 were cautiously derived from the framework developed in the previous chapter (Chapter 4, Section 4.4, PP.77-84) to guide the transformation process. By applying the two algorithms to the input GRL model, we obtained an Object-Z specification.

The last part of Algorithm 1 prescribes the final review and the validation of the specification. Due to the importance and the inherent tedious nature of specification validation in general, this last specification phase is fully presented in (a separate chapter) Chapter 6.

Chapter 6

Validation of the case study specification

In the previous chapter, two algorithms were derived from the framework, developed in Chapter 4, to guide the formalisation of a GRL model. These algorithms were applied to the GRL model of the case study, built from scratch in the light of the GRL construction process, to produce an Object-Z specification. Due to the importance of ensuring the correctness of the specification, the main objective of this chapter is to review the specification and proceed with its validation following the four-way framework for validating a specification (see Dongmo and van der Poll [61]).

Next is an overview of the different categories of a specification validation.

6.1 Specification validation

Validating a formal specification is a tedious task that has been studied by researchers and practitioners over many years. Different levels/categories of validation are employed, each of which addresses a specific aspect of the specification and requires specific tools:

- ***Reviews and Inspection*** - this involves manually checking a specification to detect and correct problems, for example, Fagan inspections (see Doolan [67], Fagan [71], Kamsties [101]). This technique is less rigorous and requires a fair amount of human effort.

- ***Parsing and type-checking*** - these two techniques are concerned mainly with detecting errors related to the specification language (syntactic and semantic errors) and aim to ensure the internal consistency of the specification (see Johnston [98]). Most of the tool support available perform both parsing and type-checking (see Parker [144]).

- ***Animation*** - animating a specification involves executing the specification with appropriately selected test data and observing its behaviour (see Gray and Schach [82], Hasselbring [87]). An animation process generally includes two major phases: the transformation of the specification into an executable form, followed by the execution phase. Although animation techniques are less rigorous than formal proofs, they have been widely adopted as a means for prototyping formal specifications.

- ***Mathematical proofs*** - properties of the system are formulated as theorems, of which the proofs are discharged in either an automated or semi-automated fashion by specialized software, namely theorem provers (see Freitas [76], McCune [120]).

Each of these categories has its own benefits, complexities, particularities and the circumstances under which we may get the best of benefits from the method. Two examples to illustrate: first, the mathematical proofs and second the animation.

Mathematical proofs, in particular, and formal methods in general, are argued to be more cost-effective for safety-critical systems; most of the known successful projects tend to be supportive (Gerhart et al. [78], Hall [85], Haxthausen [90], Wassyng and Lawford [186], Woodcock et al. [195]). The strength of these methods are mainly attributed to precision, rigour and the level of detailed analysis performed during the specification process. Such expectations from a formal specification make the process of transforming informal descriptions of the initial user requirements into mathematical-like expressions a tedious and difficult task. However, once well-formalised, it becomes possible to automate the validation of the specification by applying theorem provers to the mathematically formulated properties of the system (Van der Poll [176]). Such a validation is also argued to require a high level of expertise and is hence equally demanding in terms of efforts and difficulties. Hence, due to the difficulties attributed to these methods, Formal methods has been mainly used in academia and on an increasing number of safety-critical projects. Its adoption in industry, initially very contested, is also increasingly gaining trust among software development practitioners as revealed in a survey by Bicarregui et al. [25].

Specification animation, also known as prototyping, is another important validation technique that has been widely used on different software models (e.g., MacEwen [116]). As typified by Sommerville [169], the benefits of prototyping includes amongst others, the elicitation and validation of user requirements (throw-away prototyping) and the evaluation of proposed solutions for feasibility, performance, etc. (experimental prototyping). In formal specifications, the term animation is the most commonly used (Salman [160], West [190]). Despite criticisms raised against it for not being rigorous enough, research in favour of ani-

mating formal specifications has been abundant. Amongst the most prominent reasons put forward in favour of animation is the ability to make the complex nature of mathematical notations transparent, thereby facilitating discussions between developers, users and other stakeholders (Gray and Schach [82], Liu and Wang [111]).

Although observable progress has been done by researchers to provide formal methods' users with techniques and tools, the scarcity of animation techniques in Object-Z is still very crucial. Next, we present an approach for animating an Object-Z specification with Prolog; derived from the existing methods developed to animate a Z specification. The approach we are proposing was developed as a full paper first published in (Dongmo and Van der Poll [62]) and later selected an published as a journal article in (Dongmo and Van der Poll [65]).

6.2 An approach to animate a Z/Object-Z specification with Prolog

Although approaches and tools to validate Object-Z specifications, especially animators and theorem provers, are still rather scarce, many more have been developed for Z. Many of the methods for animating Z use Prolog. Two main approaches have been proposed: formal program synthesis and structure simulation (see West and Eaglestone [188]). Formal program synthesis obtains a Prolog program from the Z specification by means of a direct two-step transformation of Z schemas. First any higher-order Z constructs are rewritten as first-order formulae, and second such first-order formulae are converted into Prolog. The challenge with this approach is that the second step is manual - there is no suitable algorithm to turn the first-order specifications into logic programs (see Nakamura [139]). Following structure simulation, a Prolog program is created based on the characteristics of the Z specification, which may have been "flattened" by eight (8) guiding rules derived for this approach (see West and Eaglestone [188]). Seven (7) of these rules are adapted in this thesis for the Z/Prolog transformation that follows :

(1) For each Z schema, create the following Prolog predicates:

- schema_type (L, N) for state schemas and schema_op (L, N) for operations. L is the list of variables associated to schema N.
- givenset (S, N) where S is a given set and N the given set name.

(2) Possible values of variables in a Z schema are described in the body of the clause using the logical relationship.

(3) Concerning Z types, the given sets characterising the schema are specified first. Each declared (type) variable is represented by two predicates, the one naming, the other giving the type. Decoration of variable names is achieved by Prolog functions; thus s? and s! are named in(s) and out(s), respectively, where s is the base name. Similarly, x and d(x) name a state variable and the post-operational state.

(4) Set operations, such as intersection and union, are (assumed to be) contained in a library of Prolog code. Otherwise, we implement these when needed.

(5) A variable that is existentially quantified in a Z schema's clause appears in the body of the Prolog translation of the clause as a Prolog variable, which does not appear in the declarations of a named variable.

(6) The Prolog translation of the conjunction C of two schemas A and B is obtained as follows: translate the signature of C, which is obtained by merging the signatures of A and B. The Prolog translation of the predicates of A and B are conjoined to obtain that of C. An analogous rule applies to the disjunction of two schemas.

(7) When a schema B is used as a type in the signature of A, during the translation, schema B is conjoined to the signature of schema A.

6.2.1 Guidelines for animating an Object-Z specification in Prolog

A three-fold process is followed for the animation: Firstly, we unfold the Object-Z classes to extract the encapsulated Z schemas. Secondly, we transform each Z element into Prolog and lastly, proceed with the execution.

Unfolding Object-Z classes

Since an Object-Z specification normally encapsulates standard Z elements with very little modifications, we take advantage of the existing Z transformation guidelines. We unpack each class individually to work on the embedded Z elements. When necessary, an identified Z element may be slightly modified to undo slight changes due to Object-Z transformation.

Transforming Z schemas into Prolog

We use the above 7 guidelines to transform each Z schema into a Prolog program. As and when necessary, more explanation is provided during the transformation process. The symbol "/" is used to couple related elements within a relation or a function.

Animating the specification

Two standard questions are used to guide the execution of the specification: Are we building the correct system? (Validation) and are we building it right? (Verification). The first question concerns the validation of the specification against the initial requirements, stakeholder goals (upward validation), as well as constraints from the application domain (leftward validation). The second question addresses the consistency and correctness of the specification (rightward validation). A successful animation of the specification also indicates that the specification can be transformed into operational software (downward validation).

As noted by West [189], an important requirement for a successful animation is the ability to trace back in the specification the source of errors when they occur. This requirement is achieved by creating a Prolog program that mimics the structure of any Z element under consideration; whenever possible, the names used in the specification are conserved in Prolog.

Considering the fact that each of the validation categories, presented in Section 6.1, may be applied in isolation or in combination with others, to the best of the authors' knowledge, there isn't yet an approach, a framework, guidelines or algorithms available to combine those categories in a specification validation. Thus, for the validation of the Object-Z specification of the case study, we have chosen to apply the four-way framework for validating a specification where at each of the four phases, a decision is made whereby among the categories, the most appropriate one is chosen.

6.3 Overview of the validation approach: 4-way framework for specification validation

The four-way framework for validating a specification (see Dongmo and van der Poll [61]), proposes to iteratively validate and amend a specification until the desired quality is obtained. As shown on Figure 6.1, at each iteration four validation phases are considered; with each phase focused on one important aspect of the specification.

- The rightward phase targets the properties of the specification related to the specification language and any associated tool support.
- The upward validation focuses on the properties of the specifications pertaining to stakeholders' expectations and initial goals.
- The leftward phase addresses attributes of the application domain,

- The downward validation ensures that the specification can eventually lead to the envisioned software product.

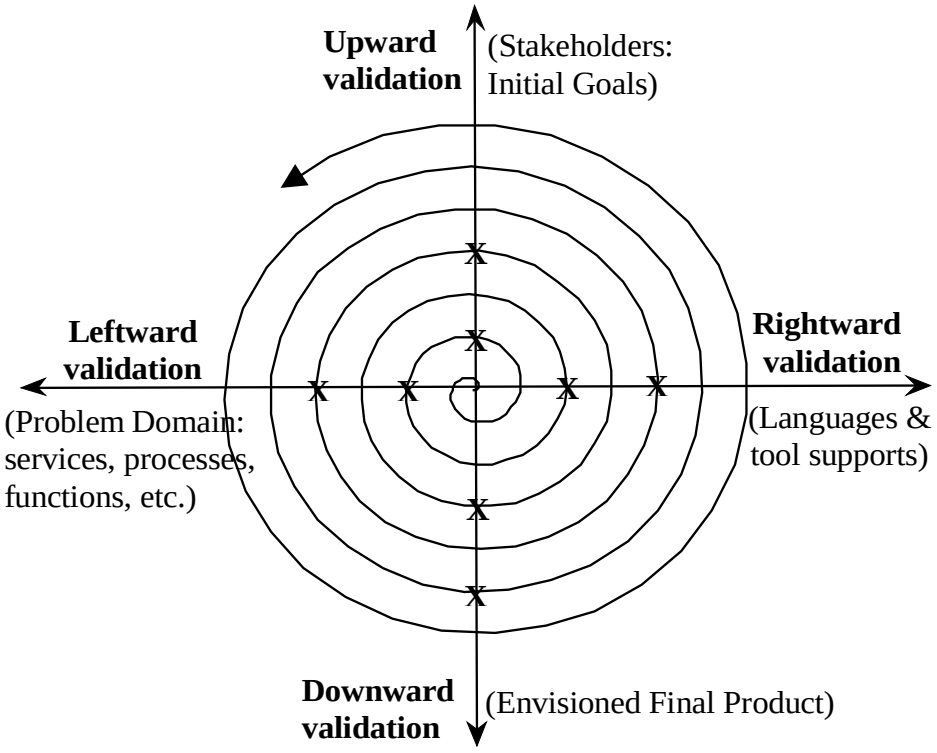

Figure 6.1: Four-way framework for validating a specification (see [61])

The validation process is repeated over the four phases, with the specification progressively updated at each phase, until the desired level of satisfaction is reached. The order of the phases is not prescriptive, but in this work, we start with the rightward phase to first ensure that the specification is well-formed and can, therefore, serve as input to the available tools.

6.4 Planning the validation for one iteration

The overall planning of the validation of the Object-Z specification of the case study for one iteration is presented in Table 6.1. The first column represents the validation phases, the second the validation techniques/methods and the third the tool supports. The last column represents the properties that needs to be validated.

Planning for one iteration			
Phases	**Technique/Method**	**Tool support**	**Target property**
Rightward	Review/Cross-referencing	Manual	consistency
	Parsing/Type checking	CZT	(Internal consistency)

Planning for one iteration			
Phases	**Technique/Method**	**Tool support**	**Target property**
Upward	Review Animation/Prototype	Manual Prolog	Traceability Correctness
Leftward	Domain analysis Animation/Prototype Theorem prover	Prolog Z/Eve Manual	Completeness Applicability
Downward	UCM modeling/Tables	jUCMNav	Feasibility

Table 6.1: Planning the validation of the Object-Z specification

A specification validation is an important task in software development which planning, we believe, should be well-prepared. One aspect that needs good attention is the definition of the objective(s) of the validation itself which implies deciding on the properties expected from the specification that are to be validated. For the same specification, the objective of the validation and consequently, the expected quality of the specification may vary depending on the objective(s) of the specification itself. Two examples to illustrate are:

Example 6.4.1. A specification may be carried out to facilitate project's cost estimation. In that case, although all important aspects of the system are to be specified, the focus would be on the functionalities of the system to be.

Example 6.4.2. The main objective of a specification may be for communication purpose. Fo example, to convince stakeholder to finance the project, or to get decision makers to vote for the project. In that case, the chief objective of the specification is communication and hence, additionally to the basic properties of a specification, properties pertaining to communication, so as readability, understandability, etc. would have to be clearly addressed.

The objective of the present validation is twofold:

- firstly, to ensure that the Object-Z specification fully describes the GRL model of the case study. The question here is:
  Question 1: ***does the Object-Z model correctly and sufficiently specify the GRL model***?

- Secondary, that the specification constitutes a solution to the initial set of problems. The question that we expect the validation to address is:
  Question 2: ***Does the Object-Z model specifies a solution to the initial set of problems?*** in other words, can a software product be obtained from the specification that resolves the initial problems.

To achieve these two validation objectives, we have decided to focus on the following properties: *(internal) consistency*, *traceability*, *correctness*, *completeness*, *applicability* and *feasibility*. Our understanding of these properties is based on two complementary perspectives that are relatively old but seem very relevant to us. The first is the IEEE Std 830-1998 [94] and the second, the work published by Boehm [27]. Our approach to validate each of these properties is explained next. Definitions 6.4.1, 6.4.2, 6.4.3, 6.4.4 and 6.4.5 were taken from the IEEE Std 830-1998 [94] document.

6.4.1 (Internal) consistency

Definition 6.4.1. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.

Definition 6.4.2. An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

If there exists at least two requirements that conflict, the specification is not internally consistent. Thus, the bulk of the validation consists in scrutinising the specification in search of at least two conflicting requirements; the purpose being to detect and eliminate all such conflicting requirements. Two techniques are used: firstly, parsing and type checking with the Community of Z Tools (CZT [117]) to eliminate as many syntactic and semantic errors as possible. Secondly, a review with cross-referencing (Boehm [27]) is performed whereby the entire specification is manually checked to identify and correct any ambiguous expression or any misconception that went through the type checker undetected; a table, diagram or a graph is constructed to analyse relationships between different elements of the specification.

From a mathematical perspective, these two techniques may not guarantee at 100% the elimination of all inconsistencies in the specification. However, *manual cross-referencing is effective for the consistency (internal, external, and traceability) and closure properties of a specification, particularly for small to medium specifications* [27]. It is also our conviction that other validation phases will bring some improvement into it. For example, the traceability analysis and animation will help detect more consistency problems.

6.4.2 Traceability

Definition 6.4.3. An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:

a Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.

b Foward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in SRS having a unique name or reference number.

Thus, items in the specification should have clear antecedents in earlier specifications or statements of system objectives. Particularly on large specifications, each item should indicate the item or items in earlier specifications from which it derives. Our approach to traceability analysis prescribes the use of review/cross-referencing techniques. The difference with applying cross-referencing to internal consistency, is that with the internal consistency analysis, we are interested in relationships between the components of the specification whereas, with the traceability, we are more concerned with the relationships between the components of the specification and the elements of some other models: higher-level model for the backward traceability and/or a lower-level model for the forward traceability.

6.4.3 Correctness

Definition 6.4.4. An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.

Each specified component must contribute directly or indirectly to the achievement of one or more initial requirements or objectives. Thus, the importance of the backward traceability to ensure that every specified requirement is clearly derived from at least one component of a higher-level model; which is in this case the GRL model. It must equally be established that every specified functionality is executable and can therefore be render operational as part of the final software product. Our approach to ensure the correctness of the Object-Z specification is based on the traceability and animation of the specification. The question one may ask is: how does traceability or animation contribute to establish the correctness of the specification?

- *Traceability*: It may be argued that definition 6.4.4 of the correctness inherently reveals the need for a traceability analysis between the specification and the software being constructed. According to the definition, `every requirement stated therein is one that the software shall meet`; this suggests the necessity to perform some mapping between the specified requirements and the functionalities of the software. However, in the case of the SRS at hand, we are more interested in the backward traceability to ensure that each component of the specification refines at least one GRL model element.

- *Animation*: We believe that animating the specification would help to establish the executability of specified requirements and hence, the correctness of the specification. The prototype would, for instance, concentrate on implementing every (or carefully selected) requirements to demonstrate their correctness.

  For each (selected) class, the state schema is implemented, with the main purpose being to prove that the components of the class appropriately describe the initial GRL model and also ensure the correctness of the constraints on variables defined in the predicate part of the state schema.

6.4.4 Completeness

Definition 6.4.5. An SRS is complete if, and only if, it includes the following elements:

a) All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.

b) Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.

c) Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

When validating this property, each of the three options a, b and c is studied individually. By examining option a), it appears that a backward traceability analysis is an appropriate means to ensure that all elements of the GRL model is well treated (or specified) in the Object-Z specification. We also suggest to execute important/significant specified components successfully mapped to important requirements. Option b) explicitly involves the definition of software outputs and realizable classes of input data. Unlike the traditional Z notation, Object-Z does not explicitly include precondition calculation and hence does not provide for a mechanism to determine valid and invalid input data. However, we have considered to perform such calculations during the specification animation. Such calculations would not require extra activities to extract operations from class schemas since such activities are already part of the animation process. Although formal specifications rarely include figures, tables and diagrams that need to be fully labelled as recommended in Option c), a manual inspection of the specification would be conducted to ensure that this recommendation is satisfied. In summary, our approach to the validation of the completeness property of the specification is summarised as followed:

- *Backward traceability* to map each important GRL model element to at least one Object-Z specification's component to ensure that those elements have been well specified.

- *Execution of selected significant operations* to validate the mapping between specification components and GRL elements obtained from the backward traceability.

- *Preconditions calculation* during the animation to determine for selected operations, realisable situations to help define software outputs both for valid and invalid input data.

- *Manual inspection* of the specification to verify that figures, tables and diagrams, if there exist, are fully labeled.

6.4.5 Applicability

The following definition is from Google < search string = "Applicability meaning" >:

Definition 6.4.6. Applicability is the usefulness of something for a particular task. When something is applicable, it is suited to something or useful for a task. The applicability of a thing refers to how useful it is in a given situation.

This work is concerned with two aspects of the applicability of the Object-Z specification: Firstly, the suitability of the Object-Z notation to specify a GRL model. The question we seek the validation to answer is the following:

Is it suitable to attempt to formalise a GRL model, especially when the model is known to be developed at an early stage of goals and requirements analysis (and includes concepts for describing non-functional requirements, whereas Object-Z does not)?

It is important to address this aspect because it involves the crucial and well-known problem of the integration of formal methods into the traditional software development process [3, 4, 147, 179]. Our validation approach to address this question is merely based on domain elements analysis: the main element being GRL models; the GRL formalisation process proposed in this work is also explored to ensure demonstrate the suitability of Object-Z to allow a GRL model as input. Animating the Object-Z specification is also believed to provide for means to strengthen our argumentation.

Secondly, the usefulness of the formal model as an intermediary step in the URN process to produce a software. Hence, the necessity to analyse the ability to build the final software

product from the formal specification, as well as, the analysis of the impact of the specification on the quality of the generated software. On this, we seek to answer through the validation, the following question:

What use is an Object-Z specification of a GRL model in the process of producing a software product?

This second aspect is rather analysed during the downwards validation phase when addressing the operational feasibility of the specification. One way to address this question, is to consider two alternatives paths that may follow the construction of a GRL model are considered: firstly, the `development of the UCM and/or further development phases from the formal specification of the GRL model`, and secondly, the `construction of the UCM diagrams directly from the initial GRL model`.

We believe that these two aspects pertaining to the applicability of the Object-Z can be assessed by performing domain analysis, reasoning about the domain and eventually animating the specification would provide for more elements of judgement. Thus, our approach to the validation of the applicability property when applied to the Object-Z specification is summarised as followed:

- Domain analysis
- Animation

6.4.6 Feasibility

Definition 6.4.7. A specification is feasible to the extent that the life-cycle benefits of the system specified exceed its life-cycle costs. Thus, feasibility involves more than verifying that a system satisfies functional and performance requirements. It also implies validating that the specified system will be sufficiently maintainable, reliable, and human-engineered to keep a positive life-cycle balance sheet (Boehm [27]).

This definition encompasses more aspects related to feasibility than those covered in this document. For example, life-cycle benefits and life-cycle costs of the software are outside the scope of this work. The operational feasibility of the specification is our concern, especially the transformation of the specification into an operational software. For instance, the operational feasibility may be motivated by the following question:

Is it possible to generate an operational software product from the specification at hand by means of existing methods/techniques?

The analysis of the operational feasibility of the formal specification under consideration is in fact a complement to the applicability of the specification. As discussed in the previous section, part of the applicability analysis aims to establish the influence of the formal specification on the quality of the final product, whereas the concern with the feasibility evaluation is to demonstrate that there effectively exist practical means (methods, techniques and tools) to generate the expected software from the specification. The adopted approach to address this property includes the following:

- Manually identify some of the existing methods, techniques and tools and show by means of arguments that they can effectively be used to transform the specification into a software product with desired characteristics.

- At this early stage of development, it may be argued that animating the specification is not enough to prove its feasibility for the reasons that, for example, the architecture of the system is still to be decided and multiple refinements of data and operations are still to be performed to obtain the real input/output data as well as the working functionalities of the system. However, a successful animation would demonstrate the ability of high-level operations to achieve the initial goals. Indicating that if the existing development methods are properly applied, the chances of producing a software that meet the initial requirements are also improved.

As it can be observed from the above planning, the bulk of the validation depends solely on the animation to establish most of the properties of the specification. Thus, the need to carefully plan the animation before its development emerges. The planning presented next is based on the prototyping process suggested by Sommerville [169], pp.409-413.

6.5 Planning the animation

A complete process of prototype development includes four phases: define the objectives of the prototype, define prototype functionalities, develop the prototype and evaluate the prototype (Figure 6.2).

6.5.1 Objectives of the animation/prototype

The main purpose of the animation is to ensure that the specification is an effective refinement of the the GRL model of the case study presented in Chapter 5. This is done by contributing to validate the following properties expected from the specification: correctness, completeness, applicability, feasibility and verifiability.

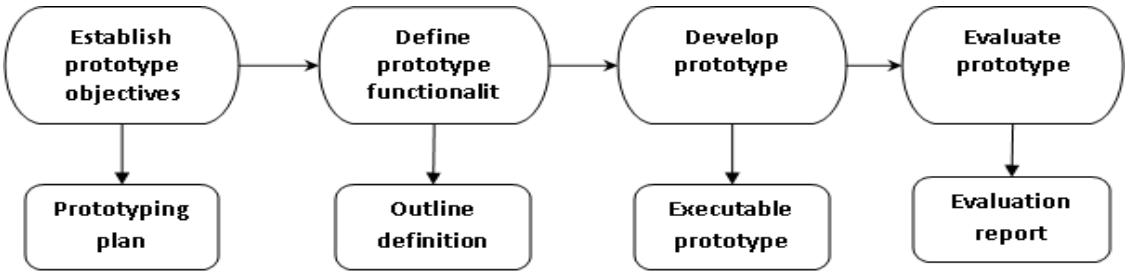

Figure 6.2: Prototyping process ([169], P.411)

6.5.2 Functionalities of the animation/prototype

An Object-Z specification is essentially composed of class schemas and definitions. Each class schema includes amongst others, a declaration part, a state schema and the operation schemas. Thus, the three categories of the specification elements implemented are: declarations/definitions, state schemas and operation schemas extracted from class schemas.

- declarations/definitions: declarations and definitions in a class schema are implemented in Prolog. Basic operations, e.g., set theoretic operations such as relation, function, union, intersection, partition, etc. also need to be implemented.

- state schemas: the two parts of each state schema (declaration and predicate parts) in a class definition are implemented in Prolog. A common clause is used to implement all state schemas: **schema_type(Variables, Name)** where **Variables** is the list of variables in the state schema and **Name** the name of the schema. For each class schema, one instance of **schema_type** is created to implement the state of the class.

- operation schemas: similarly to states schemas, each operation within a class is implemented in Prolog using the clause **schema_op(Variables, Name)** where **Variables** represents the list of input/output variables in the operation and **Name** the name of the operation.

Considering the high number of class schemas in the specification, including the templates, that in normal circonstances have to be refined and implemented, only selected ones are included in the prototype.

6.5.3 Executable of the animation/prototype

Guidelines to develop a Prolog program for a given Object-Z specification were proposed in Section 6.2. Those guidelines are followed to transform generate a Prolog program from the Object-Z specification developed in the previous chapter.

List of Prolog clauses for Object-Z classes' state schemas

Table 6.3 presents the list of Prolog's components that are to be created to implement the state schemas of Object-Z classes. The first column of the table contains Object-Z class schemas. The second column includes the Prolog clauses that need to be created to implement the state of the class schema whereas, the last column, labeled "Incl?", uses "Yes" or "No" response to indicate whether the Prolog clause is implemented as part of the prototype or not.

Object-Z	**Prolog clauses**	**Incl?**
ClsMetadata	schema_type (Variables, metadata)	Yes
ClsGrlModelElement	schema_type (Variables, modelelt)	Yes
ClsLinkableElement	schema_type(Variables, linkableelt)	Yes
ClsActor	schema_type(Variables, actor)	Yes
ClsGrlContainableElt	schema_type(Variables, containableelt)	Yes
ClsIntentionalElement	schema_type(Variables, intentionalelt)	Yes
ClsBelief	schema_type(Variables, belief)	Yes
ClsResource	schema_type(Variables, resource)	Yes
ClsTask	schema_type(Variables, task)	Yes
ClsGoal	schema_type(Variables, goal)	Yes
ClsComplementaryAction	schema_type(Variables, complementaryaction)	Yes
ClsSoftGoal	schema_type(Variables, softgoal)	Yes
ClsApplicant	schema_type(Variables, applicant)	Yes
ClsMotivator	schema_type(Variables, motivator)	No
ClsAdministrator	schema_type(Variables, administrator)	No
ClsServer	schema_type(Variables, server)	Yes
ClsIntranet	schema_type(Variables, intranet)	Yes
ClsInternet	schema_type(Variables, internet)	Yes
ClsAccessOwnApp	schema_type(Variables, accessownapp)	Yes
ClsSubmitAppOnline	schema_type(Variables, submitapponline)	Yes
ClsMeetingToMotivEval	schema_type(Variables, meetingtomotivate)	No
ClsSubmitMotivEvalOnline	schema_type(Variables, submitmotivationonline)	No
ClsReportOnSubmitedApp	schema_type(Variables, reportonsubmittedapp)	Yes
ClsGenStatsOnApp	schema_type(Variables, generatestatistics)	Yes
ClsProvideInfoGenForms	schema_type(Variables, provideinfogenforms)	No
ClsKeepPgDataInDb	schema_type(Variables, keepprogdataindatabase)	No
ClsGenFormSections	schema_type(Variables, generateformsections)	No

Object-Z	Prolog clauses	Incl?
ClsMaintSubmitedApp	schema_type(Variables, maintainsubmittedapp)	No
ClsMaintDb4SuppPg	schema_type(Variables, maintaindb4submittedapp)	No
ClsGrlCaseStudy	schema_type(Variables, grlcasestudy)	Yes

Table 6.2: List of Prolog clauses associated to Object-Z class schemas states

Implementing each state schema's clause.

The clause `schema_type([Arg1,Arg2,..,Argn], statename)` is created for a schema that has n components or variables and which name is statename. Normally, the state of an Object-Z class does not have a name however, when extracted from the class, it becomes a Z schema and hence must be given a name. Figure 6.3 shows the steps (which also represents the structure) to implement the clause. The clause `varname(Arg, varname)` associates the input argument `Arg` to the variable name `varname` of the Z state schema. This clause is particularly important for outputs formatting since the variable name is most often associated to the value contained in `Arg` for example, "age = 25". The code for the state of three classes: *ClsMetadata*, *ClsActor* and *ClsGrlContainableElement* are presented next to illustrate.

Example 6.5.1. : The class *ClsMetadata*

```
%%%%%%%%%%%%%%%%%%%%%%%%%
% State schma of ClsMetadata
%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_,name).
varname(_,value).
schema_type([Name, Val], metadata):-
%Variables and partial functions
varname(Name,name),
varname(Val, value),
%reinforcing variables' type
string(Name),
nat(Val).

% Testing the clause
%%%%%%%%%%%%%%%%%%
Amzi! Prolog  Listener
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% implementation of: schema_type([Var1,Var2,..,Varn], statename)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1-   clauses to associate input arguments to variable names
varname(_,varname1).
varname(_,varname2).
        .
        .
varname(_,varnamen).

schema_type([Arg1,Arg2,..,Argn], statename):-

2-      Associate variables to given sets to check the type of variables
        givenset(T1, type1),
        givenset(T2, type2),
                .
                .
        givenset(Tm, typem),

3-      Associate each input argument to a variable name
        varname(Arg1, varname1),
        varname(Arg2, varname2),
                .
                .
        varname(Argn, varnamen),

4-      Variables' type definition/verification
        string(Arg1),       %verify that Arg1 is a string
        nat(Arg2),          %Arg2 is a natural number >0
        Arg2 @> 0,
        element(Argn, Tm),  %Argn is of type typem
                .
                .
5-      Predicate part of the state schema
```

Figure 6.3: Prolog structure of clauses implementing state schemas

```
Amzi Prolog Listener 5.0.18h Windows
Aug 21 2000 20:19:21
Copyright (c) 1987-2000 Amzi! inc.
?- reconsult('C:\\...\\Prolog - Case study\\operationaleltch5.pro')
yes
?- schema_type([$age$,14], metadata).
yes
?-
```

Example 6.5.2. : The class *ClsActor*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State schma of the class ClsActor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, linkableelt).

schema_type(LinkableElt, actor):-

%Variables and partial functions
varname(LinkableElt, linkableelt),

schema_type(LinkableElt, linkableelt).

%Testing the clause
%%%%%%%%%%%%%%%%%%
?- schema_type([[$001$,$ImprovAppprocess$, [$Qty$, 10]],high, 5], actor).
yes
?-
```

Example 6.5.3. : The class *ClsGrlContainableElement*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State schema for the class ClsGrlContainableElement
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, linkable).
varname(_, actor).
```

```
schema_type([Linkble,Act], containable):-

    % Variables and partial functions
varname(Linkble, linkable),
varname(Act, actor),

schema_type(Linkble, linkableelt),
schema_type(Act, actor).

% Testing the clause
%%%%%%%%%%%%%%%%%
?- schema_type([[[$001$,$ImprovAppprocess$, [$Qty$, 10]],high, 5],
               [[$002$,$ImprovAppprocess$, [$Qty$, 10]],high, 5]], containable).
yes
?-
```

The implementation of the first set of classes, specifying the GRL conceptual elements, revealed the inputs for clauses implementing Object-Z classes that were obtained through multiple inheritances are progressively more complex; which implies the difficulty for the user to interact with the system. To analyse the situation with the purpose to facilitate user interaction (user interface), the hierarchical structure of those classes is presented in Figure 6.4. In our previous work (Dongmo [60], PP.164-170), we firmly argued that multiple levels of inheritance and polymorphism in a software specification have negative effects on the quality of the specification. The idea was inspired by the study conducted by Hatton [89] where the author uses a mathematical model of human reasoning and some empirical work that aims to compare defects in programs, resulting from procedural languages and the Object-Oriented programming language C++, to infer that some concepts in Object-Orientation, do not conform to the way humans think. Due to the limited size of human's short-term memory, Hatton points out that "encapsulation" only partly matches human reasoning, and that neither inheritance nor polymorphism does so. Although such a conclusion is debatable, it is worth observing that multiple hierarchical levels of inheritance and polymorphism, as well as the (large) size of objects (that may be due to encapsulation), in a software specification, may compromise the quality of the specification, especially when it comes to the readability, the understandability and the clarity. This is because multiple exchanges of information between the short-term and long-term memory is required, for example to identify inherited classes, before the correct information (on the inherited class) becomes accessible to the reader [60].

In line with this idea, van der Poll and Kotzé [177] proposed in principle #5 to refine each operation in a Z specification into a sequence of "primitives" to maintain high "cohesion" in which, a primitive manipulates at most, one state component. Conversely, a low cohesion indicates the grouping of unrelated activities.

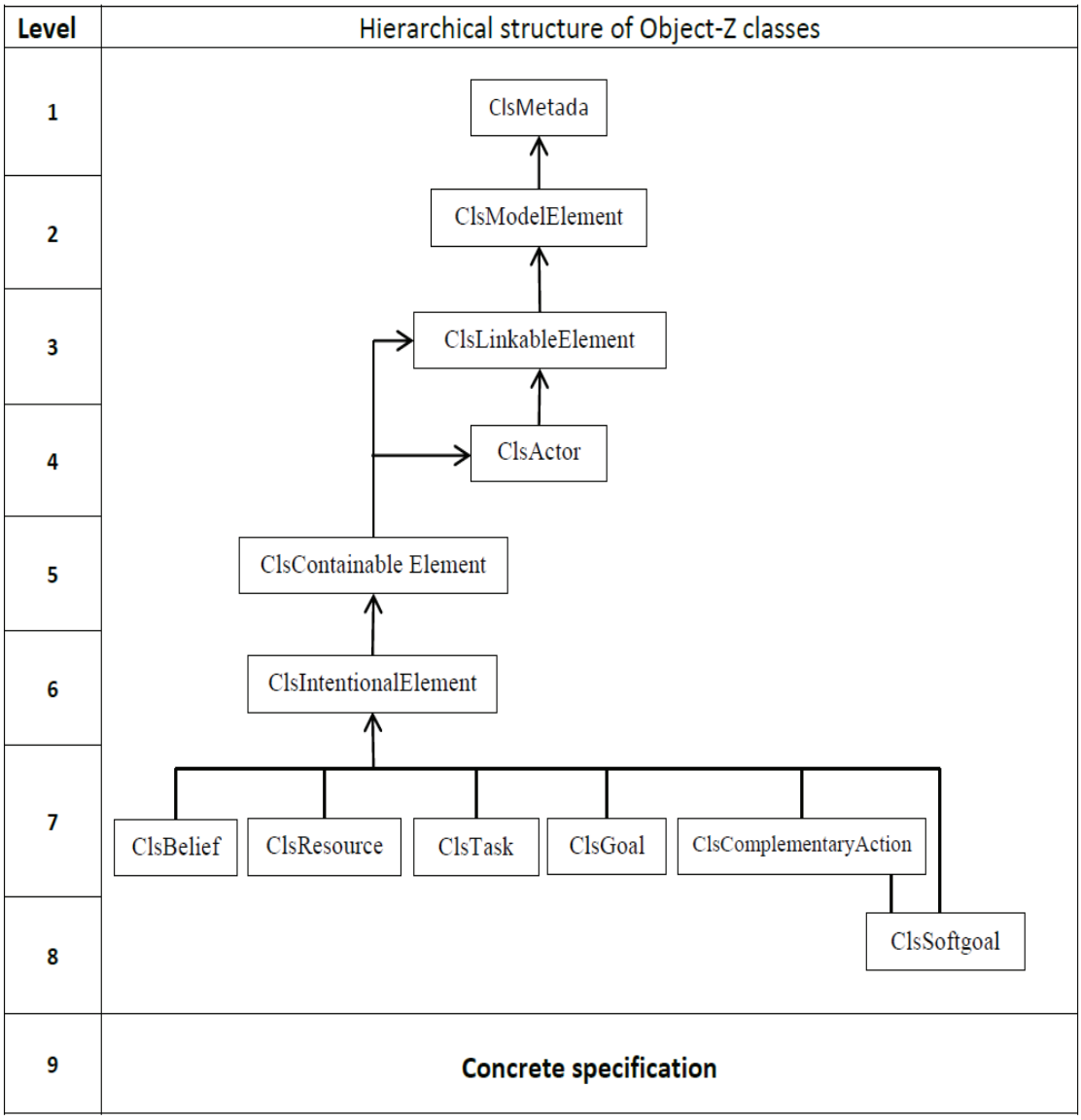

Figure 6.4: Hierarchical structure of Object-Z spec of GRL conceptual elements (Fig.4.6, p.69)

Object-Z classes	Prolog clauses' inputs
ClsMetadata	[Name1, Val]
ClsGrlModelElement	[Id, Name2, [Name1, Val]]
ClsLinkableElement	[[Id, Name2, [Name1, Val]], Imp, ImpQtive]
ClsActor	[[Id, Name2, [Name1, Val]], Imp, ImpQtive]
ClsGrlContainableElt	[[[Id, N2, [N1, Val]], Imp, IQtive], [[Id, N4, [N3, Val]], Im, ImQtive]]
ClsIntentionalElement	[[[Id, N2, [N1, Val]], Imp, ImpQtive], [[Id, N4, [N3, Val]], Imp, ImpQtive]]

Table 6.3: List of Prolog clauses associated to Object-Z class schemas states

To address the complexity of input arguments to the schema_types clauses two alternatives were examined: the first option is to re-structure the entire specification; the second alternative to simplify the complex input arguments.

Re-structuring the specification

It is possible to exploit some Object-Z concepts and operators to modify the structure of the specification with the goal to simplify user interfaces.

Example 6.5.4. One can merge the inherited class (the superclass) into the inheriting class (subclass) by merging state variables and conjoining other features of the two classes. By doing so, a variable that appears in both schemas with the same name and type is introduced only once in the subclass hence, reducing the total number of variables in the resulting class.

Example 6.5.5. If a class B inherits the class A to extend operation opa defined in A with opb defined in B, such inheritance can be avoided since the dot operator ($\bullet$) combined with sequential operation composition ($\fatsemi$) can be used elsewhere (e.g., in an interface definition) to define the composite operation needed:

$$op \mathrel{\widehat=} [x? : A, y? : B \mid ..] \bullet x?.opa \fatsemi y?.opb.$$

These two examples show that the structure of a specification can be modified to reduce the number of components in each class schema. Knowing that the complexity of the user interface depends on multiple factors including the number and the nature of the input objets, we believe the implementation of the specification obtained in the above examples will have an improve user interface. However, the price for such improvement can be enormous. A few cases are enumerated next.

Considering the fact that the structure of the Object-Z specification, at hand, is based on that of the GRL conceptual model, modifying the structure of the specification without

doing the same with the GRL conceptual model, renders the traceability property of the specification more difficult to establish. Each class schema A in the specification formalises a specific GRL conceptual model and can therefore be traced with less effort. However, merging a superclass A with its subclass (B), as in example 6.5.3, would make it more difficult to trace the resulting class from the original GRL model. Another important aspect to consider is the loss of modularity in the merged class that may equally render the specification difficult to modify. Since inheritance in Object-Z provides for a means to represent the architectural structure of the system, eliminating an inheritance implies the destruction of this architectural information.

Based on this brief analysis, we have chosen to find a mean to simply the structure of the input arguments instead of modifying directly the structure of the specification.

Simplifying the input arguments to schema_types

To simplify the structure of the input arguments to each schema_type, the main idea is to create an additional argument that serves as reference or unique identifier for the other arguments. Instead of passing the entire set of variables of the inherited class as arguments to the clause implementing the inheriting (subclass), only the reference is passed. This approach reduces the structure of the input argument of any schema_type to only two arguments:

1. A unique reference
2. A simple list of arguments containing the components of the state of the class and variables containing each the identifier/reference of an object of each inherited class.

The modified structure of the Prolog clause is therefore the following:

$$schema\_type([Ref, [Ref1, .., Refn, Var1, .., Varn]], statename)$$

The use of a reference in the list of arguments to replace the full set of inherited components comes with a number of advantages (e.g., the simplification of the user interface, the modularisation of the system and more convenient approach to implement the concept of object in Object-Z) but consequently requires more programming work. A reference in this case, effectively implements the concept of instantiation as it is conceptualised in Object-Z; an object of a class is in fact manipulated only through a reference to an instance of that class (Duke et al. [69]). This implementation approach works perfectly for all inheritances defined by aggregation and considers those defined by inclusion through the set of objects that it specifies.

The disadvantage of this approach is to require more programming work. Additional operations need to be developed to allow access to data of an inherited object through the

reference to the object; this is also how it works with Object-Z. As stated in by Graeme Smith,

> *The only way an object can be changed is via the application of one of its classs operations using the dot notation (Smith [163]).*

The dot operator (•) uses the identifier (reference) of an object to access the methods and other features of the class. The creation of additional operations to link each unique reference to a set of data that may be consulted to ensure the validity of the inherited components.

Object-Z class	**Prolog clauses' inputs**
ClsMetadata	schema_type([RefMd,[Name, Val]], metadata)
ClsGrlModelElement	schema_type([RefMe, [RefMd, Id, Name]], modelelt)
ClsLinkableElement	schema_type([RefLe, [RefMe, Imp, ImpQtive]], linkableelt)
ClsActor	schema_type([RefA, RefLe], actor)
ClsGrlContainableElt	schema_type([RefCe, [RefLe, RefA]], containablelelt)
ClsIntentionalElement	schema_type([RefIe, RefCe], intentionalelt)

Table 6.4: List of states' schemas' clauses using OZ objects' references

Example 6.5.6. Implementing the state of the class *ClsMetadata*.

```
%%%%%%%%%%%%%%%%%%%%%%%%%
% State schma of ClsMetadata
%%%%%%%%%%%%%%%%%%%%%%%%%

varname(_,refmetadata).
varname(_,name).
varname(_,value).

schema_type([Ref, [Name, Val]], metadata):-

% Variables and partial functions
varname(Ref, refmetadata),
varname(Name,name),
varname(Val, value),

string(Ref),
```

```
string(Name),

save_stateref(stateref([Ref,[Name, Val]])),

reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([Ref, [Name, Val]], metadata)

Amzi! Prolog (IDE Only)
Licensed to
Free Version

Interpreting project: phdproject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: 'commonclauses.pro', 'ozprologclauses.pro', 'staterefs.pro'
Type 'quit.' to end and [Ctrl]-C to stop user input.

?- schema_type([$md001$,[$Size$,23]], metadata).
yes
?- stateref([md001, LinkedData]).
LinkedData = [`Size`, 23] ;
no
?-
```

Object-Z	**Prolog**	**Incl?**
ozImprovAppProc	ClsApplicant	Yes
ozFlexibleProcess	ClsApplicant	Yes
ozImprovMotivEvalProc	ClsMotivator	No
ozCombineMotivEval	ClsMotivator	No
ozResolvEmailPb	ClsAdministrator	No
ozImprovEvalMotivTime	ClsAdministrator	No
ozFacilAppManagement	ClsAdministrator	Yes
ozReduceAppTime	ClsApplicant	Yes
ozAppOnline	ClsApplicant	Yes
ozColabWithAdmin	ClsMotivator	No
ozMotivEvalOnline	ClsMotivator	No

Object-Z	Prolog	Incl?
ozHaveConsistForms	ClsAdministrator	No
ozInstantAccessToPgData	ClsAdministrator	No
ozColabHRSys	ClsServer	No
ozGenSecBelief		No

Table 6.5: List of Object-Z components to be animated

6.6 The rightward validation phase

As mentioned earlier in Section 6.4.1, this validation phase aims to eliminate language related errors and ensure that the specification is internally consistent and well-formed. Two techniques are used: firstly, an automatic parsing and type checking of the specification with the Community of Z Tools (CZT) and secondly, the (manual) review of the specification with cross-referencing.

6.6.1 Type checking the Z/Object-Z specification of the case study

The Object-Z specification of the case study being validated is presented in Appendix C. The specification was initially constructed in a Latex document using the OZ.STY macro (Allen [7]) and later type-checked with the Community of Z Tools CZT version 1.5.0 the result of which indicated 220 errors found. The full list of these errors can be consulted in Appendix D.0.2. The errors detected by the type checker are of two categories: the first

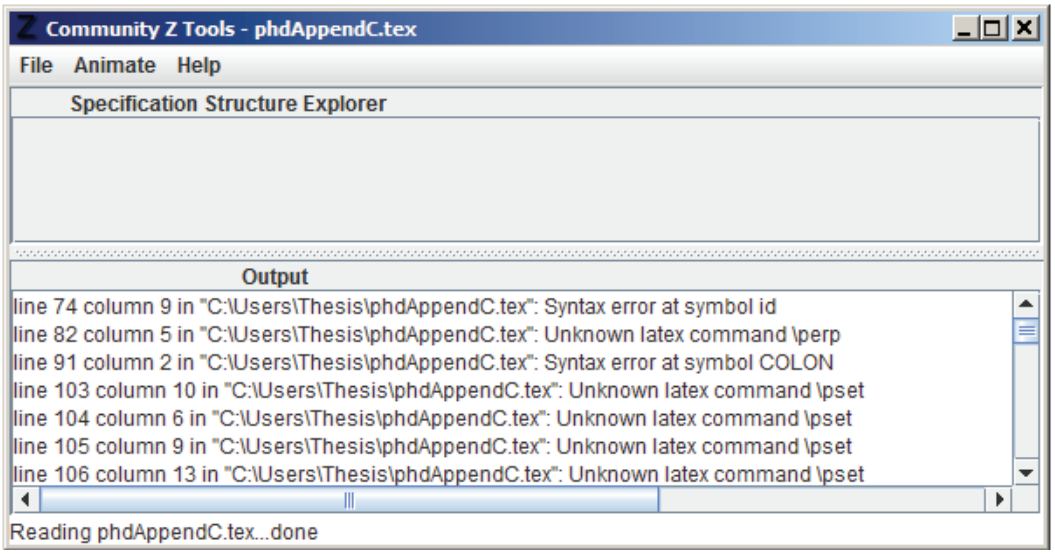

Figure 6.5: Type checking the specification with CZT

category includes errors due to incompatibility between OZ.Sty and CZT.Sty, reported as "Unknown commands" and the second category includes syntax errors. A total of 197 errors, representing about 90% of errors detected were due to oz commands (oz.sty) misunderstood by the czt parser. The czt tool reported them as *Unknown latex command* followed by the OZ command that couldn't be parsed or checked.

Dealing with unknown command errors

To correct each of such errors, we simply replaced the OZ command with the corresponding CZT's one. The complete list of such replacements is presented in Table 6.6

No.	OZ command	CZT command
1	\pset (\power)	\power
2	\ST	\where
3	\fovr	\oplus
4	\all (\forall)	\forall
5	\bbar and \cbar	| or \mid
6	\map	\mapsto
7	\exi (\exists)	\exists
8	\exione	\exists_1
9	\perp (\bot)	%%Zchar \perp U+266D
10	\uni	\cup
11	\subs	\subseteq
12	\prod	\cross
13	\zimp	\implies
14	\defs	= =
15	\int (\inter)	\cap
16	\seqone	\seq_1
17	\	$\sim$

Table 6.6: Mapping between OZ and CZT commands

Dealing with Syntax and other errors

This category of errors was corrected by slightly modifying the command or by changing the variable's name. For example, oz allows the arguments of the operator \Delta() to be parenthesised whereas czt does not. We simply removed the parentheses to get it work for both oz and czt. The use of the word id as a variable or as the prefix in a variable's name was also the source of a couple of syntax errors, we simply used a different word (ref) to replace id.

Status of the specification after parsing and typechecking.

Out of the 220 errors, 215 were corrected and the 5 presented next remained.

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Thesis>java -jar czt.jar ozedtypecheck phdAppendC.tex
ParseException errors for oz
line 283 column 24 dialect oz in "phdAppendC.tex": Syntax error at symbol BAR
line 312 column 80 dialect oz in "phdAppendC.tex": Syntax error at symbol RPAREN
line 695 column 0 dialect oz in "phdAppendC.tex": Unknown latex command \begin
line 699 column 0 dialect oz in "phdAppendC.tex": Unknown latex command \end
line 695 column 7 dialect oz in "phdAppendC.tex": Syntax error at symbol
                                                  genschemaAddActorref?actors
C:\Users\Thesis>
```

We couldn't understand the reason for the first two errors since the two oz latex commands to which they refer are indeed correct from standard Z and Object-Z perspectives. These two errors could arguably be due to an abnormal behaviour of the parser.

It is also very plausible that the last three errors were generated because the CZT parser or typechecker could not recognise the following generic schema definition or translate it into unicode; we own this idea to Miller who is quoted below the schema.

```
\begin{genschema}{AddActor}{ref}
.
.
\end{genschema}
```

> *LATEX \begin{xxx} and \end{xxx} environments cannot be defined using LATEX markup directives. If a Z extension needs to provide new LATEX envi-*

ronments, the LATEX to Unicode converter needs to be adapted directly (Miller et al. [126]).

Thus, the three errors could have been corrected by manually associating some sequence of unicode to the generic schema constructs directly in the prelude part of the specification. Such adaptation was not carried out because the construct is known to be correct (from standard Z and Object-Z perspective) at its current stage and further utilisation of czt was not envisaged, for example to animate the specification. However, despite the above errors, the usefulness of the validation remains. A successful type checking ensures that the input specification is well-formed that is, the definitions and class schemas, as well as their components, including the operations and variables, are well-defined. This is particularly important, for example, in an environment where the generation of the OZ specification is automated, with no graphical representation, and is meant to serve as input to another tool.

6.6.2 The review of the specification

An inspection of the specification in Appendix C was performed at two levels: firstly, at the specification level by analysing the inter-relationships between class schemas to evaluate the complexity of the overall specification. Secondary, each individual class schema was inspected to evaluate its complexity, detect contradicting elements within a class, check that each class is well-structured with respect to a manageable size, especially the size of the schema, and check that the formulas in the predicate parts of the state and operation schemas within a class are intuitively clear to an average reader.

The inspection of the entire specification

Figure 6.4, presented on page 142 is the result of the overall inspection of the specification in Appendix C. This particular figure is important because it represents the common part of an hierarchical structure of any Object-Z specification of a GRL model generated using our transformation framework and associated algorithms proposed in Chapter 4 and 5. This part is common to all generated specifications because the class schemas and objects of such a specification are obtained by inheritance and/or instantiation of class schemas at level 7 and 8. Thus, the specific structure of each particular specification (representation of the concrete GRL model) would start from level 8 and/or level 9.

The main advantage of the figure is that it illustrates a formal representation of the GRL conceptual model and its structure and is therefore suited to formally analyse the conceptual model. Considering our position regarding the adverse effects of multiple hierarchical inheritance and polymorphism on the quality of an Object-Z specification (see discussion

above, on page 142 or refer directly to: Dongmo [60], PP.164-170), further analysis of the structure in Figure 6.4 (with or without the modification of the GRL conceptual model) purposing to optimise the structure with the goal to improve on the quality of the specification should be envisaged. However, such analysis is outside the scope of this work and is therefore recommended for further research.

The inspection of individual class schemas

Each class schema of the Object-Z specification, in Appendix C was visually examined with the average schema' size less than half a page. Schemas were intentionally kept simple and only essential operations were specified for each class, for the reason that the chief purpose was to illustrate our proposed GRL model transformation framework. In practice, the situation would have been slightly different because all essential static and behavioral information would have been specified. Contradictory components or formulas couldn't be found within each class schema and most of the predicates were simplified enough for an average reader to be able to comprehend.

However, it is important to observe that the use of the polymorphic data type notation in the following two classes: *ClsIntentionalElement* and *ClsGrlCaseStudy*, keeps the specification simple to read but, covers a complex data type for which developers need to be aware of.

Example 6.6.1. Consider the following declaration from the class *ClsIntentionalElement*:

$$\forall\, elt : \downarrow ClsIntentionalElement$$

An object identifier *elt*, as defined in this expression, can be associated to any object of the class *ClsIntentionalElement* or any object of any other class that inherits the class *clsIntentionalElement*. Additionally, for *elt* to access a property or method of its class, the property or method must be listed in the visibility list of the class thus, the need to readily prepare all subclasses of the superclass for such access.

The benefit of insisting on this particular aspect of the specification is not to criticise the notation but, to make designers aware of the complex data structure that the notation hides. For we firmly believe that the knowledge of the level of complexity of the specification can positively contribute to a successful planning of further development phases, including: the choice of techniques, methods, and/or tools.

Example 6.6.2. At the implementation phase, planning the instructions that manipulate variables of type Integer, String or similar is inherently simpler than to plan instructions to manipulate structure of arrays or pointers to complex data structures.

The diagram in Figure 6.4 shows at the hierarchical level 7 that all class templates, from which other class schemas and objects composing the specification are generated, are subclasses of the class *ClsIntentionalElement*; meaning almost the entire specification is concerned by the polymorphic definition in *ClsIntentionalElement*. This further illustrates the notational power of a polymorphic definition but also appeals for awareness since subclasses do not implement any mechanism to indicate to the reader that the inherited class has such a complex definition.

6.7 Upward validation phase

This phase is mainly about ensuring that the specification at hand satisfies the stakeholders goals and initial requirements. To perform this validation, a twofold approach is adopted: firstly, each specification component is mapped to at least one initial requirement or stakeholder's goal to ensure that the component addresses the need(s) of stakeholder. Secondary, a correctness proof is carried out to ensure that each component correctly responds to the stakeholder wish or correctly specifies the requirement to which it is associated.

6.7.1 Requirements traceability

Considering the fact that the initial requirements comprise two distinct type of models, the GRL conceptual model and the GRL model for the case study, two different traceability analysis are conducted. The first analysis aims to trace Object-Z's classes, namely templates from the GRL conceptual model whereas, the second trace Object-Z components from the GRL model for the case study.

Tracing Object-Z specification templates from the GRL conceptual model

Table 6.7 represents the traceability table mapping Object-Z class schemas to GRL conceptual model's elements. The GRL elements are from the GRL conceptual model presented in Chapter 4 (see Figure 4.6, page 69). As discussed in Chapter 4, elements pertaining to GRL evaluation are not in the scope of this work. The Object-Z classes are also presented in Chapter 4, Section 4.3.

	Object-Z template	GRL Conceptual model elements														
No.	**OZ classes**	**Metadata**	**Model elt**	**Linkable elt**	**Actor**	**Containable elt**	**Intentional elt**	**Belief**	**Resource**	**Task**	**Goal**	**Softgoal**	**Element link**	**Dependency**	**Decomposition**	**Contribution**
01	ClsMetadata	x														

	Object-Z template	GRL Conceptual model elements														
No.	**OZ classes**	**Metadata**	**Model elt**	**Linkable elt**	**Actor**	**Containable elt**	**Intentional elt**	**Belief**	**Resource**	**Task**	**Goal**	**Softgoal**	**Element link**	**Dependency**	**Decomposition**	**Contribution**
02	ClsGRLModelElement	i	x													
03	ClsLinkableElement	i	i	x												
04	ClsActor	i	i	i	x											
05	ClsContainableElement	i	i	i	i	x										
06	ClsIntentionalElement	i	i	i	i	i	x									
07	ClsBelief	i	i	i	i	i	i	x								
08	ClsResource	i	i	i	i	i	i		x							
09	ClsTask	i	i	i	i	i	i			x						
10	ClsGoal	i	i	i	i	i	i				x					
11	ClsSoftGoal	i	i	i	i	i	i					x				
12	ClsElementLink	i	i	i									x			
13	ClsDependency	i	i	i									i	x		
14	ClsDecomposition	i	i	i									i		x	
15	ClsContribution	i	i	i									i			x

Table 6.7: Tracing Object-Z elements from GRL Conceptual model

The letter x in a cell (intersection between a line and a column of the table) indicates that the Object-Z element in the line, formally specifies the GRL element in the column. Whereas, the letter i is used to indicate that the Object-Z element indirectly addresses the GRL element by inheriting another Object-Z class that either directly (x) or indirectly (i) specifies the conceptual element in the column.

Table 6.7 shows that each Object-Z class in the specification either specifies directly a GRL element, or indirectly contributes to the formalisation of such an element. It is also worth noting that, in the table, each GRL element is mapped to at least one formal specification component. Since mapping a class schema to a GRL element does not guarantee that the class correctly specifies the element, the next step in this validation path is therefore, to proof the correctness of the formal specification; such a proof is presented in Section 6.7.2.

Tracing Object-Z specification from the GRL model of the case study

Table 6.8 shows the mapping between the elements of the GRL model of the case study (see fig.5.2, p.106) and the components of its Object-Z specification (see Appendix C).

	Object-Z Elements	GRL model elements for the case study																																
No.	OZ classes	**Applicant**	ImprovAppProc	FlexibleProcess	ReduceAppTime	AppOnline	AccessOwnApp	SubmitAppOnline	**Motivator**	ImprovMotivEvalProc	CombineMotivEval	ColabWithAdmin	MotivEvalOnline	MeetingsToMotivEval	SubmitMotivEvalOnline	ReportOnSubmitedApp	**Administrator**	ResolvEmailPb	ImprovEvalMotivTime	FacilAppManagement	HaveConsistForms	InstantAccessToPgData	GenStatsOnApp	ProvideInfoGenForms	KeepPgDataInDb	**Server**	ColabHRSys	GenFormSections	MaintSubmitedApp	MaintDb4SuppPg	GenSecBelief	Intranet
01	**ClsApplicant**	x																																
02	ozImprovAppProc	c	x																															
03	ozFlexibleProcess	c		x																														
04	ozReduceAppTime	c			x																													
05	ozAppOnline	c				x																												
06	ClsAccessOwnApp	c					x																											
07	ClsSubmitAppOnline	c						x																										
08	**ClsMotivator**								x																									
09	ozImprovMotivEvalProc								c	x																								
10	ozCombineMotivEval								c		x																							
11	ozColabWithAdmin								c			x																						
12	ozMotivEvalOnline								c				x																					
13	ClsMeetingToMotivEval								c					x																				
14	ClsSubmitMotivEvalOnline								c						x																			
15	ClsReportOnSubmitedApp								c							x																		
16	**ClsAdministrator**																x																	
17	ozResolvEmailPb																c	x																
18	ozImprovEvalMotivTime																c		x															
19	ozFacilAppManagement																c			x														
20	ozHaveConsistForms																c				x													
21	ozInstantAccessToPgData																c					x												
22	ClsGenStatsOnApp																c						x											
23	ClsProvideInfoGenForms																c							x										
24	ClsKeepPgDataInDb																c								x									
25	**ClsServer**																									x								
26	ozColabHRSys																									c	x							
27	ClsGenFormSections																									c		x						
28	ClsMaintSubmitedApp																									c			x					
29	ClsMaintDb4SuppPg																									c				x				
30	ozGenSecBelief																									c					x			
31	ClsIntranet																																	x
32	ClsInternet																																	

Table 6.8: Tracing Object-Z elements from GRL concrete model

Object-Z components are represented in lines whereas, GRL elements are in columns of the table. Object-Z components are of two types: class schemas and objects which are instances of the class schemas. A cross (x) in a cell (intersection between a line and column) of the table indicates that the Object-Z component in the line is the main component used to specify the GRL element in the intersecting column. The letter 'c' in a table's cell indicates that the Object-Z component in the line, additionally to specifying another GRL element, also contributes to the specification of the element in the intersecting column. Elements in lines and columns were grouped per actor's definition and the specification of each element in an actor's definition naturally contributes to the specification of the actor.

With tables 6.7 and 6.8, we have shown that the Object-Z specification developed in Chapter 4 and 5 formally describe, respectly, the GRL conceptual model and the GRL model of the case study. It is therefore important to establish that the Object-Z model correctly specify the two GRL models.

6.7.2 Establishing the correctness of the Object-Z specification

Considering the wish of the the stakeholder(s) to get the GRL model in fig.5.2 formally specified, from Definition 6.4.4, it is inferred that the Object-Z specification under consideration is correct if and only if every component of the specification satisfactorily specifies a part of the GRL model of the case study and that the entire model is fully specified.

The Object-Z specification fully describes the GRL model

The backward traceability analysis performed in Section 6.7.1 readily proves that every element of the GRL model of the case study is specified by at least one component of the Object-Z specification. In Table 6.8 each element of the GRL model is associated to at least one component of the formal specification indicating that the element was indeed specified. The completeness of the specification can also be established by analysing the specification process.

To specify the input GRL model, all the elements of the model are first identified, in Section 5.4.2 and reported in Tables 5.4 and 5.5. Thereafter, the formal specification of those elements is carefully planned using Table 5.6, in which for each GRL element, the associated Object-Z object(s) needed to formalise the element is indicated and finally, the planned objects are created. Thus, when the formalisation is terminated, it is guaranteed that the entire GRL model is fully covered.

By means of specification animation, we show that Object-Z specification satisfies the input GRL model.

Approach to analyse the satisfiability of the Object-Z specification

The task to address the correctness of a software specification or the extend to which a specification satisfies the initial requirements is complex in nature. This is mainly due to the fact that the initial requirements against which the validation relies on is generally informal, subjective, and not always fully documented. Thus, the need to define before hand the scope of the validation that is for example decide which specific aspect(s) of the requirement(s) to focus on.

In this work, the analysis of the satisfiability of a GRL model by its formal specification is limited to the following definition:

Definition 6.7.1. A formal specification that fully specifies a GRL model is said to be satisfactory if:

(1) internal relationships between GRL elements are preserved in the formal specification,

(2) the properties of each individual GRL element are traceable in the formal specification.

We also believe that the formal specification of a GRL model should preserve the semantic of each formalised element. However, this aspect is not explicitly addressed at this level of the analysis because, similarly to GRL model elements that own their semantic to GRL conceptual elements, the formal specification too owns its semantic to the Object-Z specification templates that specifies the GRL conceptual model.

A straightforward approach to establish the correctness of the Object-Z specification is used to justify that the specification explicitly integrates the relationships between GRL elements and preserves the properties of each formalised element.

The formal specification of the relationships between GRL elements

The formalisation process readily includes internal relationships between GRL elements. Step 2 of Algorithm 1 provides for an operation to traverse the input GRL model to identify all the link elements describing such relationships; the complete list of identified links is shown in Table 5.5. Depending on the type of the link, the formal specification is obtained by instantiating one of the following templates: *ClsDependency*, *ClsDecomposition* or *ClsContribution*; the reference to the object hence created is kept in the component *alllinks*

defined in class *ClsGrlCaseStudy* by means of the operation *Newlink* that adds the object to the component.

Animating the class *ClsGrlCaseStudy* focusing on the link elements provides for a means to reveal the properties of a link element to a non-expert stakeholder and ensure the operation *Newlink* does exactly what it is expected to do and that in the end, the objects listed in Table 5.5 are created and kept in the system. However, it would be more convenient to examine the structure of every component of the specification to ensure that the properties of the formalised model are well-preserved before proceeding with the animation which main objective is to convince the external world.

Examining the specification process and/or the structure of class schemas

It is important to notice that the graphical representation of a GRL model, like that of the case study represented in Figure 5.2, reveals only two types of information on its elements: the type of the element, graphically revealed by the shape of the object representing the element, and the relationship with other elements represented by annotated links elements. In the preceding section, we have shown that relationships between GRL elements are being formalised. In the same vein, it can be established that the type of each formalised object can be traced from the specification.

There are different ways to trace the type of a GRL model element in its Object-Z specification. One alternative is to consider the template from which an Object-Z class schema or an object was derived either by inheritance or by instantiation. Each component of the specification is either a template, a class schema generated from a template by inheritance, or an object derived from a template by instantiation. Recalling from Chapter 4 that each template is an Object-Z class schema designed for a specific type of GRL element, determining the type of an initial GRL element becomes straightfoward. The traceability analysis in Table 6.8, associates to each input GRL element an Object-Z component that formalised it. Table 5.6 on page 110 indicates for every Object-Z element, the template used to create the element.

Example 6.7.1. In further development phases, when manipulating, for example, *ozFlexibleProcess*, it is important to know the type of the GRL element that this component specifies to decide on further actions to take. For example, a softgoal modeling a non-functional requirement will continue to stimulate the creation of complementary non-functional actions (CNF-actions), discussed earlier in Chapter 4, Section 4.2.3 on page 62, during the complete cycle of the software development system (see SDS in Figure 4.3, p.61).

As indicated in Table 6.8, the template used to create the object *ozFlexibleProcess* is *ClsSoftGoal* thus, we deduce without accessing the GRL element itself that *ozFlexibleProcess* specifies a softgoal. In practice, when implementing our proposed GRL formalisation framework with the associated algorithms, one may not avoid implementing a formal mapping between Object-Z templates and GRL conceptual elements to allow a proper selection of templates during the specification process.

The type of a GRL element being specified or some other useful information, can equally be determined by examining the the structure (specifically) the state of some class schemas.

Example 6.7.2. The unique identifier (same one that identifies the GRL element) of any actor's class is kept in the component *actors* of the class *ClsGrlCaseStudy*. So, the identifier of a class schema being kept in that component readily indicates that the class specifies a GRL actor's definition.

Example 6.7.3. Each class schema specifying an actor's definition includes the following components to kept the references to objects formalising GRL elements contained in the GRL actor being specified:

- *softgoals* to keep the references to objects specifying softgoals
- *goals* to keep the references to objects specifying goals
- *beliefs* to keep the references to objects specifying beliefs
- *nfdependers* to keep references to all objects specifying softgoals that depend on the actor being specified.

Other properties of GRL elements that are not represented on the final model are either inherited from the conceptual elements (e.g., metadata) or available to the URN development tool (jUCMNav) to aid with the construction and the analysis of the model (e.g., size, position, line color, etc). Although this work, at its present stage, is focused on the final version of GRL model, its main purpose is to lay the foundation for an interactive environment whereby for instance, the flexibility and usability of the URN graphical environment is exploited to guide the construction of the formal specification of the system.

Example 6.7.4. Consider the component $softgoals : \mathbb{P}\, ClsSof$, defined as a set of softgoals, to keep all the softgoals contained in an actor's definition, and the operation *NewSoftGoal* to update the component *softgoals*. In the final version of the GRL model, the list of softgoals in each actor's definition is known. So, this design decision that provides for the possibility to update the list is justified when the specification considers the GRL model in its construction phase.

Next, such constructive operations are animated using Prolog to show that when the construction is terminated, the various components of the specification are left in the correct state. For instance, the variable *softgoals* in the class *ClsApplicant* must contain the two softgoals: *ozImprovProc* and *ozFlexibleProcess* in the end.

Approach to Prolog animation of the Object-Z operations

Our approach to animate an Object-Z specification is presented in Section 6.2. The approach is used in Section 6.5.3 to implement state of the class schemas. The generic clause proposed to implement operation schemas is:

```
schema_op([Arg1,Arg2,..,Argn], operationname)
```

where `Arg1,Arg2,..,Argn` is the list of n components or variables from the z operation schema, namely, `operationname`. Figure 6.6 shows the steps (which also represents the structure) to implement the clause. As in the case of state schemas, the clause `varname(Arg, varname)` associates the input argument `Arg` to the variable name `varname` of the Z operation schema.

Animating the class *ClsActorApplicant*

By animating the class *ClsApplicant*, we seek to establish the correctness of the state of the class before and after any of its operations is performed, as well as the correctness of the operations defined within the class.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State schema for the class ClsApplicant
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refapplicant).
varname(_, applicantid).
varname(_, sofgoals).
varname(_, goals).
varname(_, nfdependers).

schema_type([RefAp, [Appid, RefAc, SoftGs, Goals, Nfds]], applicant):-

%Variables and partial functions
varname(RefAp, refapplicant),
varname(Appid, applicantid),
varname(SoftGs, softgoals),
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% implementation of: schema_op([Var1,Var2,..,Varn], inout(opname))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1-  Predicates

2-  clauses to associate input arguments to variable names
varname(_,varname1).
varname(_,varname2).
        .
        .
varname(_,varnamen).

schema_op([Arg1,Arg2,..,Argn], inout(opname)):-

3-      Associate variables to given sets to check the type of variables
        givenset(T1, type1),
        givenset(T2, type2),
                .
                .
        givenset(Tm, typem),

4-      Associate each input argument to a variable name
        varname(Arg1, varname1),
        varname(Arg2, varname2),
                .
                .
        varname(Argn, varnamen),

5-      Variables' type definition/verification
        string(Arg1),      %verify that Arg1 is a string
        nat(Arg2),         %Arg2 is a natural number >0
        Arg2 @> 0,
        element(Argn, Tm),  %Argn is of type typem
                .
                .
6-       Predicate part of the state schema

7-       Final outputs
```

Figure 6.6: Prolog structure of clauses implementing operation schemas

```
varname(Goals, goals),
varname(Nfds, nfdependers),

stateref([RefAc, X]),
%Process RefAc and X

save_stateref(stateref([RefAp, [Appid, RefAc, SoftGs, Goals, Nfds]])),
reconsult(staterefs).
```

The constraint on non-functional dependents softgoals (*nfdependers*), that depend on the actor's definition,

$$\forall\, nfelt \in nfdependers \bullet nfelt.actor.id \neq id$$

was not reinforced to keep the implementation simple but also because this implementation does not explicitly reveal inherited properties. Examples of hidden information include the followings:

- Inherited from *ClsLinkableElement*: *importance_type* and *qualitative*
- inherited from *ClsModelElement*: *element_identifier*, *element_name*,
- inherited form *ClsMetaData*: *metadata_name*, *metadata_value*.

Thus, the need for a predicate to access those information and make them accessible. The predicate `data()` is defined to search inherited state schemas and make their components data directly accessible.

```
data(RefMd, Mdata, metadata):-
stateref([RefMd,Mdata]).

data(RefMe, [], Medata, modelelt):-
stateref([RefMe, Medata]).
data(RefMe, Mdata, [RefMd|Medata], modelelt):-
stateref([RefMe, [RefMd|Medata]]),
data(RefMd, Mdata, metadata).

data(RefLe,[],[],Ledata, linkableelt):-
stateref([RefLe, Ledata]).
data(RefLe,Mdata,Medata,Ledata,linkableelt):-
stateref([RefLe,Ledata]),
```

```
element(RefMe, Ledata),
data(RefMe,Mdata,Medata, modelelt).

data(RefAc,Mdata,Medata,Ledata,actor):-
stateref([RefAc,RefLe]),
data(RefLe,Mdata,Medata,Ledata,linkableelt).

data(RefCe, Mdata, Medata, Ledata, containableelt):-
stateref([RefCe,[RefLe, RefAc]]),
data(RefLe, Mdata, Medata, Ledata, linkableelt).

data(RefIe, Mdata, Medata, Ledata, intentionalelt):-
stateref([RefIe, RefCe]),
data(RefCe, Mdata, Medata, Ledata, containableelt).
```

The operation schema *NewSoftGoal* is implemented next.

NewSoftGoal

$\Delta(softgoals)$
$sgoal? : ClsSoftGoal$

$softgoals' = softgoals \oplus \{sgoal?\}$

The only component of the class *ClsActorApplicant* that is updated by this operation is *softgoals*. The declaration of the input component $sgoal? : ClsSoftGoal$ implicitly encompasses the operation to create an object of the class *ClsSoftgoal*. The reference to the newly created object is added to the list *softgoals* of all softgoals contained in the class.

Creating a new softgoal ($sgoal? : ClsSoftGoal$)

An instance of the class *ClsSoftGoal* inherits data from the following classes: *CLsMetaData*, *ClsModelElement*, *ClsLinkableElement*, *ClsContainableElement*, *ClsIntentionalElement* and *ClsComplementaryAction* (see Figure 6.4). The last class can be omitted when the object is newly created since the list of CFN-actions for a softgoal is continuously updated during the entire software development process after its creation.

Inherited class	**Property**	**ozImprovAppProc**	**ozFlexibleProcess**
ClsMetadata	name	maxtime	size
	value	2	36
	state ref.	'md001'	'md002'
ClsModelElement	identifier	id001	id002
	Name	ozImprovAppProc	ozFlexibleprocess
	state Refs	[me001, md001]	[me002, md002]

Inherited class	Property	ozImprovAppProc	ozFlexibleProcess
ClsLinkableElement	importance	High	Medium
	qualitative	95	80
	state Refs	[le001, me001]	[le002, me002]
ClsActor	state Refs	[ac001, le001]	[ac002, le002]
ClsContainableElement	state refs	[ce001,[le001, ac001]]	[ce002,[le002, ac002]]
ClsIntentionalElement	state refs	[ie001, ce001]	[ie002, ce002]

Table 6.9: Data for the two softgoals of the actor applicant

```
Licensed to
Free Version

Interpreting project: phdproject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: 'commonclauses.pro', 'ozprologclauses.pro', 'staterefs.pro'
Type 'quit.' to end and [Ctrl]-C to stop user input.

?- schema_type([$md001$, [$maxtime$, 2]], metadata).
yes
?- schema_type([$md002$, [$size$, 36]], metadata).
yes
?- schema_type([$me001$, [$md001$, $id001$, ozimprovappproc]], modelelt).
yes
?- schema_type([$me002$, [$md002$, $id002$, ozflexibleprocess]], modelelt).
yes
?- schema_type([le001, [me001,high,95]],linkableelt).
yes
?- schema_type([le002, [me002,medium,80]],linkableelt).
yes
?- schema_type([ac001,le001],actor).
yes
?- schema_type([ac002,le002],actor).
yes
?- schema_type([ce001,[le001, ac001]], containableelt).
yes
?- schema_type([ce002,[le002, ac002]], containableelt).
yes
?- schema_type([ie001, ce001], intentionalelt).
```

```
yes
?- schema_type([ie002, ce002], intentionalelt).

yes
?-
```

We now test the data linked to the intentional element with the reference number ie001:

```
?- data(ie001, Metadata, ModelEltdata, LinkableEltdata, intentionalelt).
.
.
Metadata = [‘maxtime‘, 2]
ModelEltdata = [‘md001‘, ‘id001‘, ozimprovappproc]
LinkableEltdata = [me001, high, 95] ;
no
?-

Interpreting project: phdproject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: ’commonclauses.pro’, ’ozprologclauses.pro’, ’staterefs.pro’
Type ’quit.’ to end and [Ctrl]-C to stop user input.

?- schema_type([sg001,[ie001,[]]],softgoal).

yes
?- schema_type([sg002,[ie002,[]]],softgoal).

yes
?- data(ie002,Metadata,Modeldata,Linkabledata,intentionalelt).

Metadata = ’[]’
Modeldata = ’[]’
Linkabledata = [me002, medium, 80] ;

Metadata = ’[]’
Modeldata = [‘md002‘, ‘id002‘, ozflexibleprocess]
Linkabledata = [me002, medium, 80] ;

Metadata = [‘size‘, 36]
```

```
Modeldata = [‘md002‘, ‘id002‘, ozflexibleprocess]
Linkabledata = [me002, medium, 80] ;
no
?-
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create an instance of the class ClsApplicant
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Interpreting project: phdproject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: ’commonclauses.pro’, ’ozprologclauses.pro’, ’staterefs.pro’
Type ’quit.’ to end and [Ctrl]-C to stop user input.

?- schema_type([ap001, [id001, ac001, [],[],[]]],applicant).
yes
?-
```

Implementing the operation schema: *NewSoftGoal*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creating a new softgoal for an applicant
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, softgoal).
schema_op([RefAp, RefSg], newsoftgoal):-

%variables
varname(RefSg, softgoal),

stateref([RefAp, [Appid,RefAc,Sgoals,Goals,Nfds]]),

% add the input softgoal RefSg to Sgoals is not already in it
insert(RefSg, Sgoals, NewSgoals),

save_stateref(stateref([RefAp, [Appid, RefAc, NewSgoals, Goals, Nfds]])),
reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Licensed to
```

```
Free Version

Interpreting project: phdproject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: 'commonclauses.pro', 'ozprologclauses.pro', 'staterefs.pro'
Type 'quit.' to end and [Ctrl]-C to stop user input.

?- schema_type([ap001, [id001, ac001, [],[],[]]],applicant).

yes
?- schema_op([ap001, sg001], newsoftgoal).

yes
?- schema_op([ap001, sg002], newsoftgoal).

yes
?-
```

6.8 The leftward validation phase

This phase is about ensuring the completeness and applicability of the specification, as well as the specification process.

6.8.1 Establishing the completeness of the specification

As discussed in Section 6.4.4, the followings activities were planned to ensure the completeness of the Object-Z specification: *backward traceability* analysis, *animation of selected components of the specification*, *preconditions calculation* for selected operations, and a *manual inspection* of the specification.

Backward traceability

A backward traceability analysis is carried-out to show that all significant requirements were addressed in the specification as recommended in option a) of Definition 6.4.5. This analysis was successfully performed in Section 6.7.1 the results of which reported in Tables 6.7 and 6.8.

Animating selected components of the specification

To reinforce the backward traceability in the effort to demonstrate option a) of Definition 6.4.5, it was also planned to animate some significant components of the specification. Such animation was fully discussed (see Sections 6.5.3, 6.5.3 and 6.7.2).

Manual inspection

An inspection of the specification was conducted during the rightward validation phase in Section 6.6.2 whereby the entire specification was fully reviewed. It is confirmed that figures, tables and diagrams, in this document in generale, and those related to the specification in particular are fully labeled and referenced hence, fulfilling option c) in Definition 6.4.5 that recommends full labels and references to all figures, tables, and diagrams in the SRS.

6.8.2 The applicability of the specification

If we consider the process of transforming a GRL model into an Object-Z specification as an operation, the analysis of the applicability of the specification can be reduced to the study of the precondition of the operation where the formal specification is seen as the postcondition (in a larger sense). Although the operation may not be strictly/fully formalised, it appears to us that such a study is convenient to ensure that the Object-Z model obtained through our proposed formalisation process is a useful description of the original GRL model.

Domain description

The analysis is based on the following definitions:

Definition 6.8.1. A **GRL model** is a set of GRL elements interconnected by means of GRL links. The different GRL elements include: softgoals, goals, tasks, resources, and beliefs.

Definition 6.8.2. An Object-Z specification is a finite set of related Object-Z elements. For example, class schemas, objects (instances of classes), operations, definitions, etc. are Object-Z elements related by means of inheritance, instantiation, inclusion, etc.

A Z specification of a GRL model and an Object-Z specification are proposed.

$$[GrlE, OzE]$$

$GrlE$ denotes the set of all possible elements of a GRL model. Since the topic of interest here is the applicability of the formal approach to describe GRL models, we prefer to keep the list of GRL elements rather abstract. In the same vein, the list of Object-Z elements denoted by OzE is also kept abstract.

Grlmodel

$$grlelements : \mathbb{F}\ GrlE$$
$$links : GrlE \pfun GrlE$$
$$\dom links \cup \ran links \subseteq grlelements$$

A GRL model is therefore, formed by a finite set of GRL elements that are interconnected by means of link elements represented by the partial function, named *links*.

InitGrlmodel

$$Grlmodel'$$
$$grlelements' = \varnothing$$
$$links' = \varnothing$$

INIT Grlmodel defines an initial state for a GRL model. Theorem 6.8.1 is used to prove that the proposed initial state is realisable.

theorem 6.8.1. $\exists\ Grlmodel' \bullet InitGrlmodel$

Ozspec

$$ozelements : \mathbb{F}\ OzE$$
$$related : OzE \pfun OzE$$
$$\dom related \cup \ran related \subseteq ozelements$$

The schema *Ozspec* specifies an Object-Z specification. As in the case of GRL models, the description of an Object-Z specification is kept simple and abstract. This is convenient for the type of analysis being conducted.

INIT Ozspec is an initial state for the schema *Ozspec*.

InitOzspec

$$Ozspec'$$
$$ozelements' = \varnothing$$
$$related' = \varnothing$$

theorem 6.8.2. $\exists\ Ozspec' \bullet InitOzspec$

The partial function *ozgrle* defined next associates GRL models' elements to Object-Z specification components. It should be perceived as being illustrated by the mappings in Table 6.8 where each GRL model element can be traced from at least one Object-Z specification's component.

$ozgrle : GrlE \pfun OzE$

The state schema *StateOzGrl* includes only one component that maps each GRL model to an Object-Z specification.

StateOzGrl

$$ozgrl : Grlmodel \pfun Ozspec$$

$$\forall grl : Grlmodel;\ oz : Ozspec \mid grl \mapsto oz \in ozgrl \bullet$$
$$\forall grle : GrlE \mid grle \in grl.grlelements \bullet$$
$$\exists oze : OzE \mid oze \in oz.ozelements \bullet$$
$$grle \mapsto oze \in ozgrle$$

The predicate part of this schema specifies the strategy adopted in this work to formalise a GRL model. That is, for an Object-Z specification in *Ozspec* to specify a GRL model, each element of the GRL model must be associated, by means of the partial function *ozgrle*, to an element of its Object-Z specification.

InitStateOzGrl

$$StateOzGrl'$$

$$ozgrl' = \varnothing$$

The initial state schema *INIT StateOzGrl* can established by proving Theorem 6.8.3.

theorem 6.8.3. $\exists StateOzGrl' \bullet InitStateOzGrl$

The formalisation proposed in this work is now summarised or abstracted with the operation *FormaliseGrlmodel*.

FormaliseGrlmodel

$$\Delta StateOzGrl$$
$$grlm? : Grlmodel$$
$$ozspec! : Ozspec$$

$$\theta StateOzGrl'.ozgrl = \theta StateOzGrl.ozgrl \oplus \{grlm? \mapsto ozspec!\}$$

The purpose of this analysis is to study the behaviour of this operation vis-a-vis the input GRL models to identify, if exist, the circumstances where a GRL model may or may not be formalisable. One way of doing this is to calculate the pre-condition of the operation.

Calculating the pre-condition with Z/Eves

A precondition is an operation, denoted by *pre*, that applies to operation schemas. Calculating a precondition of an operation helps to describe precisely the conditions under which

the operation is applicable, and therefore helps to avoid applying operations outside their domain. Such calculation involves, in general, two major steps (see Potter et al. [150], Woodcock [193], Woodcock and Davis [194]):

- First, to define the precondition schema of the operation by removing the after-state variables and outputs from the declaration part of the operation schema and existentially quantifying them in the predicate.
- Secondly to simplify the schema by applying predefined inference rules and techniques, such as the one-point-rule, which is defined later in this section.

The precondition for the operation *FormaliseGrlmodel* is defined as:

$$\mathit{Define}\ \ \text{pre}\ \ \mathit{FormaliseGrlmodel} \mathrel{\widehat=} \text{pre}\ \mathit{FormaliseGrlmodel}$$

The schema representation of pre *FormaliseGrlmodel* is:

pre *FormaliseGrlmodel*

$$\begin{array}{l} \mathit{StateOzGrl} \\ \mathit{grlm}? : \mathit{Grlmodel} \\ \hline (\exists\, \mathit{StateOzGrl}';\ \mathit{ozspec}! : \mathit{Ozspec}) \bullet \\ \quad \theta \mathit{StateOzGrl}'.\mathit{ozgrl} = \theta \mathit{StateOzGrl}.\mathit{ozgrl} \oplus \{\mathit{grlm}? \mapsto \mathit{ozspec}!\} \end{array}$$

Obviously, this schema does not seem to be easy to manipulate since the type of the input and outputs variables are themselves state schemas. Multiple steps may therefore be required to developed and simplify the schema. Thus, the decision to use Z/Eves theorem prover tool to automatically calculate the precondition. The software does the calculation when attempting to prove the Theorem 6.8.4.

theorem 6.8.4. $\forall\, \mathit{StateOzGrl};\ \mathit{grlm}? : \mathit{Grlmodel} \bullet \text{pre}\ \mathit{FormaliseGrlmodel}$

To successfully perform the calculation, the Z specification defined in this section, including the given sets: *OzE* and *GrlE*, the state schemas: *Grlmodel*, *Ozspec*, and *StateOzGrl*, as well as the partial function *ozgrle* and the operation *FormaliseGrlmodel* were first loaded onto Z/Eves followed by a command to request Z/Eves to prove Theorem 6.8.4 by reduce and to simplify the result whenever possible:

$$\begin{array}{l} => \mathit{try} \forall\ \mathit{StateOzGrl};\ \mathit{grlm}? : \mathit{Grlmodel} \bullet \text{pre}\ \mathit{FormaliseGrlmodel}; \\ \qquad \mathit{prove\ by\ reduce};\ \mathit{simplify}; \end{array}$$

After multiple steps of development and simplification, the theorem prover generated the result presented next. Only the inputs, the first and last steps are shown here; the middle steps were deleted to reduce the size. Lines were also added in between the proof steps to

make them more visible.

Z/Eves proof result:

$$
\begin{array}{l}
StateOzGrl \\
\land\ grlm? \in Grlmodel \\
- \\
\Rightarrow (\exists\, ozgrl' : \mathbb{P}(\langle\!|\ grlelements : \mathbb{P}\, GrlE, links : \mathbb{P}(GrlE \times GrlE)\ |\!\rangle \times \\
\quad \langle\!|\ ozelements : \mathbb{P}\, OzE, related : \mathbb{P}(OzE \times OzE)\ |\!\rangle); \\
\quad ozspec! : \langle\!|\ ozelements : \mathbb{P}\, OzE, related : \mathbb{P}(OzE \times OzE)\ |\!\rangle \bullet \\
\quad FormaliseGrlmodel) \\
- \\
deleted\ steps\ ... \\
- \\
\Rightarrow (\exists\, ozspec! : \langle\!|\ ozelements : \mathbb{P}\, OzE, related : \mathbb{P}(OzE \times OzE)\ |\!\rangle \bullet \\
\quad ozspec! \in Ozspec \\
\land\ (\forall\, grl\_0 : Grlmodel;\ oz\_0 : Ozspec \mid (grl\_0, oz\_0) \in ozgrl \oplus \{(grlm?, ozspec!)\} \bullet \\
\qquad (\forall\, grle\_0 : GrlE \mid grle\_0 \in grl\_0.grlelements \bullet \\
\qquad\quad (\exists\, oze\_0 : OzE \bullet (oze\_0 \in oz\_0.ozelements \\
\land\ (grle\_0, oze\_0) \in ozgrle))))) \\
- \\
Command\ had\ no\ effect. \\
=>
\end{array}
$$

This result looks interesting considering the state of the theorem at the beginning of the proof and the result after all the possibilities being explored.

To summarise, the condition for the input GRL model ($grlm?$) to be transformed into $ozspec!$ is for each element of $grlm?$ to be formalisable; where the formal specification of the element is a component of the $ozspec!$. This is incorporated into the following mathematical expression:

$$
\begin{array}{l}
\forall\, grle\_0 : GrlE \mid grle\_0 \in grl\_0.grlelements \bullet \\
\qquad\quad (\exists\, oze\_0 : OzE \bullet \\
\qquad\qquad (oze\_0 \in oz\_0.ozelements \\
\land\ (grle\_0, oze\_0) \in ozgrle)))
\end{array}
$$

Recall that $(grle\_0, oze\_0) \in ozgrle$ indicates that the Object-Z element oze_0 formalises the GRL element $grle\_0$. This condition enlighten the usefulness of $ozgrle$, to transforms GRL elements into Object-Z components, in the overall formalisation process. Thus, the importance of Object-Z templates developed in Chapter 4, Section 4.3, pp. 66 - 77 on which $ozgrle$ is entirely based.

6.9 The downward validation phase

This validation phase seeks to demonstrate that the evisaged system can effectively be obtained from the specification(feasibility) and that it can be demonstrated that such a system meets its requirements.

6.9.1 The operational feasibility analysis

It is a known fact software products or computers' systems can be generated from Z/Object-Z specifications. Known refinement techniques and successful software and hardware projects developed from Z/Object-Z specifications are very supportive. Hence, the issue at this stage is not about the feasibility of Object-Z specifications in general, but rather about the Object-Z specification describing GRL models obtained by applying the GRL models formalisation process proposed in this work.

Constructing a UCM from the Object-Z specification

The analysis of the relationship between GRL and UCM was discussed in Chapter 4, Section 4.1, pp.53-59. Focusing on the joined and iterative construction of GRL and UCM illustrated in Figure 4.1, it was observed that UCM refines GRL and the bulk of UCM inputs are generated by GRL through goals analysis and operationalisation. It was also argued that although use cases and/or scenarios from user requirements can be used directly as inputs to UCM, those inputs to be justified must be connected in one way or another to some goals or softgoals in GRL. To the best of the authors knowledge, there isn't any explicit or formal mapping between the inputs from GRL and the model elements of UCM meaning that the refinement of each input merely depends on the analyst understanding.

The idea in this section is to compare the inputs to UCM from two sources:

- the Object-Z specification, developed in Chapter 5 and presented in (Appendix C),
- the GRL model in fig.5.2, p.106.

The purpose of this comparison is initially to show that the formalisation process does not loose information. Additionally, it enlightens any improvement brought into the process by the formal model.

Linking GRL elements to UCM model elements

Table 6.10 presents the inputs from the GRL model with an attempt to map each GRL element to the UCM model elements that are most likely to be used for its specification.

No.	GRL Elements: GRL elements	**Type of input**	UCM model elements: Responsibility	AndFork-OrFork	AndJoin-OrJoin	WaitingPlace	StartPoint-EndPoint	EmptyPoint	StaticStub	DynamicStub	SynchronizationStub	Team	Process	Object	Agent	Actor
01	**Applicant**	Actor							X			X		X	X	X
02	ImprovAppProc	Softgoal														
03	FlexibleProcess	Softgoal														
04	ReduceAppTime	Goal														
05	AppOnline	Goal														
06	AccessOwnApp	Task	X				X	X					X		X	
07	SubmitAppOnline	Task	X				X	X					X		X	
08	**Motivator**	Actor							X			X		X	X	X
09	ImprovMotivEvalProc	Softgoal														
10	CombineMotivEval	Softgoal														
11	ColabWithAdmin	Goal														
12	MotivEvalOnline	Goal														
13	MeetingToMotivEval	Task	X				X	X					X		X	
14	SubmitMotivEvalOnline	Task	X				X	X					X		X	
15	ReportOnSubmitedApp	Task	X				X	X					X		X	
16	**Administrator**	Actor							X			X		X	X	X
17	ResolvEmailPb	Softgoal														
18	ImprovEvalMotivTime	Softgoal														
19	FacilAppManagement	Softgoal														
20	HaveConsistForms	Goal														
21	InstantAccessToPgData	Goal														
22	GenStatsOnApp	Task	X				X	X					X		X	
23	ProvideInfoGenForms	Task	X				X	X					X		X	
24	KeepPgDataInDb	Task	X				X	X					X		X	
25	**Server**	Actor							X			X		X	X	X
26	ColabHRSys	Goal														
27	GenFormSections	Task	X				X	X					X		X	
28	MaintSubmitedApp	Task	X				X	X					X		X	
29	MaintDb4SuppPg	Task	X				X	X					X		X	
30	GenSecBelief	Belief														
31	Intranet	Resource										X		X		
32	Internet	Resource										X		X		

Table 6.10: Mapping GRL components to UCM model elements

UCM model elements are represented in columns and GRL elements in lines. Darkened

lined (in gray) indicate GRL elements that are not considered for UCM specification. These include goals and softgoals, as well as link elements that are not in the table. The connection between inputs elements in lines and UCM model elements in columns, represented by a cross (X), results from an attempt to identify UCM elements that are most likely to be used in the UCM modelling of an input element. Such identification is mainly based on the type of the input entity considered, as it is described in the GRL notation. The following examples are presented to illustrate:

Example 6.9.1. In GRL, an actor models an entity that has intentions and carries out actions to achieve its goals; it often represents a stakeholder or a system (see URN[1], p.26). The actions carried out by an actor are the tasks that operationalize goals and softgoals of the actor definition. Thus, the most plausible UCM elements for an actor definition include: Team, Object, Agent and Actor's components. Since an actor may also represent a system, it is very likely to create a (static) stub for such an actor's definition.

Example 6.9.2. A task specifies a particular way of doing something (see URN [1], p.30). This definition of a task is so generic that any action or activity would be included and therefore, any UCM model element describing actions would be considered: Responsibility, Start-Point, End-Point, Process, and Agent.

Example 6.9.3. A resource in GRL is a physical or informational entity for which the main concern is whether it is available (see URN[1], p.30). To the authors' understanding, this definition suggests a UCM Object component as the most plausible model element to specify a resource.

It is the author's belief that knowing beforehand UCM elements required to specify an input entity would positively contribute to the automation of the process. However, a thorough analysis of the individual entity remains a viable option that provides for detailed information needed for the complete modeling of the entity.

Linking Object-Z components to UCM model elements

Table 6.11 illustrated the association between the Object-Z components and UCM conceptual elements. Object-Z entities are obtained from the traceability analysis table shown in (Table

6.8,P.154), that readily associates each GRL element (used in Table 6.10) to the Object-Z element(s) that formalises it. The column titled "**Type source GRL**" includes the type of GRL element formalised by the Object-Z element that precedes it in the same line. A cross symbol (X) in a cell indicates a mapping identical to one of the corresponding GRL element in Table 6.10. A star symbol (*) is used for any extra linking due to the formalisation.

No.	Object-Z components: OZ compnents	Type source GRL	UCM model elements: Responsibility	AndFork-OrFork	AndJoin-OrJoin	WaitingPlace	StartPoint-EndPoint	EmptyPoint	StaticStub	DynamicStub	SynchronizationStub	Team	Process	Object	Agent	Actor
01	**ClsApplicant**	Actor	*				*	*	X			X		X	X	X
02	ozImprovAppProc	Softgoal		*	*	*			*	*	*	*			*	
03	ozFlexibleProcess	Softgoal		*	*	*			*	*	*	*			*	
04	ozReduceAppTime	Goal														
05	ozAppOnline	Goal														
06	ClsAccessOwnApp	Task	X			*	X	*					X		X	
07	ClsSubmitAppOnline	Task	X			*	X	*					X		X	
08	**ClsMotivator**	Actor	*				*	*	X			X		X	X	X
09	ozImprovMotivEvalProc	Softgoal		*	*	*			*	*	*	*			*	
10	ozCombineMotivEval	Softgoal		*	*	*			*	*	*	*			*	
11	ozColabWithAdmin	Goal														
12	ozMotivEvalOnline	Goal														
13	ClsMeetingToMotivEval	Task	X			*	X	*					X		X	
14	ClsSubmitMotivEvalOnline	Task	X			*	X	*					X		X	
15	ClsReportOnSubmitedApp	Task	X			*	X	*					X		X	
16	**ClsAdministrator**	Actor	*				*	*	X			X		X	X	X
17	ozResolvEmailPb	Softgoal		*	*	*			*	*	*	*			*	
18	ozImprovEvalMotivTime	Softgoal		*	*	*			*	*	*	*			*	
19	ozFacilAppManagement	Softgoal		*	*	*			*	*	*	*			*	
20	ozHaveConsistForms	Goal														
21	ozInstantAccessToPgData	Goal														
22	ClsGenStatsOnApp	Task	X			*	X	*					X		X	
23	ClsProvideInfoGenForms	Task	X			*	X	*					X		X	
24	ClsKeepPgDataInDb	Task	X			*	X	*					X		X	
25	**ClsServer**	Actor	*				*	*	X			X		X	X	X
26	ozColabHRSys	Goal														
27	ClsGenFormSections	Task	X			*	X	*					X		X	
28	ClsMaintSubmitedApp	Task	X			*	X	*					X		X	
29	ClsMaintDb4SuppPg	Task	X			*	X	*					X		X	
30	ozGenSecBelief	Belief														
31	ClsIntranet	Resource	*			*	*	*						X		
32	ClsInternet	Resource	*			*	*	*						X		

Table 6.11: Mapping Object-Z components to UCM model elements

When linking Object-Z classes to UCM conceptual elements, the main criteria was to con-

sider the class itself and its components without performing any content analysis of both the class and its components. The goal is to intentionally limit the contribution of the formal model at the initial phase of a UCM model construction by considering only the structure of the specification. The idea is for example to let the system suggest different options for the initial model and when the designer has decided, detailed analysis of the formal specification may hence be exploited to refine the (generated/initial) UCM model.

To complete the table, the column indicating the type of GRL element being formalised is first used. For each type, the mapping for the same type in Table 6.10 is replicated using the same cross symbol (X). After this first step, the star symbol is used to complete the table with additional linking that results from considering the type and composing elements of the input object-Z entity.

Example 6.9.4. since a class schema generally contains a state with variables and at least one primitive operation to update the variables in the class, UCM elements that may be used for an Object-Z class should at least include:

- Responsibility point(s) for primitive operations, and
- Component to model the state of the class schema.

Example 6.9.5. each Object-Z entity describing a softgoal comprises a component that prescribes actions (CNF-action) on parts or entire model to propagate the influence of the non-functional requirements on the model. Examples of CNF-actions:

- Introduce concurrency with: And-fork and Join-fork to reinforce performance,
- Group related elements together (stubbing, team) to apply security measures,
- Create alternative paths (Or-fork, Or-Join, stubs) to reinforce flexibility or compatibility.

It is worth noticing that a CNF-action defines actions on the UCM model but not actions in the model. That is why the element Responsibility point is not selected in the table.

Comparing the two linkings

The comparison is based on the assumption that if for each input entity all the UCM element(s) needed for its specification is known, then the generation of the complete UCM specification is made easier and the construction process can also be, partially or fully, automated. Thus, by comparing the two tables, the following observations appear:

1- The linking in Table 6.10 are included in Table 6.11 confirming that the formalisation process does not cause any loss of information.

2- Inputs from Object-Z are likely to be easier to model with UCM than the GRL elements that they formalise. This is supported by the fact Table 6.11 readily suggests more UCM elements to model each of the Object-Z inputs than Table 6.10 for GRL elements.

3- With the concept of Complementary Non-Functional action (CNF-action), the Object-Z specification provides for a mechanism to stimulate thinking about actions to perform on the UCM to optimize the model with the purpose to achieve non-functional/quality requirements.

Two examples are used next to illustrate this comparison: the first example presents the UCM modeling of the actor definition *Applicant* and all elements bound to it as well as *Internet* one of the resources needed. The second constructs the UCM specification of the class schema *ClsApplicant*, the two classes: *ClsAccessOwnApp* and *SubmitAppOnline*, as well as, the class schema *ClsInternet*.

Example 6.9.6. UCM specification of *Applicant* and *Internet*

- *Applicant*- four UCM elements proposed are: Static Stub, and Team, Object, Agent, and Actor components. Since it can be known from the GRL model that the actor's definition includes other elements, the best choice would be on the Team and Actor components. Since there is doubt on whether an actor component can contain other elements,choosing the the Team component over the actor component is preferable.

  To summarise, the use of the Stub is not yet decided, the object, Agent and Actor components are discarded, and the Team component chosen. This context can be save for future consideration. For example,

  savecontext(*Appliquant*[StaticSTub(0), Team(1), Object(-1),Agent(-1), Actor(-1)]).

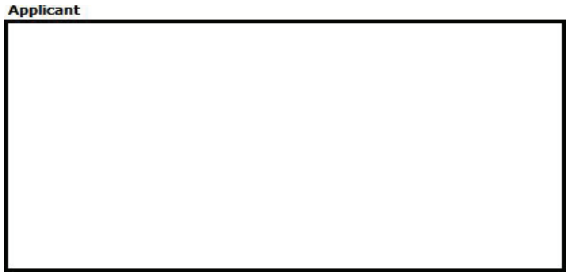

Figure 6.7: UCM specification of the GRL actor applicant

- *AccessOwnApp / SubmitAppOnline*:
  The task *AccessOwnApp* defines the means through which an applicant access its own submitted application. The suggested elements in Table 6.10, for a Task, are: Responsibility, Start and End Points, Process, and Agent. Process, Start and End Points may be chosen and hence, the following context saved for future use:
  savecontext(*AccessOwnApp*[Responsibility(-1), StarPoint(1), EndPoint(1), EmptyPoint(1), Process(1), Agent(-1)])
  The task *SubmitAppOnline* defines a means for an applicant to submit its application online. The task is analysed in the vein as *AccessOwnApp*.

- *Internet* is a resource needed for online communications as intended in Figure 5.2. To specify the resource, the only possible UCM conceptual elements to select from (see Table 6.10), are: Team and Object. Since the resource stands alone, the Team component is more appropriate for its modeling.

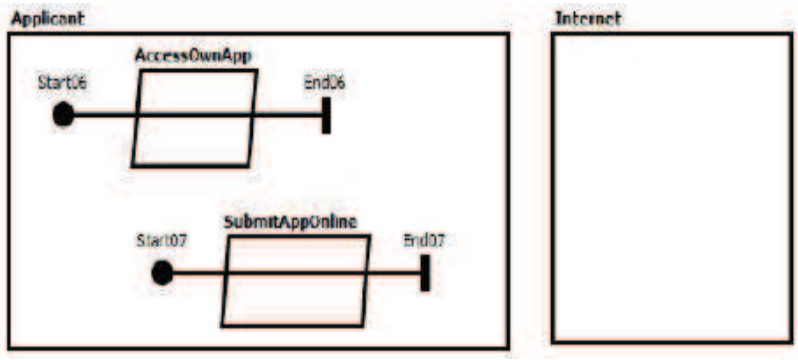

Figure 6.8: UCM specification of the GRL actor applicant

At this stage, more detailed UCM map may be obtained only by means of further analysis.

Example 6.9.7. UCM specification of *ClsApplicant*, *ClsAccessOwnApp*, *ClsSubmitAppOnline*, and *ClsInternet*

- To specify the class *ClsApplicant*, Table 6.11 provides for a set of conceptual elements from which one may choose the most appropriate. These are: Responsibility, StartPoint, EndPoint, EmptyPoint, StaticStub, Team, Object, Agent, and Actor. Some of the elements are naturally made to be together, for example, a Responsibility point is always placed between a Start point and an End Point.

  It is also important to notice that any class schema with state variables, naturally defines at least primitives operations to update its state. These operations are abstracted in this example with one responsibility point, namely, *UpdateState*.

UCM model elements chosen for the class *ClsApplicant* are therefore:

- Responsibility point (*UpdateStateApplicant*), with a start point and an end point, to represent the primitive operations to update the state of the class,
- a team component to represent the class itself,
- an object component to represent the state of the class, and
- the decision to use a stub may be taken at a later stage.

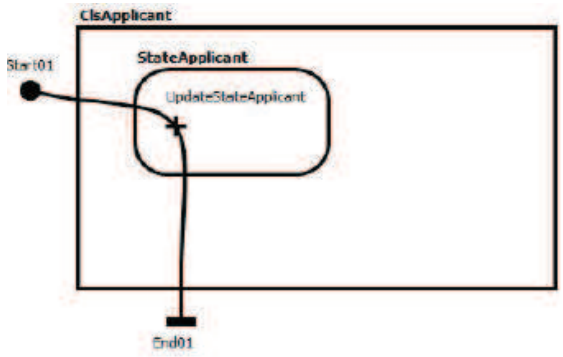

Figure 6.9: UCM specification of the class schema *Clsapplicant*

- *ClsAccessOwnApp* / *ClsSubmitAppOnline*
  The UCM conceptual elements provided in Table 6.11 for class schemas formalising the same type of GRL elements are the same. The elements to choose from are: Responsibility, WaitingPlace, StartPoint, EndPoint, EmptyPoint, StaticStub, Team, Process, Object, and Agent. The selection for each of the two classes is the same:
  - Responsibility point, StartPoint and EndPoint, to represent the operation *SelectApp* for the class *ClsAccessOwnApp* and *SubmitApp* for the class *ClsSubmitAppOnline*,
  - no decision for WaitingPlace and StaticStub,
  - Team, Object and Agent rejected, and
  - Process to represent each of the two classes.

  The resulting map is presented in Figure 6.10. Notice that the state of the two classes are not represented. This is because their common component, *inet*, is in fact a reference to an object of another class and hence accessible from that class. The variable, *nfdependers* in the class *ClsAccessOwnApp* is updated only when the specification is being constructed; afterward, it becomes like a constant. The UCM modeling of the class *ClsInternet* describing a resource needed by other classes is considered next.

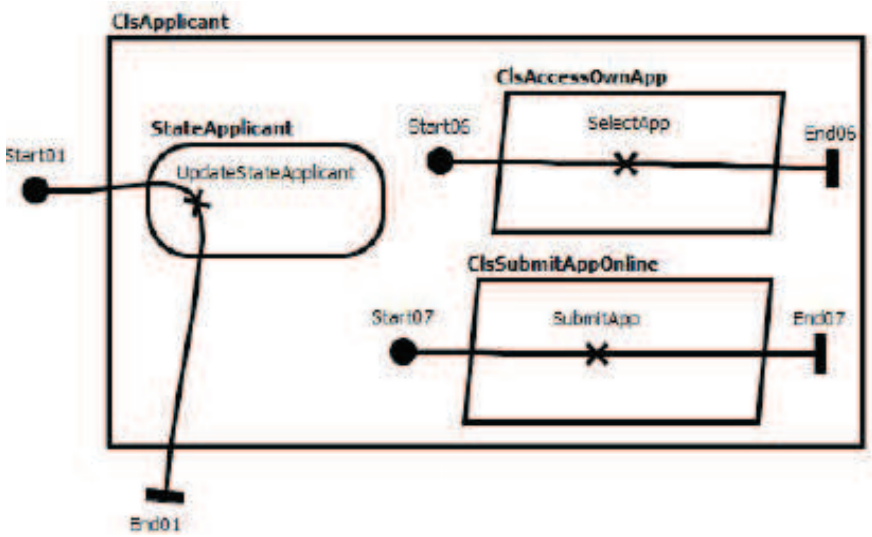

Figure 6.10: UCM specification of the class *ClsApplicant* and processes

- The UCM modeling of the class *ClsInternet* is done in a similar way as in the cases above. The resulting model is presented in Figure 6.11. The purpose of including this class in the example is mainly to show how the selection of some UCM elements can be made at later stages.

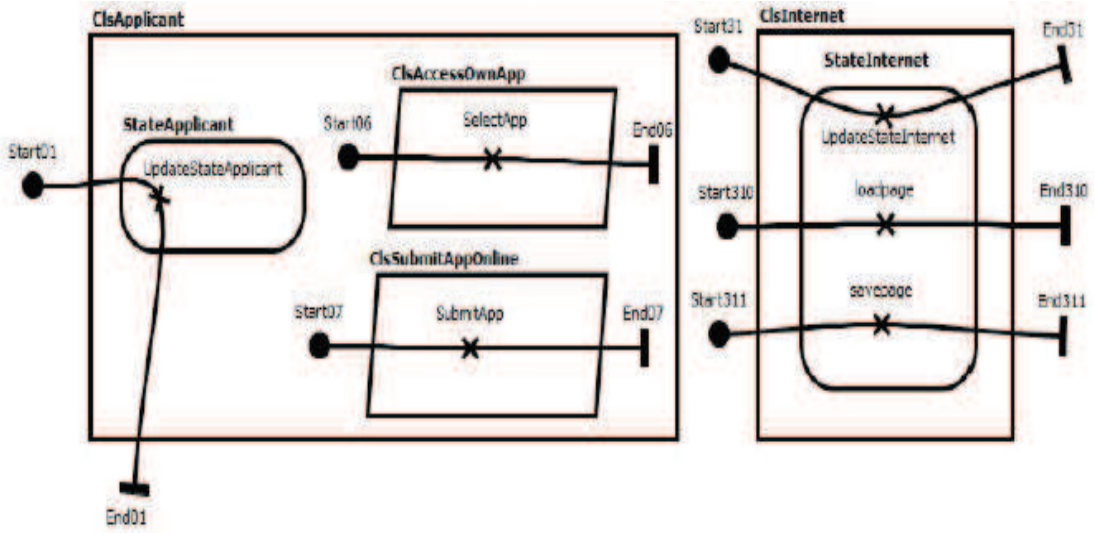

Figure 6.11: UCM specification for ClsApplicant, ClsAccessOwnApp, ClsSubmitAppOnline, ClsInternet

As mentioned above, the two classes *ClsAccessOwnApp* and *ClsSubmitAppOnline* inherit objects from the class *ClsInternet*. This justifies why a waiting place and/or a stub were suggested for their modeling: means to connect to the map specifying the inherited class. A quick look at the operations *SelectApp* in *ClsAccessOwnApp* and *SubmitApp* in *ClsSubmitAppOnline* also confirms the necessity for the connection and makes it more convenient to choose the

WaitingPlace over the StaticStub. For example, consider the following operation:

$inet.saveform(myform?)$

The operation requires *inet*, a reference to an object of the class *ClsInternet* to save an input webpage via the internet connection. The advantage of the waiting place is that, not only, it would provide means to communicate (send a request via the Internet) but, also allows for a timeout-recovery mechanism to control the communication. The next example is presented

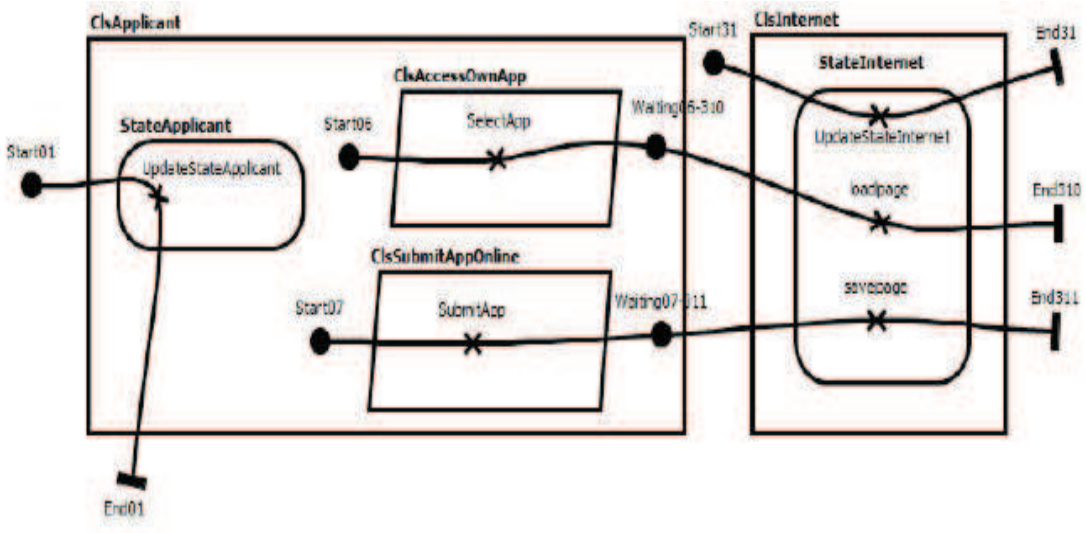

Figure 6.12: UCM model for ClsApplicant, ClsAccessOwnApp, ClsSubmitAppOnline, ClsInternet

to illustrate how the analysis of softgoals may help to create CNF-actions to optimise the model.

Illustrating the concept of Complementary Non-Functional Actions: CNF-actions

Example 6.9.8. as mentioned above, the component *nfdependers*, in a class schema, contains the list of softgoals that depend on the class for their achievement. In the case of the model in Figure 6.12, only two softgoals are involved: the first is *ozImprovAppProc* whose main goal is to improve the application process. The second is *ozFlexible* whose purpose is for the application process to be flexible. The second softgoal is in fact a refinement of the first one. However, both can still be analysed together since Table 6.11 suggests the UCM conceptual elements for any softgoal.

The suggested list of elements to choose from includes: AndFork, OrFork, AndJoin, OrJoin, WaitingPlace, StaticStub, DynamicStub, SynchronizationStub, Team component, and Agent component.

The analysis of softgoals in this case, stimulates thinking about:

- Concurrency/parallelism (AndFork, AndJoin),
- Alternatives (OrFork, OrJoin),
- Communication (WaitingPlace [Timer]),
- SubSystems (StaticStub, DaynamicStub, SynchronizationStub), and
- Modularisation (Team component, Agent).

with the purpose to perform appropriate actions (CNF-actions) on the model to achieve stated software qualities. For example, in Figure 6.12, one may argue that having two distinct path segments to connect to the Internet; one to access submitted application forms and the other to submit new applications, is not flexible and also time consuming because it requires an applicant to open a new Internet connection for each operation. This problem involves two aspects of the map: the Internet communication and the choice between parallel and alternative execution of the two processes.

Concerning the second aspect, that is to make decision about parallel or alternative execution of the two processes, the most relevant CNF-action is to or-join the two path segments before the connection to the Internet and Or-Fork them before they are processed. This would eliminate one waiting place and keep only one for the communication.

Regarding the control of communication, since there is already a waiting place to control the communication, the CNF-action would consist to transform the waiting place into timeout-recovery mechanism by replacing the waiting place with a Timer and by adding a timeout path through which appropriate actions may be taken when timeout occurs.

$$
\begin{array}{lll}
CNF-action & \widehat{=} & \\
\quad id & = & ozFlexibleProc.id \\
\quad name & = & \\
\quad importance & = & ozFlexibleProc.importance \\
\quad impQualitative & = & ozFlexibleProc.impQualitative \\
\quad phase & = & \text{“}UCM\ specification\text{”} \\
\quad actor & = & ClsApplicant.id \\
\quad action & = & \text{“}Replace\ watingPlace06\ with\ Timeout-Recovery\ mechanism\text{”} \\
\quad domain & = & UCM < ClsApplicant,\ ClsAcessOwnApp,\ ClsSubmitAppOnline, \\
 & & ClsInternet > \\
\quad description & = & \text{“}map\ optimisation\ to\ achieve\ flexibility\ in\ order \\
 & & to\ improve\ the\ application\ process.\text{”}
\end{array}
$$

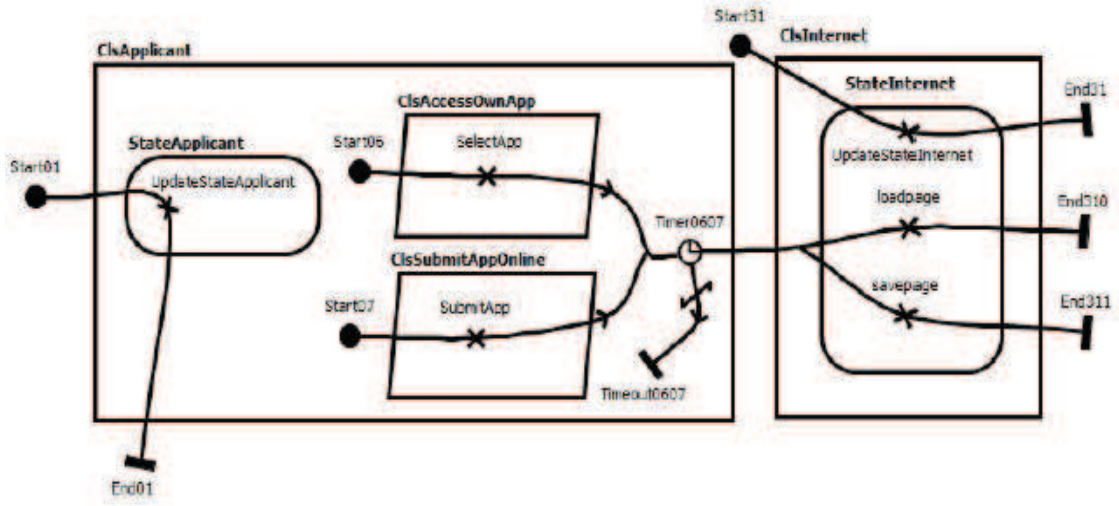

Figure 6.13: Optimised UCM model for ClsApplicant, ClsAccessOwnApp, ClsSubmitAppOnline, ClsInternet
The resulting map is represented in Figure 6.13.

6.10 Chapter conclusion

This chapter was presented in three parts. The first part introduced different specification validation techniques including parsing and type checking, animation, and mathematical proofs. An approach to animate an Object-Z specification using Prolog was also suggested followed by an introduction to the four-way framework for validating a specification.

The second part covered the overall planning of the validation. Based on the four-way framework, the validation of the Object-Z specification of the case study was planned for an iteration over four phases. The rightward validation phase to establish the consistency of the specification, the upward targeting the traceability and correctness of the specification, the leftward to address the completeness and applicability of the specification, and lastly the downward phase during which the feasibility of the specification was analysed. The four-phase planning was summarised in Table 6.1. The planning of the Prolog animation of the specification was also discussed based on the prototyping process in Figure 6.2 proposed by Sommerville [169].

The third and last part included the four-phase validations process. The rightward phase during which the specification was parsed and type checked using the Community of Z Tools (CZT) and errors corrected followed by a review that enlightened the hierarchical structuring of the specification illustrated in Figure 6.4. The upward phase proceeded with

the traceability analysis (see Table 6.7 and 6.8) to show that each Object-Z specification component was traceable from the original GRL model and that to each GRL element in the model, there was at least one Object-Z entity that formalises it. The correctness of the specification was also established through animation and argumentation. The leftward phase exploited the traceability analysis, the result from the Prolog animation, as well as the review of the specification performed in previous phases to conclude the completeness of the specification. Domain elements and properties were formalised with the Z notation, theorems formulation, as well as the use of the theorem prover Z/Eves were all together exploited to justify the applicability of the specification. The downward validation phase focused on the UCM modeling process to discuss the operational feasibility of the specification.

The next chapter proceeds with the analysis and generalisation of the main ideas developed in Chapters 4, 5 and 6.

Chapter 7

Analysis and Generalisation

This chapter performs a review of the work presented in the previous chapters of this document. The purpose of the review is to analyse all identifiable contributions and when possible, examine the possibility to generalise the result. The contributions are organised into categories including: contributions related to the URN construction process, the complementary non-functional actions (CNF-actions), contributions related to the proposed GRL formalisation approach, to the case study as well as contributions resulting from the Object-Z specification validation.

Next is presented the analysis of contributions related to the URN construction process.

7.1 URN construction process

As a result of the analysis of relationship between GRL and UCM based on the URN construction process presented in Chapter 4, Section 4.1, pp. 53 - 59, it was concluded that:

(a) a UCM specification is a refinement of a GRL model,

(b) a more consistent link/relationship needs be established between GRL and UCM to fill the conceptual gap that currently exists between the two techniques.

These two results are discussed next.

7.1.1 Discussion about UCM refining GRL

The UCM model in Figure 6.8, p.179, as well as the modeling process followed in (Example 6.9.6, p.178) to generate the model, best illustrates how a UCM model should not be perceived as a mere translation of the input GRL elements, but rather their refinements. It is observable from the example that to generate a more expressive UCM specification, a

detailed analysis of the input elements is required. This discussion owes its worthiness from the inherent nature of the concept of refinement and the two important activities that it requires to be accomplished.

- the first activity includes, for instance, transforming/decomposing the model from its current state into a more detailed one, and
- the second consists to validate the refined model against the initial one.

These two activities bring in the need to rethink, at the conceptual level, a mechanism to facilitate the tracing/linking of UCM elements to those of the input GRL. One would also expect the suggestion of a method to analyse GRL elements to facilitate their UCM specification.

As illustrated in Example 6.9.6, developed in Chapter 6, Table 6.10, on page 173 provides for a means to associate GRL elements to UCM conceptual components to facilitate the construction of the UCM map. During the UCM specification, for an input GRL element, the table together with the UCM elements selected, using the table and accessible through the operation such as

$$savecontext(\text{Appliquant}[StaticSTub(0), Team(1), Object(-1), Agent(-1), Actor(-1)])$$

constitutes a perfect approach to link GRL elements to those of UCM and hence, facilitate the tracing of those elements from one model to another. Let's recall that the operation *savecontext*, used above, stores the following information:

- *Applicant* - a GRL element of type actor that is to be specified,
- *StaticStub*, *Team*, *Object*, *Agent*, and *Actor* - UCM conceptual elements proposed to specify the actor's definition *Applicant*, based on the information in Table 6.10,
- $StaticStub(0)$ - the selection of this element is not yet decided.
- $Team(1)$ - a team component is chosen to specify the actor definition *Applicant*.
- $Object(-1)$, $Agent(-1)$, and $Actor(-1)$ - these components are rejected.

It is clear that having this information stored and made available provides for a technique for backward and forward traceability between GRL and UCM elements. Thus, the next important aspect discussed next, focuses on the analysis technique to facilitate the UCM modeling of an input GRL element.

7.1.2 Discussion about improving the relationship between UCM and GRL

Example 6.9.7 presented from page 179 to 184, illustrates a UCM specification of a set of Object-Z elements, the classes: *ClsApplicant*, *ClsAccessOwnApp*, *clsSubmitAppOnline*, and *ClsInternet*, resulting in the UCM model presented in Figure 6.11, p.181. The modeling process used in Example 6.9.6 was followed with the use of Table 6.11, similar to Table 6.10, to suggest for each input Object-Z element, a list of UCM conceptual elements appropriate to specify the input Object-Z entity. However, a closer look at the two UCM maps produced in both examples, reveals the map from Example 6.9.7 is more detailed and complete (describes better the input elements and is more expressive) than the one in Example 6.9.6.

The high quality of the UCM model produced in Example 6.9.7 can rightfully be attributed to two important factors that need further discussion:

- the application of our proposed concept of Complementary Non-Functional Actions, namely CNF-actions, to stimulate thinking about additional actions to perform, on a model in construction, to reinforce the achievement of softgoals (non-functional requirements), and
- the formalisation of GRL elements, with Z/Object-Z, that induces a preliminary detailed analysis of the initial GRL model before the UCM specification.

Each of these two points are further discussed next.

7.2 Complementary Non-Functional Action: CNF-action

7.2.1 The essence of CNF-action

This work proposed in Section 4.2.3, pp. 62 - 66 the concept of Complementary Non-Functional Action (CNF-action) to propagate the analysis of non-functional and quality requirements to each software development phase as illustrated in Figure 4.3.

We argue that the major part of softgoals and/or non-functional requirements are actions performed on models and not functional requirements that can be explicitly added to the list of existing functional requirements.

Inherently, a non-functional requirement has an influence on the system at each phase of the software development and a once-off analysis of them would be too limited to bring in the

full potentials of such requirements.

One of the most difficult aspect of non-functional requirements is conflict resolution when analysing the influence of the requirements specification phase.

7.2.2 Further efforts needed

The main objective of the proposed concept, the CNF-action, is to stimulate at each software construction phase, thorough thinking about developing strategies to allow parallel/concurrent analysis and modeling of functional and non-functional/quality requirements. Thus, further efforts are expected to address the following aspects:

1. develop at each phase strategies for non-functional requirements analysis,
2. strategy for conflict resolution, and
3. strategy to document and propagate actions performed from one stage to the next one.

Next is the discussion about the URN formalisation approach presented in this work.

7.3 Contributions related to the GRL formalisation approach

One of the most important contributions of this work is the GRL formalisation approach presented in Chapter 4, illustrated in Chapter 5 with a case study and validated in Chapter 6. The fundamental idea of the formalisation process is based on the basic GRL modeling approach introduced next.

7.3.1 The basic GRL modeling approach

Figure 7.1 illustrates our perception of the basic GRL modeling process summarised in three phases: phase A, B, and C.

Phase A: the input analysis, consisting to identify amongst the model elements, the most appropriate one(s) to model the input under consideration.

Phase B: one or more instance(s) of the GRL conceptual element(s) identified during the input analysis phase is created and added to the GRL model in construction.

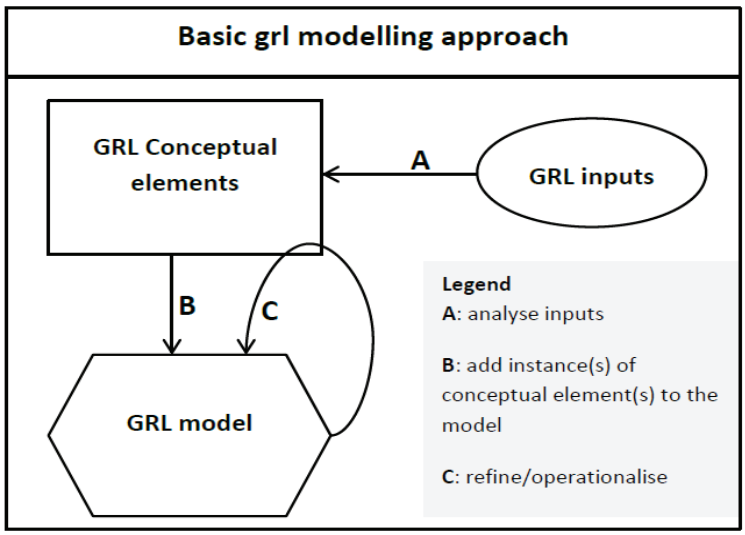

Figure 7.1: Illustrating the basic GRL modeling approach

Phase C: element(s) in the GRL model are iteratively refined/operationalised to obtain the final version of the model. The refinement process consists to analyse the element at hand, identify the most appropriate model elements (e.g., Task, Contribution) that would best refine/decompose/operationalise the selected element. Then, the model element(s) are instantiated and added to the model in construction.

In summary, each GRL model, specifying a set of inputs, is obtained from the GRL conceptual elements by instantiation. The idea for the formalisation deduced from this basic approach is the following: if all conceptual elements are formalised, then the formal specification of a GRL model can directly be obtained from the formal specifications of the conceptual models.

7.3.2 The basic formalisation strategy

The generic approach to formalise an input GRL model, proposed in this work, fully depends on the ability of the formal technique to formalise the GRL conceptual elements. The approach is depicted in Figure 7.2 with four specification steps, namely, Phase A, B, C, and D briefly explained in the legend.

Phase A: the formalisation of GRL conceptual elements with the development of templates; each template specifying a specific model element. It is only when the complete set of templates is created that the technique can be used to specify an input GRL model.

Phase B: the analysis of the input GRL model pertaining to identify, for each input element, an appropriate template to formalise the element.

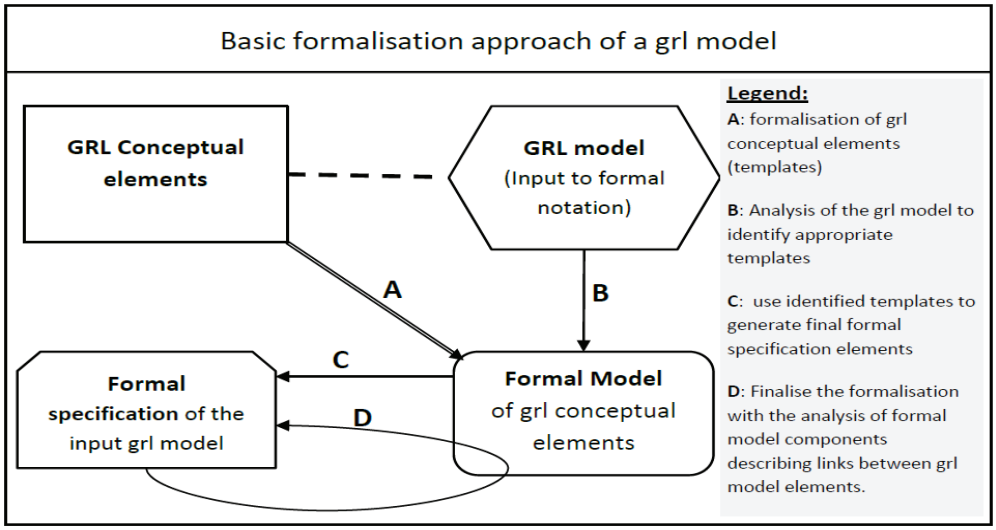

Figure 7.2: Illustrating the basic GRL formalisation approach

Phase C: the transformation of the template, identified during the input analysis, to generate the formal specification of the input element. In the case of the Object-Z technique used in this work to illustrate the approach, the transformation process is based on two concepts: inheritance and instantiation.

Phase D: the formal model constructed in the previous phases is finalised in the light of components describing links connecting the elements of the input GRL model. The process may require the use templates to create new objects to update the formal specification.

To summarise, the condition for a (formal) method/technique to be considered for GRL models formalisation based on the above basic approach is the following:

Considering the formal notation X, the condition for formalising a GRL model into X are:

- the ability of the method X to formalise GRL conceptual elements, namely templates (element type),

- the ability of the method X to formalise elements of the input GRL model from the templates.

This strategy was formalised in (Chapter 6, Section 6.8.2, pp. 167 - 171) with the Z notation. The operation *ozgrle* defined as: $ozgrle : GrlE \pfun OzE$ formalises each GRL conceptual element into a template. The operation *FormaliseGrlmodel* was defined to transform an input GRL model *grlm*? into an Object-Z specification *ozspec*!.

FormaliseGrlmodel

$\Delta StateOzGrl$
$grlm? : Grlmodel$
$ozspec! : Ozspec$

$\theta StateOzGrl'.ozgrl = \theta StateOzGrl.ozgrl \oplus \{grlm? \mapsto ozspec!\}$

The precondition calculation for the operation *FormaliseGrlModel* was automated with Z/Eves theorem prover, the result of which obviously confirmed that for the formal specification *ozspec*! to formalise the input GRL model *grlm*?, each GRL conceptual element must be formalisable.

```
Note that although in the Z specification of the strategy, Object-Z was
considered the formal method, no Object-Z concept was used neither in the
specification nor in the proof thus, the formal technique could simply be
kept generic and be noted for example, method X.
```

7.3.3 Future efforts needed

The Object-Z method was exploited in this work to explore different aspects of this formalisation strategy. However, for the strategy to become a model or standard, more experiments with other types of formal techniques (e.g., algebraic methods, ontology) are still required. One should also consider analysing the ability of the strategy to cope with changes in GRL model affecting either the structure or the model elements.

The major contributions of this work were generated in the following areas: when developing the templates and the framework presented in Chapter 4. The other phase during which tangible contributions were developed is during the application to the case study presented in Chapter 5. And lastly, some contributions came from the validation phase discussed in Chapter 6 during which the object- z specification of the case study, as well as the specification process were analysed.

7.4 Contributions associated to Object-Z specification

To complete the Object-Z specification of GRL models, three main contributions were developed in this work. Firstly, an approach to formalise each GRL conceptual element to create templates. Secondarily, a framework to formalise GRL models with Object-Z was proposed. And lastly, algorithms to describe each important step of the formalisation process were derived.

7.4.1 Templates: approach to formalise GRL model elements

The formalisation of the GRL conceptual model was presented in (Chapter 4, Section 4.3, pp.66 - 77). The specification focused on two important meta-classes and their sub-classes which comprise the meta-class of linkable elements (GRLLinkableElements) and that of element links (ElmentLinks) (see Table 4.2, p.67).

Elements pertaining to model evaluation were not included since the evaluation elements apply to the model whereas our interest was mainly on the model construction process. In fact the idea of excluding these elements is that if the model can be formalised, then the mechanism to evaluate the formalised model can also be derived.

The main criteria for the construction of templates are:

- The formalised version of the GRL abstract model must keep the original (hierarchical) structure of the model. This was made possible by the use of the concept of inheritance in Object-Z at the class level.
- The specification was kept to the minimum. For example, primitives operations that are nomally included in class schemas to update the variables in the class were not included. The assumption was that if a class kept to the minimum (without some operations and even variables) can successfully be used to formalise a GRL element, then adding more information or details to it would not change the strategy but rather add more details to the final specification.

Since GRL abstract elements are themselves structured from meta-classes to subclasses, a top-down approach was adopted as the main formalisation strategy. Super classes/metaclasses were first considered followed by subclasses where Object-Z classes formalising sub-classes inherited those formalising the super-classes.

The formalisation of each model element was based on the analysis of the element (semantic, relationships, etc) and highly facilitated by the fact that Object-Z and UML used for the modeling of GRL abstract elements share a number of concepts in common including the concept of: the class, instantiation, inheritance, etc.

Complementary Non-Functional Actions: CNF-actions

To make our suggested CNF-action more visible, an abstract class was created that inherited the class of intentional elements. CNF-actions are not inherently part of any of the URN components, so the decision to integrate them into the model was to allow to GRL

specifications to benefit from such actions (to apply them when necessary to GRL models) and also to prepare them for the next software development phase.

Another advantage of our attempt to integrate CNF-actions into GRL abstract model, is that, it rendered the association of CNF-actions with other GRL elements, especially softgoals easier. A CNF-action became an intentional element similarly to tasks and resources and hence, allowing the use of the concept of class to formalise it as an an Object-Z schema (*ClsComplementaryAction*). Thus, in a specification of a softgoal, the list of all CNF-actions of the softgoal is simply defined as a subset of the set of objects of the class *ClsComplementaryAction*.

Future efforts

As observed during the validation phase, keeping the structure of the GRL model elements when creating templates may result in having a formal model that inherits the complex structure of the original model and which also may have negative effects on the quality of the system interfaces, as demonstrated during the animation. This may, if proper care is not taken during the refinement, introduce recursions into the code at the implementation phase.

- Having successfully ensured that templates developed with the above criteria can indeed be used to formalise GRL models, the next move would consist develop templates that do not necessary conserve the structure of the original GRL model but rather focus on optimizing the final formal models.

- Another important aspect to focus on is to develop approaches, guidelines, and or techniques to analyse and generate CNF-actions at each software development phase.

The next contribution to discuss is the framework proposed to guide the formalisation of the specification of an input GRL model.

7.4.2 Framework to formalise an input GRL model

This work proposed in Chapter 4, Section 4.4, pp.77 - 88, a framework to guide the Object-Z specification of an input GRL model. Two alternatives transformation approaches were first discussed and the direct transformation was recommended. The framework insists also on GRL model traversal strategy to identify GRL elements to be formalised (see Section 4.4.1, p.78), as well as the specification approach for each type of elements (see Sections 4.4.2 and 4.4.3, pp.79 - 83). The framework also recommends to update the specification in the light

of element links (see Section 4.4.4, pp.84 - 86) with guidelines provided to analyse each type of link element.

7.5 Contributions associated to the case study

The case study used in this work to support our ideas is presented in Chapter 5, pp.91 - 121. The contributions that were made during the case study analysis include the followings: an approach to assist with scope definition and requirements analysis, algorithms, and the formalisation of the GRL model for the case study.

The suggested strategy to assist with scope definition and requirements analysis is first presented.

7.5.1 Suggested strategy for requirements analysis

GRL inputs are generally expected to be stakeholders' intentions that can be specified using GRL model elements such as softgoal, goal, task, resource, and belief. However, as in the case study in this work, the reality is not always as expected; the initial requirements or problem description may require a preliminary analysis to define the scope of the system and identify or determine stateholders' real intentions.

Enterprise architectures are commonly accepted as standard to bridge the gap between the realities within enterprises and computer systems. However, the high cost and efforts required to develop an enterprise architecture do not make such solutions easily accessible to small and medium enterprise. Thus, our suggestion to exploit information, generally readily available in companies' organograms to fill the gap. The approach provides for guidelines to construct and model an organogram as a graph. Three algorithms were proposed to manipulate the graph. The graph model representing an organogram together with the algorithms were formalised with Object-Z and animated with Prolog aiming to show how the model can be used to facilitate scope definition and requirements analysis.

7.5.2 Suggested algorithms for graph manipulation

The three algorithms proposed to manipulate graphs are the followings:

Algorithm 3: The main algorithm

The main algorithm provides inputs to the two other graph traversal algorithms and coordinates their execution from the beginning to the end.

Algorithm 4: The horizontal search

The horizontal algorithm scans the graph horizontally, from left to right, to identify and process the relevant information necessary to resolve the problem at hand.

Algorithm 5: The vertical search

This algorithm scans the graph vertically, from top to bottom, to identify and process information necessary to resolve any problem at hand.

Contributions made when applying the formalisation framework to specify the GRL model of the case study are presented next.

7.5.3 Applying the framework to the GRL model of the case study

The framework was successfully used to formalise the GRL model developed for the case study. The correctness of the Object-Z specification generated for the case study, was extensively discussed in Chapter 6.

Algorithm 1

This algorithm summarises the GRL model transformation process introduced in the framework and structure the different parts of the formalisation process (see Algorithm 1, p.85). The algorithm scans an input GRL specification and performs multiple tasks including the followings:

1. ***templates***: ensures that there is a template for each abstract element, otherwise create one.

2. ***GRL traversal strategy:*** prescribes three traversals of the input GRL specification. One to identify the actors, the other to identify free elements (not bound to any actor) and actors' elements (bound to actors), and the third traversal to identify link elements. It is important to observe that the emphasis is not necessary on the number of times the GRL spec is to be traversed but instead on the different type of outputs that are expected.

3. Instructions are given on how to use the template to specify each of the following three types of GRL elements: ***Actor's definition***, ***Task***, and ***Resource***.

4. The class schema ***ClsGRLSpec*** is a special class that acts as the system interface through which all other classes can be accessed.

5. Update of the specification in the light of link elements (See Algorithm 2 presented next).

6. Instructions to finalise the specification.

Algorithm 2: Updating the specification

The algorithm, presented on page 87, provides for more detailed instructions to guide the analysis of each type of link element to update the specification in construction. One of the advantages of this algorithm is to prepare the system for appropriate application of CNF-ctions to achieve non-functional and quality requirements. This is done by affecting objects specifying softgoals into the right class schemas.

Example 7.5.1. Consider the softgoal *ImprovAppProc* (Improve Application Process) in the GRL model of the case study (see Figure 5.2, p.106). This softgoal is decomposed into the softgoal *FlexibleProcess* and the goal *ReduceAppTime* respectively with the links $dc1$ and $dc2$ (see Table 5.5, p.109). The Object-Z specification of the source softgoal (*ImprovAppProc*) is *ozImprovAppProc* and those of the two others are, respectively, *ozFlexibleProcess* and *ozReduceAppTime* (see Table 5.6, p.110).

1- *ozFlexibleProcess* (is not a class schema): refinement/decompositions branches from the softgoal *FlexibleProcess* are:

   (a) $\langle$*FlexibleProcess*, *AccessOwnApp*$\rangle$ leads to task: *AccessOwnApp* which Object-Z specification is *ClsAccessOwnApp*,

   (b) $\langle$*FlexibleProcess*, *AppOnline*, *SubmitAppOnline*$\rangle$ leads to the task *SubmitAppOnline* which Object-Z specification is *ClsSubmitAppOnline*,

   (c) $\langle$*FlexibleProcess*, *AppOnline*, *GenFormSections*$\rangle$ leads to the task *GenFormSections* which Object-Z specification is *ClsGenFormSections*, and lastly,

   (d) $\langle$*FlexibleProcess*, *AppOnline*, *Internet*$\rangle$ involving the dependency link dp1: (Applicant:ApplyOnline,Free:Internet) that is treated is step 1 when processing dependency links.

   As recommended in the last part of **Step 2** of the algorithm, *ImprovAppProc* is added to the variable named *nfdependers* of each of the following Object-Z class schemas: *ClsAccessOwnApp*, *SubmitAppOnline*, *GenFormSections*.

2- *ozReduceAppTime* (is not a class schema): a similar analysis as in (1), additionally to some of the refinement branches found in (1), the following branch is also to be considered:

⟨*ReduceAppTime*, *AppOnline*, *MotivEvalOnline*, *SubmitMotivEvalOnline*⟩

the first task on the branch is *SubmitMotivEvalOnline* which Object-Z specification is *ClsSubmitMotivEvalOnline*. *ImprovAppProc* is also added to the variable named *nfdependers* of the class *ClsSubmitMotivEvalOnline*.

3- In the same vein, the two contribution links from *AccessOwnApp* and *AppOnline* to *FlexibleProc* are analysed by applying Step 3 of the algorithm. This results in adding the softgoal *ozFlexibleProc* to the variable *nfdependers* of the following class schemas: *ClsAccessOwnApp*, *ClsSubmitAppOnline* and *ClsGenFormSections*.

4- Following the same analysis approach, the execution of Step 1 of the algorithm with the following dependency link:

dp1: (Applicant, ApplyOnline), Internet, (Nil, Nil)

results in adding both *ozImprovAppProc* and *ozFlexibleProc* to the variable, *nfdependers* of the class *ClsInternet*.

Important note: Having, for example, an object *ozA* specifying a softgoal A in the variable *nfdependers* of the class *ClsB*, indicates that the softgoal A relies on B to be satisfied. In that case, *ClsB* becomes the prime target of CNF-actions for *ozA*: the minimum application domain.

Example 7.5.2. From the Example 7.5.1, the minimum domain for CNF-actions for each of the two softgoals *ImprovAppProc* and *FlexibleProc* are:

Domain *ImprovAppProc* is:
{*ClsAccessOwnApp*, *ClsSubmitAppOnline*, *ClsGenFormSections*, *ClsSubmitMotivEvalOnline*, *ClsInternet* }

Domain *FlexibleProc* is:
{*ClsAccessOwnApp*, *ClsSubmitAppOnline*, *ClsGenFormSections*, *ClsInternet* }

These are system components or elements on which CNF-actions may be applied to improve either the application process (*ImprovAppProc*) or to render the application process more flexible (*FlexibleProc*). Depending on the relationship of these elements with their environment or other components in the system, the domain may be extended to include more elements.

These two examples 7.5.1 and 7.5.2 justify why only *ozImprovAppProc* and *ozFlexibleProc* were considered in Example 6.9.8, Figure 6.13, p.184 when constructing the UCM model of the Object-Z specification. They equally justify the application domain indicated noting that the domain was extended to include the class schema *ClsApplicant* and that at the time CNF-actions were created, two domain's elements were not yet available: *ClsGenFormSections* and *ClsSubmitMotivEvalOnline*.

7.5.4 Areas that require some improvements

- Design an intelligent system to implement the organogram approach,
- Optimize the algorithms and study the possibilities to associate heuristics,
- Test the proposed framework on a couple of known (failed/suceeded) industrial projects.

Contributions made during the validation are discussed next.

7.6 Contributions associated to the Validation

The validation presented in Chapter 6, pp. 123 - 185 is a multipurpose operation carried out in this work to examine different facets of the proposed formalisation framework as well as the qualities of the resulting formal specification. A number of contributions were made amongst which the idea to use Prolog to animate an Object-Z specification discussed next.

7.6.1 Animating an Object-Z specification with Prolog

The approach proposed in this work to animate an Object-Z specification with Prolog is presented in Chapter 6, Section 6.2, pp. 125 - 127. Seven rules are suggested to transform an Object-Z specification into Prolog and three guidelines provided for a proper use of the rules and to monitor the animation process.

The strategy was successfully used in Chapter 5, Sections 5.2.3 and 5.2.4, pp.100 - 103, to animate the organogram model and hence facilitate the scope definition and problem

analysis. As indicated in Table 6.1, p.129, the proposed animation strategy was largely used in Chapter 6 to animate the Object-Z specification of the case study and thereby contributing to further illustrate the correctness and the completeness of the specification.

7.6.2 Application of the four-way framework for validating a specification

As shown in Figure 6.1, p.128, the four-way framework for validating a specification proposes guidelines to iteratively examine a software specification and correct errors until the desired quality is obtained. Applying only one iteration would also help to determine the status of the specification at some development. Another advantage of the framework is that it facilitates the planning and monitoring of the validation over four different phases whereby each phase focuses on one important aspect of the specification. Thus, before the validation, it stimulates thinking about the properties of the specification that ought to be verified, the techniques and methods to be used, as well as tools needed for the validation.

The framework is opened about properties, methods, techniques and tools needed for the validation of a specification. That is why a successful application to a specific type of specification is a contribution to the field because it provides for more examples of properties required to ensure the quality of the specification and more importantly, associates to those properties examples of successfully used techniques, methods and tools to validate the selected properties.

Figure 7.3 illustrates the complete validation process adopted in this work. The overall planning of the validation is discussed in Chapter 6, Section 6.4, pp. 128 - 135 and summarised in Table 6.1. p. 129. During the planning phase, each identified property to be validated is defined and the validation approach, including the technique/method and tools clearly indicated. The planning of the animation is adapter from the prototype process in Figure 6.2, p. 136 proposed by Sommerville [169] and fully presented in Chapter 6, Section 6.5, pp. 135 - 147.

In Figure 7.3 above, the arrows from the rectangles illustrating the planning to the validation phases indicate the repartition of the validation tasks per phase. The arrow from one phase to another indicates the (cyclic) order in which the validation was performed with the results from one phase re-usable in the next one thus, avoiding to repeat exactly the same activity in two different validation phases.

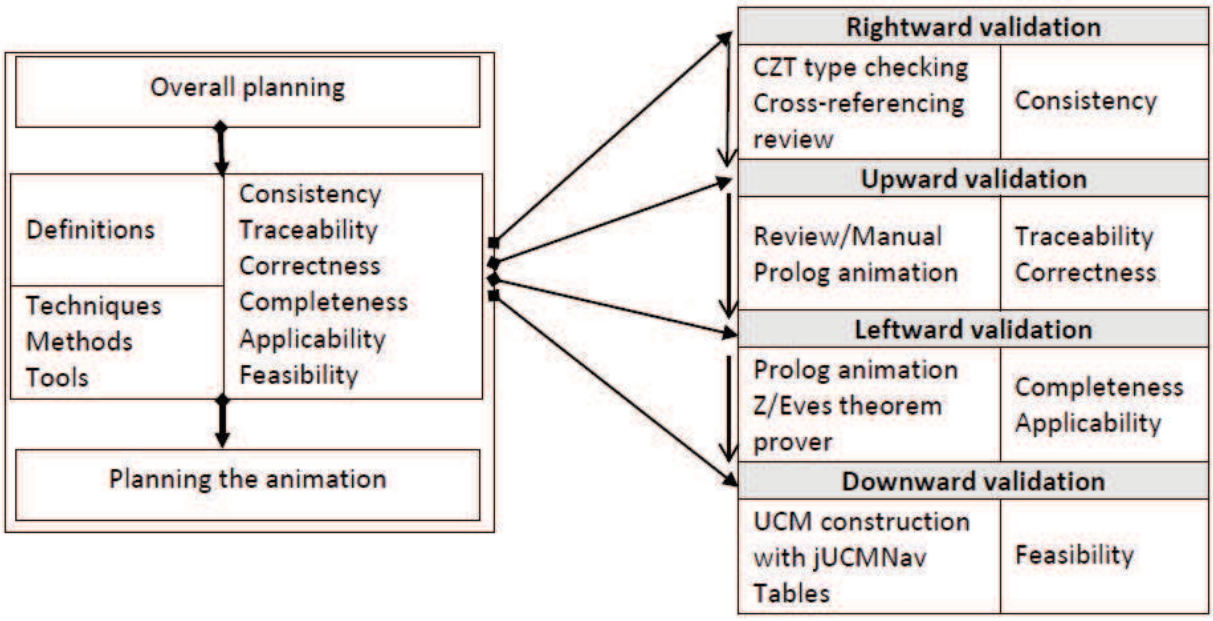

Figure 7.3: Summary of the complete validation process

7.6.3 Automated proofs

To establish the consistency, or more precisely to detect and correct language related errors in the Object-Z specification of the case study discussed in Chapter 5, Section 5.4, pp. 107 - 121 and fully presented in Appendix C, the CZT tool was used to parse and type check the specification. This exercise was indeed a normal activity however, together with the approach adopted to address issues raised, it constitutes a good learning example. More important was the use of the theorem prover Z/Eves to assist with the proof of the applicability of the specification

Considering the basic formalisation approach proposed in this work and illustrated in Figure 7.2, p.192, the real question that ought to be resolved was:

can all possible GRL models be formalised using this strategy?

The approach adopted to address the question was to consider a domain consisting of all possible GRL models, all possible Object-Z specifications and find the condition under which for a given GRL model there exist an Object-Z specification that formalises it. The domain was therefore formalised with the Z notation as well as the formalisation strategy as an operation that transforms GRL models into Object-Z specifications. The Z specification was parsed and type checked with Z/Eves. The theorem prover was also used to calculate the precondition for the formalisation operation the result of which confirmed that the condition for the input GRL specification to be formalised is for each conceptual element used in the specification to be formalisable; that is to be transformable into a template.

This result ensures that if more properties or constraints are discovered in the domain, they can simply be added to the Z specification and the theorem prover will be able to automatically re-calculate the pre-condition.

The main purpose of the validation process covered in Chapter 6, pp. 123 - 185 was to provide for sufficient supporting arguments to decide on the qualities of the Object-Z specification as well as the specification process. Based on the validation, those qualities are therefore discussed next.

7.6.4 Qualities of the formal specification

The specification is consistent

During the rightward validation phase, the consistency of the Object-Z specification was intensively discussed. With the type checker czt, more than 98% of language related errors detected could be corrected and the remaining 2% presented no serious risk to the overall quality of the specification. The inspection of individual class schemas as well as the specification as a whole also confirmed the absence of inconsistencies between class schemas and within each individual class.

As illustrated in Figure 6.4, p. 142, the inspection of the entire specification showed that the specification may have a complex structure due to the multiple levels of hierarchical inheritance of class schemas derived from the very structure of the GRL conceptual elements being formalised. However, it was also illustrated through the Prolog animation that the negative effect of such a complex structure can be eliminated during the development.

The specification is traceable

A twofold traceability analysis was conducted in Chapter 6, Section 6.7.1, pp. 152 - 155 during the upward validation phase. The analysis shows that all Object-Z elements specifying the GRL conceptual elements are clearly traceable from those elements and that each element was clearly specified (see Table 6.7, p. 153). As shown in Table 6.8, p. 154, the Object-Z specification was also shown to be traceable from the input GRL model of the case study and more importantly, the input GRL was fully specified.

The next point of interest is to decide on the correctness of the specification.

The specification correctly addresses stakeholders' needs

The correctness of the Object-Z specification was analysed in Chapter 6, Section 6.7.2, pp. 155 - 167 during which different aspects of the specification were examined and some selected components animated. Based on the traceability analysis, it was concluded that the Object-Z specification fully specifies the input GRL model. It was also illustrated by analysing the specification process that the specification fully describes the relationships between the elements of the GRL model. In the same vein, the Prolog animation of the specification was exploited to establish that each GRL element was correctly specified. The animation of the state of the class schemas was conducted in Chapter 6, pp. 138-146, 159-162 and Appendix D, pp. 267 - 277 to show that the specification conserves the properties of the element being specified and the animation of the operations was also conducted in Chapter 6, pp. 159-166 to show how operations were also correctly described.

The specification is complete

The completeness of the specification was concluded based on the traceability analysis conducted in Chapter 6, Section 6.7.1, pp. 152 - 155 and summarised in (Table 6.7, p.153 and Table 6.8, p.154), the animation discussed above, and a manual review of the specification performed in (Chapter 6, Section 6.6.2, pp.150 - 152).

The formalisation process is applicable to all instances of GRL model

The validation of the applicability of the formalisation process was conducted in Chapter 6, Section 6.8.2, pp. 167 - 171 and fully discussed in detail in Chapter 7, Section 7.3, pp. 190 - 193 the result of which indicates that the formalisation of a GRL model, solely depends on the ability to generate templates: the formal specification of the GRL conceptual elements. From this result, it is therefore, concluded that the proposed Object-Z formalisation process has the ability to formalise any instance of the GRL model.

An operational system can very well be generated from the Object-Z specification

The operational feasibility of the specification, indicating the ability to transform the specification into an operational system, was established in the previous chapter in Section 6.9.1, pp. 172 - 184 by following the same construction approach to build two UCM models one from the GRL model of the case study and the other from the Object-Z specification of the same GRL model.

The analysis of the two UCM models clearly reveals that the one obtained from the formal specification is more elaborated, includes more details and inherently integrates, through the

construction process, mechanism to perpetrate the influence of CNF-actions to the resulting UCM map.

7.6.5 Qualities of the formalisation process

The GRL formalisation process proposed in this work that includes the framework developed in Chapter 4 together with the associated algorithms, present a number of characteristics among which the following three qualities are worth discussing: the traceability, the ability to be automated and the ability to be generalised.

The process facilitates traceability

It is important to observe that the traceability of the Object-Z specification normally derives from the ability of the formalisation process to readily prepare, during construction, elements of the formal model to be traced back from the GRL model. Next are two illustrative examples.

Example 7.6.1. Table 4.5 on page 82 which main objective is to indicate for each type of input element, the action that has to be performed on the template to formalise the element, also readily prepare the templates in the final specification to be easily traced back to the GRL conceptual elements and other specification elements to be traced back to templates.

Example 7.6.2. Table 5.6 which is built to aid the construction process readily prepares components of the final specification to be easily traced back from the input GRL model.

This quality is important because it makes it possible for the process to contribute to the building of an interactive environment whereby, for example, the GRL model and its formal specification are interactively created.

The process can be automated

Although it is not formally proven, the assurance for this quality stems from the experience gained by successfully applying the process to the case study. The other two factors that support the idea are: the status of the formal specification in Appendix C and the modularity of Algorithms 1 and 2.

As represented in Appendix C, the Object-Z specification of the GRL model of the case study is still in construction. Let's illustrate with the following example:

Example 7.6.3. Consider the class *ClsActorApplicant* presented in Chapter 5, p. 114. The operations: *NewSoftGoal*, *NewGoal*, *NewBelief*, and *Addnfdepender* of this class are all useful only during the construction process. When, for example, all the softgoals, goals and beliefs contained in the actor definition, namely, *Applicant* and their relationships with other elements of the GRL model are processed, these operations becomes useless and a new class schema without these operations can therefore, be created with updated components to specify the actor's definition *Applicant*. Thus, the successful animation of this class in Chapter 6, also shows the ability to implement the formalisation process. However, a closer at the two main algorithms for the process reveals more evidences.

The modularity of the two algorithms: Algorithms 1 and 2, is another factor that one can rely on to show that the specification process can well be implemented. Let's consider, for example, the first five steps or modules of Algorithm 1.

Algorithm 1

Module 1 is about creating templates. Since templates are created only once, the module can be implemented (independently of others) and be updated only when the GRL conceptual model has also been updated.

Module 2 is about scanning an input GRL model to identify all its elements. The jUCMNav is an example of a software that successfully implement this module to perform, for instance, an evaluation or to export the model into another format.

Module 3 is about creating an Object-Z class schema; many examples of software that create Object-Z exist. In the same vein, Modules 4, 5 and 6 are also about creating an Object-Z.

Module 7 is about executing Algorithm 2.

Algorithm 2

This algorithm is mainly about examining each GRL link element (stored in a table, e.g., Table 5.5, p.109) to make appropriate processing decision based on the type of the link element. The algorithm is subdivided into three modules where each module examines each type of link element. Since the algorithm involves mostly the manipulation of data stored in a table, the question of whether it can be automated or not becomes obsolete.

The process is generalisable

This quality was fully discussed in (Section 7.3.2, pp.191 - 193) where it was demonstrated that the formalisation strategy is not Object-Z dependant. According to the analysis, in theory, any formal method that satisfies the following two conditions can very well be used to formalise an input GRL model:

Considering the formal notation X, the condition for formalising a GRL model into X are:

- the ability of the method X to formalise GRL conceptual elements, namely templates (element type),
- the ability of the method X to formalise elements of the input GRL model from the templates.

7.7 Chapter conclusion

This chapter has presented the discussion of the contributions made per main topic developed in this work. An analysis of the URN construction process leaded to the conclusion that a UCM model is a refinement of GRL model and a more consistent link/relationship between the two models needs to be established. Another important contribution is the concept of Complementary Non-Functional Actions (CNF-actions) to allow the development of non-functional requirements alongside the functional ones throughout the complete software development cycle. The proposed GRL formalisation process was also discussed to have among others, three important qualities which are the ability to prepare the final formal specification to be traceable from the initial requirements, be automated and is generalisable.

To generate a GRL model for the case study, an approach exploiting enterprise organograms to facilitate goal analysis was proposed; three algorithms were also derived to manipulate graph modeling the organograms. The Object-Z specification of the GRL model of the case study, resulting from the suggested formalisation process, was shown to be consistent, traceable from the initial GRL model, correct, complete and operationally feasible. To validate

these specification's properties, the four-way framework for validating a specification was adopted and a strategy derived to animate Object-Z specifications with Prolog.

The discussion of each contribution in this chapter was conducted independently from each other. the next chapter combines them in a way to respond to the research questions.

Chapter 8

Summary of main findings, conclusion and future work

Most of the contributions made in this work were fully analysed in Chapter 7, as well as possible generalisation of some of the proposed ideas. This chapter purposes to derive from that analysis, a summary of the main findings and relate them to the research problem. In a short discussion, it enlightens the extent to which the research questions, posed in Chapter 1, were addressed. The chapter is concluded with some future potential research areas requiring attentions to complement the findings.

8.1 Main findings

This study is part of a broader research area of combining semi-formal and formal software techniques. The complex problem of integrating formal methods into the existing traditional software processes was investigated with the focus on Z/Object-Z as formal technique and URN as a semi-formal one. The goal being to develop strategies to formalise goal models specifying both functional and non-functional requirements, derive mechanisms to perpetrate, beyond their operationalisation, the influence of non-functional requirements on further software models hence improving their qualities.

To this end, the first research question posed was the following:

RQ1 : **what lightweight (enterprise) model could there be to facilitate the process of goals and/or requirements elicitation at an initial phase of requirements elicitation and analysis?**

This question is interested in: where the goals/requirements that are to be formalised come from and how they are extracted from their original sources. This requires the knowledge

of enterprise models [128, 145, 181]; which are generally known to be appropriate sources of goals and requirements (see for example, [55, 104]), as well as the existing elicitation process. The bulk of the work done, by the researcher, to address the question was published in [66] the result of which was applied to the case study in Chapter 5. We proposed an approach for establishing guidelines to construct enterprise organograms in a bottom-up fashion and transform these into useful models that can be exploited in goal and requirements elicitation phases to identify vital sources of information within an entire organisation. Our approach also proposes strategies, including three formulated algorithms, to manipulate the model and derive the necessary information in a simplistic manner.

The main benefit of the proposed method stems from the simplicity and availability of enterprise organograms to which appropriate information may be cautiously added to construct flexible and lightweight enterprise models vital to goal and requirement elicitation. Such a model therefore achieve objective RObj 1 defined in Chapter 1 on page 7. Having developed our lightweight model, the next important step in the process requires the identification or construction of a goals/requirements modeling and analysis approach that may take advantage of the model. Thus the next research question:

RQ2 : To what extent could the URN model construction process take advantage of any lightweight model that would result from the research question 1?

This question requires in the first place an investigation of the literature to identify any existing construction process. But also more importantly, a detail analysis of such a process to bring forth any of its weak points that need to be improved. Such weak aspects would serve as measures to evaluate the contribution of the integrated formal technique to improve the quality of the process.

The basic top-down URN construction process presented in Chapter 2 is generic to GORE methods that follow, for most of them, the same Softgoals Interdependency Graphs(SIG) construction principles rooted from the NFR framework (Chung and Nixon [50], Merilinna et al. [122], Mylopoulos et al. [137]). One of the particularities of URN stems from the need to combine the construction of GRL and UCMs for which an iterative and inter-related process (for constructing GRL and UCM models) was proposed by Liu and Yu [110]. A detailed analysis of the process presented in Chapter 4 leaded to the following conclusions:

- ***UCM refines GRL***.
  Although a UCM map can be created at different abstraction levels (this depends on the level of details), for the map to be justified, each of its elements/components must be linked to some goal/softgoal in a GRL model. Otherwise, it would be reasonable to

ask why a software engineer would want to add to a software the functionalities that do not support neither business objectives nor any of the stakeholders' intentions.

- ***There is a gap between GRL and UCM***.
  We provided few examples in Chapter 4 to illustrate how this gap is conceptual. UCM has a lot more concepts than those needed to model the three types of inputs from GRL, namely, beliefs, resources and tasks. For example, non-functional requirements are known to be critical inputs to guide system architecture [149] however, they are not (directly) part of the inputs from GRL. This makes it difficult to trace UCMs' elements specifying the system architecture from the initial GRL model.

  To our view, this gap is also an indicator of the weakness of GORE methods; their inability to fully operationalise, decompose or refine softgoals, especially those describing NFRs.

This analysis also constitutes an important step towards achieving the second objective of this research (RObj 2). Knowing that UCM refines GRL indicates the ability of URN to be expended in different manners. For example by integrating other techniques or methods in between GRL and UCM which can consequently contribute to fill the gap between the two modeling techniques. As argued in Chapter 5, the gap between UCM and GRL leads to traceability issues for which Merilinna et al. [122] suggested their NFR+ framework tool to manage the entire requirements specification and design process.

In this work we believe in the benefit of extending the analysis of NFRs beyond the traditional operationalisation/decomposition/refinement proposed in GORE methods thus, the reason for the next research question.

RQ3: To what extent are goals describing non-functional requirements formalisable?

The initial idea was to formalise NFRs embedded in GRL models. However, due to the fact SIG own their value to the use of elements such as goals, tasks, resources and beliefs to refine softgoals, isolating the softgoals and formalising them without considering those elements would simply result in a loss of vital information in the formal model. That is why the entire GRL model was considered in this work with the focus on softgoals. This question is in fact the most challenging one since it is about analysing and formalising NFRs with Z/Object-Z that does not naturally integrate mechanisms for describing NFRs.

As illustrated in Figure 4.3, a process-oriented analysis of NFRs was performed in Chapter 4

based on the idea that the development of NFRs should be extended beyond the refinement proposed by GORE methods and further analysis considered at each software development phase. We have therefore suggested the concept of Complementary Non-Functional Action (CNF-action) that allows the software developers to continuously think about the actions to be taken to propagate the influence of NFRs at each development phase from requirements specification to the implementation. CNF-actions include actions performed on intermediate models to improve the quality, as well as important decisions regarding for example the selection of appropriate techniques/methods or tools at given phase. The concept of CNF-action was applied to construct the UCM in Figure 6.13.

Regarding the formalisation of GRL models, an approach was proposed whereby an Object-Z class schema, called template, is created for each type of GRL conceptual element including CNF-actions. The formalisation of an element in a GRL model is therefore obtained by instantiating the template for the element being specified or by creating new class from the template by inheritance. A framework was developed to guide the formalisation process from which two algorithms were derived. Figure 7.2 on page 192 illustrates the complete formalisation process presented in four phased after detailed analysis. The approach was successfully applied to the GRL model of the case study to produce an Object-Z specification. The suggested CNF-actions concept has therefore provided for means to address the research objective RObj 3. In the same vein, the proposed formalisation framework successfully applied to the case study in Chapter 5 has very well contributed to the achievement of the fourth research objective (RObj 4).

The approach hence summarised constituted the first step towards addressing the research question. However, specific concerns were equally the target of the study. Thus, the following sub-question:

(a) ***To what extent can a GRL model, describing both functional and non-functional requirements, serve as input to a formal specification techniques, case of Z/Object-Z?***

As discuss in Chapter 7, Section 7.3.2, the proposed formalisation strategy was specified with Z and the Z/Eves theorem prover applied to the generated Z specification of the approach to determine the precondition for a GRL model to be formalised (see Chapter 6, Section 6.8.2, pp. 167 - 171). It was therefore formally illustrated that, with the proposed method, any GRL model can be specified with Z/Object-Z. The only two important conditions for a formal notation to be able to formalise an input GRL model, as generated by the Z/Eves theorem prover, are:

- the ability of the method to generate a template for each type of GRL elements, and
- the ability of the method to formalise elements of the input GRL model from the templates.

The next sub-question:

(b) ***To what extent is it possible to formalise a goal model describing both functional and non-functional requirements in Z and Object-Z?***

The GRL formalisation strategy proposed in this work was successfully applied to the GRL model of the case study to produce the Object-Z specification presented in Chapter 5, which full version is in Appendix C. To specify a softgoal, an Object-Z class schema was first created for CNF-actions (complementary non-functional actions). In the class template for softgoals, a component containing the list of CNF-actions for the softgoal at the specification phase was added. With each element of the list representing an object of the following class schema for CNF-actions:

ClsComplementaryAction

$\lceil(id, name, mdata, phase, action, domain, option)$

ClsIntentionalElement

$phase : ClsPhase$
$action : ClsAction$
$domain : ClsDomain$
$option : Description$

This made it possible to use the same approach to formalise all GRL elements, including goals, softgoals, tasks, resources, beliefs, and links. The validation carried-out in Chapter 6, pp.123 - 185 and discussed in Chapter 7, Section 7.6.4 revealed the qualities of the Object-Z specification of the case study from different perspectives and thence indicating the success of the formalisation process. However, some qualities of the specification could very well stem from the input model being formalise thus, the reason for the next research question:

RQ4: What would the impact of a semi-formal modelling technique/method for non-functional requirements be on the process of constructing a formal specification?

The Object-Z specification of the GRL model of the case study was shown to embed some good qualities. For example, the specification is consistent, correctly and completely addresses the stakeholders' needs and an operational system could be obtained from the specification as well. Each of the Object-Z's elements can be traced back from the initial GRL

model and vis versa. Some of these qualities such as the traceability are attributed to the formalisation process however, the contribution of the input GRL model is considerable.

Before the formal specification, the initial goals/requirements are elaborated, classified or categorised, prioritised, modeled, analysed and evaluated. For example the semi-formal technique readily structures the requirements, provides for means to render them distinguishable and links to relate them one to another. We argue that this preliminary work evidently contributes to the quality of the formal specification and to make the formalisation process more flexible. This maybe well apprehended if we consider the situation where goals and/or requirements are formalised immediately after they are identified.

One aspect of the influence of the initial model on the formal specification was spotted out during the validation of the specification. It was shown that the structure of the Object-Z specification in Figure 6.4, p. 142 was very similar to that of the GRL conceptual model in Figure 4.6, p. 69 from which Object-Z templates were created. This evidence of the impact of the initial model on its formal specification highly contributes to the achievement of the sixth research objective (RObj 6).

RQ5: To what extent can a formal specification of a GRL model help to improve on the process and quality of URN models?

During the discussion of the operational feasibility of the formal specification in Section 6.9.1, pp. 172 - 184, the same construction approach was followed to build two UCMs; one directly from the GRL model of the case study and the other from the Object-Z specification of the same GRL model. It appeared that the UCM map obtained from the formal specification was more elaborated than the one generated directly from the GRL model. The map included more details and inherently integrated, through the construction process, mechanism to perpetrate the influence of CNF-actions to the resulting UCM. This observation may not be enough to show the full potentials of the formal model to improve the quality of the URN, nevertheless, it addresses the fifth research objective (RObj 5) and opens perspectives for elaborating on the topic.

8.2 Concluding notes

One may ask at this point if the research problem was solved. The answer may not be straightforward however, a lot has been done towards achieving the final goal which is to integrate formalism into URN without decreasing its flexibility and usability which are important factors for the growing adoption of the method in industry and even in academia.

Figure 8.1 summarises most of the work done so far in this direction and presents the structure of the improved URN we aim to develop with Object-Z serving as an intermediate technique between GRL and UCM.

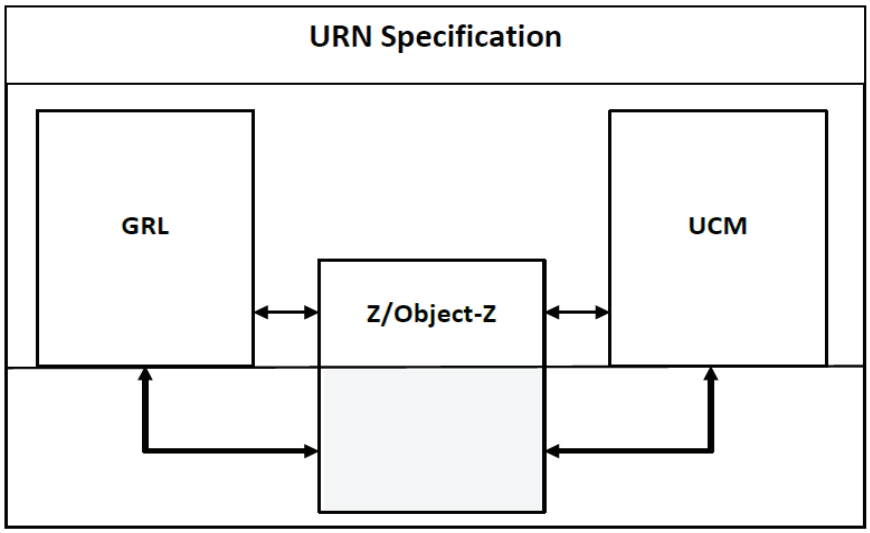

Figure 8.1: An approach to integrate Object-Z into URN process

8.2.1 The process from GRL to Object-Z

This work has proposed means to transform GRL models into Object-Z specifications. The process was proven to allow forward and backward traceability between the GRL model and the constructed Z/Object-Z specification. It was also shown that the process can very well be automated. However, the degree of automation is still to be determined. In Figure 8.1, the shaded part of the Z/Object-Z module, in Figure 8.1, is a graphical representation of whatever can be automated and hence hidden to users. The main objective being to hide as much difficulties inherent to formal techniques as possible and present to users an interface, to the best, as flexible as that of URN itself.

The GRL formalisation framework follows what we called a direct transformation where the input is the whole GRL model. However, most of the operations within the framework (see Algorithms 1 and 2) treat GRL elements individually. Such individual treatment of the elements of the input model makes it easier to complete the formalisation process such as to allow inter-active transformation of each element during the model construction.

8.2.2 The Object-Z specification

Most of the contributions of this work discussed in Chapter 6 and 7 aimed to evaluate the Object-Z specification, as well as the formalisation process. The specification was therefore argued to embody some good qualities including the ability to serve as input to UCM making it possible for the semi-formal notation to benefit from the precision of mathematical operations that may be operated on the formal model. For example, preconditions needed in UCM can be calculated on the input Object-Z specification and simply transferred to UCM.

Another useful aspect is that each class schema of the specification includes the component, *nfdependers*, containing the list of all the NFRs that depend on the class to be satisficed. Consequently, UCM elements resulting from the transformation of the Object-Z class are potential targets of CNF-actions of those NFRs. The benefit of this is huge since the analysis of the causalities between NFRs and software design, particularly architectural design, has long been the concern of numerous research work (Examples: [8, 44, 81, 103, 107, 196]).

8.2.3 From Object-Z to UCM

To reinforce the existing URN process [110], we proposed in Section 6.9.1, pp. 172 - 184 the use of a table (see Table 6.10) to record possible mappings between GRL elements and those of UCM. Owing to the traceability between GRL model and its Object-Z specification, similar mappings between the Object-Z specification and UCMs' elements (see Table 6.11) were adequately exploited to construct a UCM map from the formal specification.

Based on the UCMs in figures 6.8 and 6.13, it was argued in Section 7.1, p. 187 that the proposed UCM construction approach is more flexible, since for each GRL element or Object-Z component, the list of possible UCM elements to use is proposed and the designer may simply choose the most appropriate one. In addition, the approach reinforces the tracing of UCM elements from the source GRL model or Object-Z specification. For example, with the following function:

$$savecontext(\text{Appliquant}[StaticSTub(0), Team(1), Object(-1), Agent(-1), Actor(-1)])$$

one may record the identifier of the team component selected for the GRL actor *Applicant* and hence create a link between the two elements.

8.3 Future work

Challenges that need to be addressed in the future to complement this work are presented under two categories: generalisation (of concepts) and implementation (of the proposed approaches).

8.3.1 Generalisation of concepts

Extend the GRL formalisation strategy to other formal techniques

The strategy proposed in this work, to formalise GRL models with Object-Z, was argued to be generalisable to any formal technique that fulfils to conditions:

1. the technique has the ability to formalise GRL conceptual elements into Object-Z metaclasses, namely templates (element type),

2. the technique has the ability to formalise elements of the input GRL model from the templates.

to confidently confirm this generalisation idea, the two conditions should be applied to two or three other formal specification techniques.

Extend the GRL formalisation idea to other semi-formal methods

As illustrated in Figure 8.1, Object-Z is integrated into URN to feed the gap between GRL and UCM and to improve on the quality of the URN model, as well as the construction process. We have also argued that the main benefit is not just to transform GRL or UCM into Object-Z and vis versa, but rather to build an environment where the flexibility and usability of URN are conserved and the hard part of the formal technique hidden to the user. Many semi-formal methods propose more than one model at different abstraction levels that share very similar types of relationships that connect GRL to UCM. For example, with UML we have scenarios, use cases diagrams, class diagrams, etc.

The challenge here is to start developing strategies, as the one proposed in this work, to integrate formal techniques into existing, well adopted semi-formal methods, whereby the semi-formal techniques serve as interface to users with much of the difficulties inherent to formal methods hidden to them. Having semi-formal techniques empowered by the formal ones could be the beginning of a massive use of formal methods without the end users being (fully) exposed to the hard part of formal methods. In this regard, the hard part of formal techniques will remain the responsibility of Formal Methods experts.

Extend the idea of CNF-actions to other software development phases

The idea behind the concept of CNF-actions is to provide for means to extend the influence of NFRs beyond the normal analysis and evaluation performed by GORE methods. That is to continue developing NFRs in parallel with functional requirements from the initial phase to the production of the operational system. We have illustrated how the concept can be integrated into the Object-Z specification of a GRL model. We have also shown how CNF-actions could be generated to enhance UCMs. However, the challenge remains because similar strategies need to be developed for each software construction phase where approaches, guidelines or mechanisms to analyse, generate, model and propagate the influence of NFRs in the form of CNF-actions are to be created.

8.3.2 Implementation

Integrate formalism into the existing jUCMNav

To become utilisable, the strategy illustrated in Figure 8.1 needs to be implemented. Different designs may be experimented to determine the most appropriate solution. For example, one may think of re-engineering the existing jUCMNav to integrate the new functionalities pertaining to Object-Z, implementing the part on Object-Z as a plug-in to jUCMNav, an intelligent or expert system to guide the URN construction with jUCMNav.

Implement the organogram approach

To get the full benefit of the organogram approach to (software project) scope definition and problem analysis proposed in this work, a software system implementing the approach ought to be constructed. If well implemented, such a system can also serve as enterprise resources management system. In large companies, one may think of a distributed system where independent and inter-connected sub-systems are developed for different branches of the enterprise. Where for example each sub-system implements a sub-graph modeling the organogram of the branch and links between the sub-systems uniting the sub-graphs to form the one representing the organogram of the enterprise as a whole.

The successful animation with Prolog also prompts the idea of developing the sub-systems (or the entire system) as expert systems or to integrate elements of intelligent computing into the software product.

Optimise the algorithms

It would be of benefit to analyse the complexity of the algorithms proposed in this work with the purpose to optimise them. Studying the possibilities to extend the algorithms with heuristics may also bring in some benefits to the different processes that use the algorithms.

Appendix A

Additional results from the systematic literature review

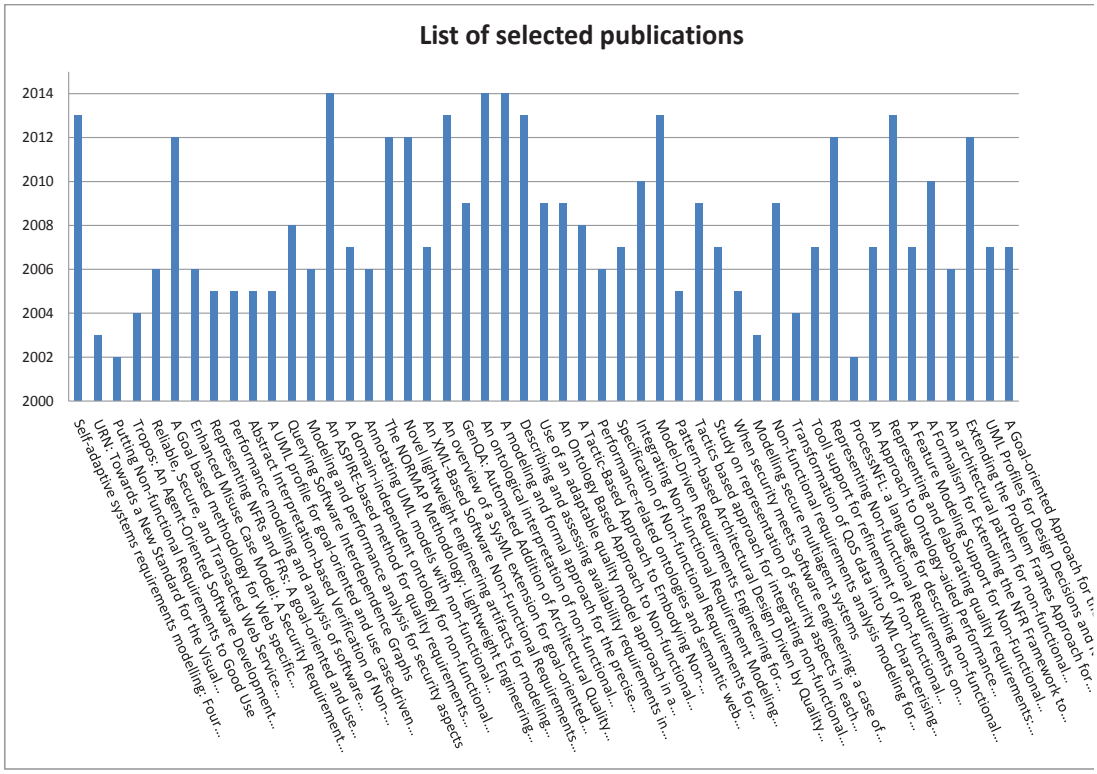

Figure A.1: List of the selected publications

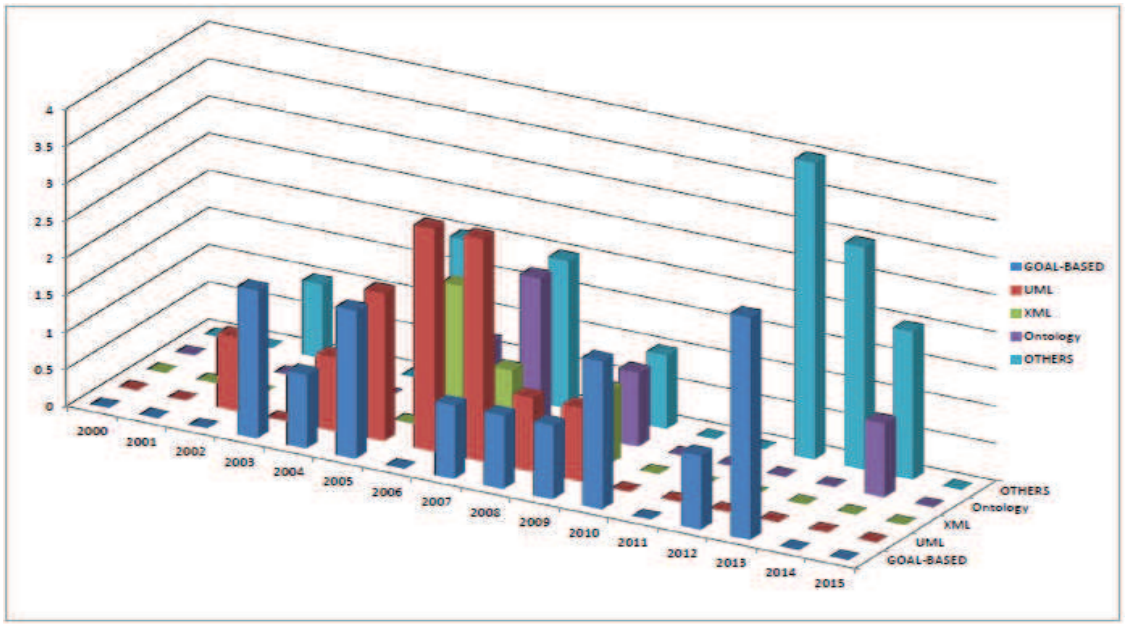

Figure A.2: List of publications per specification technique

Appendix B

Additional items for case study developed in Chapter 5

B.0.1 Business objectives

	Node's objectives		
Label	**Domain**	**Managment**	**Operational**
cd	College	Deanery	Deanery
cdob1	Develop research and postgraduate supervision capacity within the institution		
cdob2	Create and environment that encourages and foster the culture of research		
cdob3	Attract funding from industries and other external bodies		
cdob4	Develop a culture of professional teaching and learning through practice		
cdob5	Increase under-graduate and postgraduate success rates		
cdob6	Create an environment where students are encouraged to actively participate to industrial projects		
crdd	College	Research department	Director
crddob1	Timeously report on research funding for different programmes		
crddob2	Encourage as many academics and students as possible to apply for funding		
crddob3	Negotiate research funding and scholarships from industries		
crddob4	Attract new postgraduate students		
crdsp	College	Research department	Support programme
crdspob1	Provide maximum support to applicants		
crdspob2	Increase the number of applicants through advertisement		
crdspob3	Reduce the overall processing time of submitted applications		
crdrm	College	Research department	Research management
crdrmob1	Attract new research student by offering them specific programmes		
crdrmob2	Timeously report on research outputs		

	Node's objectives		
Label	**Domain**	**Managment**	**Operational**
crdrmob3	Develop research and innovation policies and incentives		
chrd	College	Human resources	Director
chrdob1	Define appropriate and time effective appointment policies		
chrdob2	Develop strategies to retain high quality staff members		
chrs	College	Human resource	Staffing
chrsob1	Appoint academic with PhD or those who are about to complete their PhD		
chrsob2	Optimize the overall appointment process		
chrsob3	Appoint quality admin and support staff with practical experiences		
csd	College	School	Director
csdob1	Increase the school research output by at least 5%		
csdob2	Increase the number of academics with PhD by 5%		
csdob3	Efficiently and timeously assist researchers applying for funding		
csdob4	Implement effective research support process		
csas	College	School	Academic staff
csasob1	Publish at least one journal article or two conference papers per year		
csasob2	Timeously motivate students' applications for scholarship and research funding		
csasob3	Supervise to completion at least one masters student in two years time		
csasob4	Increase the number of industrial projects for Masters and PhD students		
csad	College	School	Admin staff
csadob1	Reduce the time management and academics spend on administrative duties		
csadob2	Assist registered and prospectives students with general information		
csadob3	Assist with all type of document exchange within the department		
cscd	College	School/COD	COD's office
cscdob1	Increase the number of publications in ISI and IBSS listed journals		
cscdob2	Efficiently implement the policies on research and tuition		
cscdob3	Offer performing postgraduate students to become academics		
cscdob4	Encourage research students to attend conferences		
cscdob5	Implement strategies to motivate performing students and staff		
cscdr	College	School/COD	Research
cscdrob1	Report on research outputs at the school level		
cscdrob2	Organise and monitor research and supervision training workshops		
cscdt	College	School/COD	Tuition
cscdtob1	Report on tuition and staff development		
cscdtob2	Improve on individual lecturer's performance		

	Node's objectives		
Label	**Domain**	**Managment**	**Operational**
cscdce	College	School/COD	Community Engagement
cscdceob1	Increase the number of staff participating to community works		
cscdceob2	Implement strategies to render community works publishable		
cscsed	College	School/CSE	Director
cscsedob1	Efficiently implement policies on industry funding research		
cscsedob2	Improve the relationship with enterprises through quality teaching & research		
cscseel	College	School/CSE	Enterprise liaison
cscseelob1	Intensify discussions with enterprise regarding their needs and problems		
cscseelob2	Includes as many students as possible in discussions with industries		
cscseelob3	Increase the number of M & D industry sponsored projects to attract students		
cscsecc	College	School/CSE	Certificate course
cscseccob1	Maximise the number of short courses addressing industry needs		
cscseccob2	Increase the number of contrats for group training by at least 7%		
cscseccob3	Increase the number of courses for skill development in government		
cscsess	College	School/CSE	Support staff
cscsessob1	Timeously respond to all type of maintenance needs		
cscsessob2	Provide maximum support in tuition through labs works and programming		
cfd	College	Finances	Director
cfdob1	Report on all financial matters within the institution		
cfdob2	Efficiently implement policies on remuneration and research funding		
cfr	College	Finances	Renumeration
cfrob1	Timeously compute and pay staff salaries		
cfrf	College	Finances	Research funding
cfrfob1	Optimize the process of releasing funds especially for conferences		
cfrfob2	Timeously release scholarships to benefiting students		

Table B.1: Business objectives

B.0.2 Vertical relationships

	Vertical relationships		
Label	**Domain**	**Managment**	**Operational**
cd	College	Deanery	Deanery
cdob1	crddob1, crddob4, chrdob1, chrdob2, csdob2		
cdob2	crddob2, csdob1, csdob3, csdob4, cfdob1, cfdob2		

	Vertical relationships		
Label	**Domain**	**Managment**	**Operational**
cdob3	crddob3, cfdob1, cfdob2		
cdob4	chrdob2, csdob2, csdob4		
cdob5	chrdob2, csdob2, csdob4		
cdob6	crddob3, cfdob2		
crdd	College	Research department	Director
crddob1	crdspob3, crdrmob2		
crddob2	crdspob1, crdspob2		
crddob3	crdrmob3		
crddob4	crdspob1, crdrmob1		
chrd	College	Human resources	Director
chrdob1	chrsob1, chrsob2, chrsob3		
chrdob2	chrsob3		
csd	College	School	Director
csdob1	csasob1, csadob1, cscdob1		
csdob2	csasob3, cscdob2, cscdob3		
csdob3	csasob2, csasob3, cscdob5		
csdob4	csasob2, csasob4, csadob3, cscsedob1		
cscd	College	School/COD	COD's office
cscdob1	cscdrob1, cscdceob2		
cscdob2	cscdrob2, cscdtob1		
cscdob3			
cscdob4	cscdrob2		
cscdob5	cscdtob2		
cscsed	College	School/CSE	Director
cscsedob1	cscseelob2, cscseelob3		
cscsedob2	cscseccob1, cscseccob2, cscseccob3, cscsessob2		
cfd	College	Finances	Director
cfdob1	cfrob1, cfrfob2		
cfdob2	cfrob1, cfrfob1, cfrfob2		

Table B.2: Vertical relationships among business objectives

B.0.3 Horizontal relationships

	Horizontal relationships		
Label	**Domain**	**Managment**	**Operational**
cd	College	Deanery	Deanery
(cdob6,cdob3)			
crdd	College	Research department	Director
(crddob3,crddob4)			
crdsp	College	Research department	Support programme
(crdspob3, crdspob1)			
crdrm	College	Research department	Research management
chrd	College	Human resources	Director
chrs	College	Human resource	Staffing
csd	College	School	Director
(csdob2, csdob1), (csdob4, csdob1)			
csas	College	School	Academic staff
csad	College	School	Admin staff
(csadob2, csadob1),(csadob3, csadob1)			
cscd	College	School/COD	COD's office
(cscdob2, cscdob1),(cscdob4, cscdob1), (cscdob5, cscdob2)			
cscdr	College	School/COD	Research
cscdt	College	School/COD	Tuition
cscdce	College	School/COD	Community Engagement
cscsed	College	School/CSE	Director
(cscsedob1, cscsedob2)			
cscseel	College	School/CSE	Enterprise liaison
(cscseelob1, cscseelob3)			
cscsecc	College	School/CSE	Certificate course
(cscseccob1, cscseccob3), (cscseccob2, cscseccob3)			
cscsess	College	School/CSE	Support staff
(cscsessob1, cscsessob2)			

	Horizontal relationships		
Label	**Domain**	**Managment**	**Operational**
cfd	College	Finances	Director
(cfdob2, cfdob1)			
cfr	College	Finances	Renumeration
cfrf	College	Finances	Research funding

Table B.3: Horizontal relationships among business objectives

B.0.4 The Prolog code for the case study

```
%list of identifiers
givenset([cd,crdd,crdsp,crdrm,chrd,chrs,csd,csas,csad,cscd,cscdr,cscdt,cscdce,
          cscsed,cscseel,cscsecc,cscsess,cfd,cfr,cfrf],identifier).

%list of objectives
givenset([cdob1,cdob2,cdob3,cdob4,cdob5,cdob6,
          crddob1,crddob2,crddob3,crddob4,
          crdspob1,crdspob2,crdspob3,
          crdrmob1,crdrmob2,crdrmob3,
          chrdob1,chrdob2,
          chrsob1,chrsob2,chrsob3,
          csdob1,csdob2,csdob3,csdob4,
          csasob1,csasob2,csasob3,csasob4,
          csadob1,csadob2,csadob3,
          cscdob1,cscdob2,cscdob3,cscdob4,cscdob5,
          cscdrob1,cscdrob2,
          cscdtob1,cscdtob2,
          cscdceob1,cscdceob2,
          cscsedob1,cscsedob2,
          cscseelob1,cscseelob2,cscseelob3,
          cscseccob1,cscseccob2,cscseccob3,
          cscsessob1,cscsessob2,
          cfdob1,cfdob2,
          cfrob1,
```

```
          cfrfob1,cfrfob2],objective).

% List of edges

edge(cd/crdd).
edge(cd/chrs).
edge(cd/csd).
edge(cd/cfd).
edge(crdd/crdsp).
edge(crdd/crdrm).
edge(chrd/chrs).
edge(csd/csas).
edge(csd/csad).
edge(csd/cscd).
edge(csd/cscsed).
edge(cscd/cscdr).
edge(cscd/cscdt).
edge(cscd/cscdce).
edge(cscsed/cscseel).
edge(cscsed/cscsecc).
edge(cscsed/cscsess).
edge(cfd/cfr).
edge(cfd/cfrf).

%Mapping nodes to objectives

%Deanery of the college
%----------------------
nodeobj(cd/cdob1).
nodeobj(cd/cdob2).
nodeobj(cd/cdob3).
nodeobj(cd/cdob4).
nodeobj(cd/cdob5).
nodeobj(cd/cdob6).
%Director of the college research department
%------------------------------------------
nodeobj(crdd/crddob1).
```

```
nodeobj(crdd/crddob2).
nodeobj(crdd/crddob3).
nodeobj(crdd/crddob4).

% Support programme
nodeobj(crdsp/crdspob1).
nodeobj(crdsp/crdspob2).
nodeobj(crdsp/crdspob3).

% Research management
nodeobj(crdrm/crdrmob1).
nodeobj(crdrm/crdrmob2).
nodeobj(crdrm/crdrmob3).

% Director Human resources
%-------------------------
nodeobj(chrd/chrdob1).
nodeobj(chrd/chrdob2).

%Human resources staffing
nodeobj(chrs/chrsob1).
nodeobj(chrs/chrsob2).
nodeobj(chrs/chrsob3).

% School director
%----------------
nodeobj(csd/csdob1).
nodeobj(csd/csdob2).
nodeobj(csd/csdob3).
nodeobj(csd/csdob4).

% Academic staff in the school
nodeobj(csas/csasob1).
nodeobj(csas/csasob2).
nodeobj(csas/csasob3).
nodeobj(csas/csasob4).
```

```
% Admin staff in the school
nodeobj(csad/csadob1).
nodeobj(csad/csadob2).
nodeobj(csad/csadob3).

%Chair of Department
nodeobj(cscd/cscdob1).
nodeobj(cscd/cscdob2).
nodeobj(cscd/cscdob3).
nodeobj(cscd/cscdob4).
nodeobj(cscd/cscdob5).

%Research
nodeobj(cscdr/cscdrob1).
nodeobj(cscdr/cscdrob2).

%Tuition
nodeobj(cscdt/cscdtob1).
nodeobj(cscdt/cscdtob2).

%Community engagement
nodeobj(cscdce/cscdceob1).
nodeobj(cscdce/cscdceob2).

%Director Centre for Software Engineering
%---------------------------------------
nodeobj(cscsed/cscsedob1).
nodeobj(cscsed/cscsedob2).

%Enterprise liaison
nodeobj(cscseel/cscseelob1).
nodeobj(cscseel/cscseelob2).
nodeobj(cscseel/cscseelob3).

%Certificate course
nodeobj(cscsecc/cscseccob1).
nodeobj(cscsecc/cscseccob2).
```

```
nodeobj(cscsecc/cscseccob3).

%Support staff
nodeobj(cscsess/cscsessob1).
nodeobj(cscsess/cscsessob2).

%Director department of finances
%------------------------------
nodeobj(cfd/cfdob1).
nodeobj(cfd/cfdob2).

%Remuneration finances department
nodeobj(cfr/cfrob1).

%Research funding finances department
nodeobj(cfrf/cfrfob1).
nodeobj(cfrf/cfrfob2).


%Vertical relationships:
%Business objectives refinement

%Deanery
%------------------
vrel(cdob1/crddob1).
vrel(cdob1/crddob4).
vrel(cdob1/chrdob1).
vrel(cdob1/chrdob2).
vrel(cdob1/csdob2).
vrel(cdob2/crddob2).
vrel(cdob2/csdob1).
vrel(cdob2/csdob3).
vrel(cdob2/csdob4).
vrel(cdob2/cfdob1).
vrel(cdob2/cfdob2).
vrel(cdob3/crddob3).
vrel(cdob3/cfdob1).
```

```
vrel(cdob3/cfdob2).
vrel(cdob4/chrdob2).
vrel(cdob4/csdob2).
vrel(cdob4/csdob4).
vrel(cdob5/chrdob2).
vrel(cdob5/csdob2).
vrel(cdob5/csdob4).
vrel(cdob6/crddob3).
vrel(cdob6/cfdob2).

%Director Research department
%----------------------------
vrel(crddob1/crdspob3).
vrel(crddob1/crdrmob2).
vrel(crddob2/crdspob1).
vrel(crddob2/crdspob2).
vrel(crddob3/crdrmob3).
vrel(crddob4/crdspob1).
vrel(crddob4/crdrmob1).

%Director human resources
%------------------------
vrel(chrdob1/chrsob1).
vrel(chrdob1/chrsob2).
vrel(chrdob1/chrsob3).
vrel(chrdob2/chrsob3).

% School director
%----------------
vrel(csdob1/csasob1).
vrel(csdob1/csadob1).
vrel(csdob1/cscdob1).
vrel(csdob2/csasob3).
vrel(csdob2/cscdob2).
vrel(csdob2/cscdob3).
vrel(csdob3/csasob2).
vrel(csdob3/csasob3).
```

```
vrel(csdob3/cscdob5).
vrel(csdob4/csasob2).
vrel(csdob4/csasob4).
vrel(csdob4/csadob3).
vrel(csdob4/cscsedob1).

% COD's office
%-------------
vrel(cscdob1/cscdrob1).
vrel(cscdob1/cscdceob2).
vrel(cscdob2/cscdrob2).
vrel(cscdob2/cscdtob1).
vrel(cscdob4/cscdrob2).
vrel(cscdob5/cscdtob2).

% Center for software engineering
%--------------------------------
vrel(cscsedob1/cscseelob2).
vrel(cscsedob1/cscseelob3).
vrel(cscsedob2/cscseccob1).
vrel(cscsedob2/cscseccob2).
vrel(cscsedob2/cscseccob3).
vrel(cscsedob2/cscsessob2).
% Direct department of finances
%------------------------------
vrel(cfdob1/cfrob1).
vrel(cfdob1/cfrfob2).
vrel(cfdob2/cfrob1).
vrel(cfdob2/cfrfob1).
vrel(cfdob2/cfrfob2).

% hrel: HORIZONTAL RELATIONSHIPS
% Supporting objectives
hrel(cdob6/cdob3).
hrel(crddob3/crddob4).
hrel(crdspob3/crdspob1).
hrel(csdob2/csdob1).
```

```
hrel(csdob4/csdob1).
hrel(csadob2/csadob1).
hrel(csadob3/csadob1).
hrel(cscdob2/cscdob1).
hrel(cscdob4/cscdob1).
hrel(cscdob5/cscdob2).
hrel(cscsedob1/cscsedob2).
hrel(cscseelob1/cscseelob3).
hrel(cscseccob1/cscseccob3).
hrel(cscseccob2/cscseccob3).
hrel(cscsessob1/cscsessob2).
hrel(cfdob2/cfdob1).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set operations needed for the animation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
element(X,[X|L]).
element(X,[Y|L]):- element(X,L).
%subset
subset([],_).
subset([A|X],Y):-
element(A,Y),subset(X,Y).

%union
union([ ], X, X ).
union(X,[],X):- !.
  union( [ X | R ], Y, Z ) :-
     element( X, Y ),
     !,
     union( R, Y, Z ).
  union( [ X | R ], Y, [ X | Z ] ) :- union( R, Y, Z ).

%Delete all the ocurrences of an element from a list
delelement(X, [], []) :- !.
delelement(X, [X|Xs], Y) :- !, delelement(X, Xs, Y).
delelement(X, [T|Xs], Y) :- !, delelement(X, Xs, Y2), union([T], Y2, Y).

% Assumes lists contain no duplicate elements.
```

```
  intersection( [ ], X, [ ] ):- !.
  intersection( X, [], []):- !.
  intersection( [ X | R ], Y, [ X | Z ] ) :-
     element( X, Y ),
     !,
     intersection( R, Y, Z ).
  intersection( [ X | R ], Y, Z ) :- intersection( R, Y, Z ).

%partition
partition([],Y,Y):- !.
partition(X,[],X):- !.

partition([X|Tail1],Y,[X|Tail2]):-
not(element(X,Y)),
intersection(Tail1,Y,[]),
partition(Tail1,Y,Tail2).

% Checks if R is a relation from A to B
rel([],_,_):- !.

rel([X/Y|R],A,B):-
element(X,A),
element(Y,B),
rel(R,A,B).

%checks if F is a function
func([],_,_):-!.

func([X/Y|F],A,B):-
element(X,A),
element(Y,B),
not(element(X/_,F)),
func(F,A,B).

% Domain and range of a relation
dom([],[]).
dom(R,D):-
```

```
setof(X,element(X/_,R),D).

ran([],[]).
ran(R,Ran):-
setof(X,element(_/X,R),Ran).

% unifying all the objectives of a group of operational elements
gadd(Ob,L,[Ob|L]).

gunion([[_,Ob,_,_,_]|Tail],D):-
gunion(Tail,D1),
gadd(Ob,Tail,D).

%Remove duplicates on a list
rem_dups([], []).
rem_dups([First | Rest], NewRest) :-
element(First, Rest),
rem_dups(Rest, NewRest).
rem_dups([First | Rest], [First | NewRest]) :-
not(element(First, Rest)),
rem_dups(Rest, NewRest).

validedges([]).
validedges([X/Y|Tail]):-
edge(X/Y),!,
validedges(Tail).

validnodeobj([]).
validnodeobj([X|Tail]):-
nodeobj(X),!,
validnodeobj(Tail).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma stateOperationalElt
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_,id).
varname(_,name).
```

```
schema_type([Id,Name], stateoperationalelt):-
%given sets
givenset(I,identifier),
%Variables and partial functions
varname(No,id),
varname(Nom,name),
element(No,I).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma stateOrganogram
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_,edges).
varname(_,nodeob).

schema_type([Edges,Obj],stateorganogram):-
%given sets
givenset(Ns, identifier),
givenset(Gs,objective),
%variables names
varname(Edges, edges),
varname(Obj, nodeob),
% variables' definition
validedges(Edges),
validnodeobj(Obj),
%Predicate
findall(X,element(X/_,Edges),DE),
rem_dups(DE,DomE),
findall(D,element(_/D,Edges),RE),
rem_dups(RE,RanE),
union(DomE,RanE,UNodes),
findall(X,element(X/_,Obj),DomOb),
rem_dups(DomOb,Dob),
subset(Dob,UNodes).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Horizontal search
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicate to select a tuple for output
hsel(Ov,Oi,V, Lhrel):-
nodeobj(V/Ov),
hrel(Oi/Ov),
not(element(Oi/Ov, Lhrel)),
nodeobj(W/Oi).
%select node
selnode(N, O, Ni, List):-
not(N==Ni),
element(O, List),
nodeobj(N/O).

varname(_, in(id)).
varname(_,inout(curobj)).
varname(_,inout(newv)).
varname(_,inout(newobj)).
varname(_,inout(newhrel)).

schema_op([Nodeid, CurObj, CurHr, NewV, NewObj, NewHrel],horizsearch):-
%Given sets
givenset(Ns, identifier),
givenset(Gs, objective),
%variables' names
varname(Nodeid, in(id)),
varname(CurObj, inout(curobj)),
varname(NewV, inout(newv)),
varname(NewObj, inout(newobj)),
varname(NewHrel, inout(newhrel)),
write('*****Horizontal search: Inputs *********'),nl,
write('Nodeid: '), write(Nodeid),nl,
write('CurObj: '), write(CurObj),nl,
%find the horizontal relationships for the input node
findall(Oi/Ov,hsel(Ov,Oi,Nodeid, CurHr),Od),
rem_dups(Od, NewHrel),
%Add input node objectives participating to horizontal relationships
findall(X,element(X/_,Od),Obs),
```

```
rem_dups(Obs, NewObj),
%union(Obs,CurObj,CurObjo),
%find new nodes
findall(N, selnode(N, On, Nodeid, NewObj), Ln),
rem_dups(Ln, NewV0),
delelement(Nodeid, NewV0, NewV),
%display outputs
write('*******Horizontal search: Ouputs *********'),nl,
write('NewObj! : '), write(NewObj),nl,
write('NewHrel!: '), write(NewHrel),nl,
write('NewV!   : '), write(NewV),nl.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vertical search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Predicate to select a tuple for output
% **** predecessors *****
vpsel(Owi,Ovi,V,Wi, Lwi, Lobj, Lvrel):-
element(Wi, Lwi),
element(Ovi, Lobj),
nodeobj(Wi/Owi),
nodeobj(V/Ovi),
not(element(Owi/Ovi, Lvrel)),
vrel(Owi/Ovi).

% **** successors *****
vssel(Ovi, Osi, V, Si, Lsi, Lobj, Lsrel):-
element(Si, Lsi),
element(Ovi, Lobj),
nodeobj(Si/Osi),
nodeobj(V/Ovi),
not(element(Ovi/Osi, Lsrel)),
vrel(Ovi/Osi).

%List of valid predecessors
```

```
validpred(W, V):-
edge(W/V),
nodeobj(V/Ov),
nodeobj(W/Ow),
vrel(Ow/Ov).

% Validate a successor node
validsucc(V, S, Lobj):-
edge(V/S),
nodeobj(V/Ov),
nodeobj(S/Os),
element(Ov, Lobj),
vrel(Ov/Os).

varname(_,in(id)).
varname(_,inout(curobj)).
varname(_,inout(newv)).
varname(_,inout(newobj)).
varname(_,inout(newhrel)).

schema_op([Nodeid, CurObj, CurVr, NewV, NewObj, NewVrel],verticalsearch):-

%Given sets
givenset(Ns, identifier),
givenset(Gs, objective),
%variables' names
varname(Nodeid, in(id)),
varname(CurObj, inout(curobj)),
varname(NewV, inout(newv)),
varname(NewObj, inout(newobj)),
varname(NewVrel, inout(newhrel)),
%display inputs
write('*****************************************'),nl,
write('      Vertical search: Inputs    '), nl,
write('Nodeid  : '), write(Nodeid),nl,
write('CurObj  : '), write(CurObj),nl,
%find all predecessor w of the input node v
```

```
findall(W, validpred(W, Nodeid),Lw0),
rem_dups(Lw0, Lw),
%find all successors S of the input node V
findall(S, validsucc(Nodeid, S, CurObj),Ls0),
rem_dups(Ls0, Ls),
%find the vertical relationships for the input node
findall(Ow/Ov, vpsel(Ow,Ov,Nodeid,W,Lw,CurObj, CurVr),Od1),
findall(Ov/Os, vssel(Ov,Os,Nodeid,S,Ls, CurObj, CurVr),Od2),
union(Od1, Od2, Od),
rem_dups(Od, NewVrel),
%Add input node objectives participating to horizontal relationships
findall(X,element(X/_, Od1),Obs0),
findall(X,element(_/X, Od2), Obs1),
union(Obs0, Obs1, Obs),
rem_dups(Obs, NewObj),
%Add all predecessor and successor nodes to the list of current vertices
union(Lw, Ls, L),
rem_dups(L, NewV1),
delelement(Nodeid, NewV1, NewV),
%display outputs
write('************Vertical search: Outputs ***************'), nl,
write('NewObj! : '), write(NewObj),nl,
write('NewVrel!: '), write(NewVrel),nl,
write('NewV!   : '), write(NewV).

%*****************************
% The main processing algorithm
%*****************************
schema_op([[],_,_,_,_], main):-!.
schema_op([Black,_,_,_,Black],main):- !.
schema_op([[X|Tail],X1,X2,X3,Black], main):-
element(X,Black),
nonvar(Tail),
nonvar(X1),
%nonvar(X2),
schema_op([Tail,X1,X2,X3,Black],main).
```

```
schema_op([[V|Tail], CurObj, CurHrel, CurVrel, Black], main):-
%Perform the horizontal search
schema_op([V, CurObj, CurHrel, NewNodes1, NewObj1, NewHrel], horizsearch),
% Perform the vertical search
schema_op([V, CurObj, CurVrel, NewNodes2, NewObj2, NewVrel], verticalsearch),
%process the horizontal/vertical search results
union(NewHrel, CurHrel, NewHrel0),
rem_dups(NewHrel0, CurHrel1),
union(NewVrel, CurVrel, CurVrel0),
rem_dups(CurVrel0, CurVrel1),
union(NewNodes1, NewNodes2, NewNodes),
union(NewNodes, Tail, NewNodes3),
delelement(V,NewNodes3,NewN),
rem_dups(NewN, Tail1),
union(NewObj1, NewObj2, NewObj),
union(NewObj, CurObj, CurObj0),
rem_dups(CurObj0, CurObj1),
gadd(V,Black, Black1),
% **********************
% Main Program Outputs
%**********************
nl,
write('************** main program ***********'),nl,
write('CurV!   : '), write(Tail1), nl,
write('CurObj! : '), write(CurObj1),nl,
write('CurHrel!: '), write(CurHrel1), nl,
write('CurVrel!: '), write(CurVrel1),nl,
write('black!  : '), write(Black1),nl,

schema_op([Tail1, CurObj1, CurHrel1, CurVrel1, Black1], main).
```

B.0.5 Execution

```
Amzi! Prolog (IDE Only)
Licensed to
Free Version
```

```
Interpreting project: MyProject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: 'operationaleltch5.pro'
Type 'quit.' to end and [Ctrl]-C to stop user input.

?- schema_op([[crdsp], [crdspob1,crdspob2], [], [], []],main).

************** main program ***********
CurV!   : [crdd]

CurObj! : [cdob3, cfrob1, cfrfob1, cfrfob2, cscseelob1, cscseelob2, cscseelob3,
cscdob4, cscdrob2, cscdrob1, cscdtob1, cscdtob2, cscdceob2, csadob2, cdob4,
cdob5, csasob1, csasob3, csasob2, csasob4, csadob1, csadob3, cscdob2, cscdob3,
cscdob1, cscdob5, cscsedob1, cdob6, crddob1, csdob2, csdob1, csdob3, csdob4,
cfdob1, cfdob2, crddob3, cdob1, cdob2, crdrmob1, crdspob3, crddob4, crddob2,
crdspob1, crdspob2]

CurHrel!: [cfdob2/cfdob1, cscseelob1/cscseelob3, cscsedob1/cscsedob2,
cscdob2/cscdob1, cscdob4/cscdob1, cscdob5/cscdob2, csadob2/csadob1,
csadob3/csadob1, csdob2/csdob1, csdob4/csdob1, cdob6/cdob3, crddob3/crddob4,
crdspob3/crdspob1]

CurVrel!: [cdob3/cfdob1, cdob3/cfdob2, cdob6/cfdob2, cfdob1/cfrob1, cfdob2/cfrob1,
cfdob1/cfrfob2, cfdob2/cfrfob1, cfdob2/cfrfob2, cscsedob1/cscseelob2,
cscsedob1/cscseelob3, cscdob4/cscdrob2, cscdob2/cscdrob2, cscdob1/cscdrob1,
cscdob2/cscdtob1, cscdob5/cscdtob2, cscdob1/cscdceob2, cdob4/csdob2, cdob5/csdob2,
cdob4/csdob4, cdob5/csdob4, csdob2/csasob3, csdob1/csasob1, csdob3/csasob2,
csdob3/csasob3, csdob4/csasob2, csdob4/csasob4, csdob1/csadob1, csdob4/csadob3,
csdob2/cscdob2, csdob2/cscdob3, csdob1/cscdob1, csdob3/cscdob5, csdob4/cscsedob1,
cdob1/crddob1, cdob1/csdob2, cdob2/csdob1, cdob2/csdob3, cdob2/csdob4, cdob2/cfdob1,
cdob2/cfdob2, cdob1/crddob4, cdob2/crddob2, crddob4/crdrmob1, crddob2/crdspob1,
crddob4/crdspob1, crddob2/crdspob2]

black!  : [crdrm, cfrf, cfr, cfd, cscseel, cscsed, cscdce, cscdt, cscdr,
             cscd, csad, csas, csd, cd, crdd, crdsp]

yes
```

```
?-
```

B.0.6 Problem analysis

pDomain Component		**p1:** limited Emails size	**p2:** Inconsistent Appl. forms	**p3:** statistics Hard to draw	**p4:** stressfull App. process	**p5:** stressfull Eval Process
crdsp	ob1	data access	errors	delay		time
	ob2	accessibility		delay	discourage	
	ob3	delay	errors	delay		time
crdrm	ob1					
crdd	ob1			delay		
	ob2					
	ob3			no info		
	ob4				discourage	delay
cfd	ob1		errors	delay		
	ob2					
csas	ob1				time	time
	ob2				delay	delay
	ob3					
	ob4					
csd	ob1	neg. impact			discourage	discourage
	ob2					
	ob3					delay
	ob4	obstacle	errors	neg. impact		
csad	ob1					
	ob2					
	ob3	more work				more work
cscd	ob1		errors			
	ob2					
	ob3					
	ob4		errors		discourage	discourage
	ob5					
cscdr	ob1		error	delay		
	ob2					
cscdce	ob2					
cscdt	ob1					
	ob2					
cscsed	ob1		errors	delay		
cscseel	ob1					

pDomain Component		**p1:** limited Emails size	**p2:** Inconsistent Appl. forms	**p3:** statistics Hard to draw	**p4:** stressfull App. process	**p5:** stressfull Eval Process
	ob2					
	ob3	comm. pb	errors	delay	discourage	discourage
cfr	ob1					
cd	ob1		errors	delay	discourage	discourage
	ob2				discourage	discourage
	ob3					
	ob4					
	ob5					
	ob6					
cfrf	ob1		errors	delay		
	ob2		errors			

Table B.4: Analysing the initial problems

B.0.7 Reduced *pNode*

Let pNodes = {
crdsp{*ob*1, *ob*2, *ob*3}, *crdd*{*ob*1, *ob*3}, *cfd*{*ob*1},
csas{*ob*1, *ob*2}, *csd*{*ob*1, *ob*3, *ob*4}, *csad*{*ob*3},
cscd{*ob*1, *ob*4}, *cscdr*{*ob*1}, *cscsed*{*ob*1},
cscseel{*ob*3}, *cfr*{*ob*1}, *cd*{*ob*1}, *cfrf*{*ob*1, *ob*2}
}

Appendix C

The Object-Z specification of the case study

ClsMetadata

$\lceil(name, value)$

$[Name]$

$name : Name$

$value : String$

ClsGRLModelElement

$\lceil(ref, name, mdata)$

$[Identifier]$

$ref : Identifier$

$name : String$

$mdata : \mathbb{F}\ ClsMetaData$

Init

$ref = \perp$

C.0.1 Actors' classes: *ClsApplicant*, *ClsMotivator*, *ClsAdministrator*, *ClsServer*

Components inherited from the superclass: *ClsActor*

$ref : Identifier$

$name : String$

$mdata : \mathbb{F}\ ClsMetaData$

$importance : ImportanceType$

$impquantitative : \mathbb{Z}$

Actor's class schema: *ClsApplicant*

ClsActorApplicant

$\lceil (name, softgoals, goals, beliefs, nfdependers)$

ClsActor

$applicantid : Identifier$
$softgoals : \mathbb{P}\ ClsSoftGoal$
$goals : \mathbb{P}\ ClsGoals$
$beliefs : \mathbb{P}\ ClsBelief$
$nfdependers : \mathbb{P}\ ClsSoftgoal$

$\forall\ nfelt \in nfdependers \bullet nfelt.actor.ref \neq ref$

INIT

$softgoals = \perp \land\ nfdependers = \perp \land\ goals = \perp \land\ beliefs = \perp$
$ClsActor.Init$

NewSoftGoal

$\Delta softgoals$
$sgoal? : ClsSoftGoal$

$sgoal?.Init\ \land\ sgoal?.actor.ref = ref$
$softgoals' = softgoals \oplus \{sgoal?\}$

NewGoal

$\Delta goals$
$goal? : ClsGoal$

$goal?.Init\ \land\ goal?.actor.ref = ref$
$goals' = goals \oplus \{goal?\}$

NewBelief

$\Delta beliefs$
$belief? : ClsBelief$

$belief?.actor.ref = ref$
$beliefs' = beliefs \oplus \{belief?\}$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

Actor's class schema: *ClsMotivator*

ClsActorMotivator

$\lceil(name, softgoals, goals, beliefs, nfdependers)$
ClsActor

$motivatorid : Identifier$
$softgoals : \mathbb{P}\ ClsSoftGoal$
$goals : \mathbb{P}\ ClsGoals$
$beliefs : \mathbb{P}\ ClsBelief$
$nfdependers : \mathbb{P}\ ClsSoftgoal$

$\forall\ nfelt \in nfdependers \bullet nfelt.actor.ref \neq ref$

INIT

$softgoals = \bot \land\ nfdependers = \bot \land\ goals = \bot \land\ beliefs = \bot$
$ClsActor.Init$

NewSoftGoal

$\Delta softgoals$
$sgoal? : ClsSoftGoal$

$sgoal?.Init\ \land\ sgoal?.actor.ref = ref$
$softgoals' = softgoals \oplus \{sgoal?\}$

NewGoal

$\Delta goals$
$goal? : ClsGoal$

$goal?.Init\ \land\ goal?.actor.ref = ref$
$goals' = goals \oplus \{goal?\}$

NewBelief

$\Delta beliefs$
$belief? : ClsBelief$

$belief?.actor.ref = ref$
$beliefs' = beliefs \oplus \{belief?\}$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

Actor's class schema: *ClsAdministrator*

ClsActorAdministrator

$\lceil(name, softgoals, goals, beliefs, nfdependers)$
ClsActor

$adminid : Identifier$
$softgoals : \mathbb{P}\ ClsSoftGoal$
$goals : \mathbb{P}\ ClsGoals$
$beliefs : \mathbb{P}\ ClsBelief$
$nfdependers : \mathbb{P}\ ClsSoftgoal$

$\forall\ nfelt \in nfdependers \bullet nfelt.actor.ref \neq ref$

INIT

$softgoals = \bot \land\ nfdependers = \bot \land\ goals = \bot \land\ beliefs = \bot$
$ClsActor.Init$

NewSoftGoal

$\Delta softgoals$
$sgoal? : ClsSoftGoal$

$sgoal?.Init\ \land\ sgoal?.actor.ref = ref$
$softgoals' = softgoals \oplus \{sgoal?\}$

NewGoal

$\Delta goals$
$goal? : ClsGoal$

$goal?.Init\ \land\ goal?.actor.ref = ref$
$goals' = goals \oplus \{goal?\}$

NewBelief

$\Delta beliefs$
$belief? : ClsBelief$

$belief?.actor.ref = ref$
$beliefs' = beliefs \oplus \{belief?\}$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

Actor's class schema: *ClsServer*

ClsActorServer

$\lceil(name, softgoals, goals, beliefs, nfdependers)$

ClsActor

$serverid : Identifier$
$softgoals : \mathbb{P}\ ClsSoftGoal$
$goals : \mathbb{P}\ ClsGoals$
$beliefs : \mathbb{P}\ ClsBelief$
$nfdependers : \mathbb{P}\ ClsSoftgoal$

$\forall\, nfelt \in nfdependers \bullet nfelt.actor.ref \neq ref$

INIT

$softgoals = \bot \land\ nfdependers = \bot \land\ goals = \bot \land\ beliefs = \bot$
$ClsActor.Init$

NewSoftGoal

$\Delta softgoals$
$sgoal? : ClsSoftGoal$

$sgoal?.Init\ \land\ sgoal?.actor.ref = ref$
$softgoals' = softgoals \oplus \{sgoal?\}$

NewGoal

$\Delta goals$
$goal? : ClsGoal$

$goal?.Init\ \land\ goal?.actor.ref = ref$
$goals' = goals \oplus \{goal?\}$

NewBelief

$\Delta beliefs$
$belief? : ClsBelief$

$belief?.actor.ref = ref$
$beliefs' = beliefs \oplus \{belief?\}$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

ClsLinkableElement

$\lceil(ref, name, mdata, importance, impQualitative)$

$ImportanceType == High \mid Medium \mid Low \mid None$

ClsGRLModelElement

$importance : ImportanceType$

$impQualitative : \mathbb{Z}$

ClsActor

$\lceil(ref, name, mdata, importance, impQualitative)$

ClsLinkableElement

ClsGRLContainableElt

$\lceil(ref, name, mdata, importance, impQualitative, actor)$

ClsLinkableElement

$actor : ClsActor$

INIT

$actor.ref = \bot$

ClsIntentionalElement

$\lceil(ref, name, mdata, importance, impQualitative, actor)$

ClsContainableElement

$(actor \neq \bot \land \forall\, elt : \downarrow ClsIntentionalElement \mid elt.actor.ref = actor.ref$

$\lor$

$actor = \bot \land \forall\, elt : \downarrow ClsIntentionalElement \mid elt.actor = \bot) \bullet$

$elt.name \neq name$

C.0.2 The OZ specification of the resources

Two resources are to be specified: *ClsIntranet* and *ClsInternet*.

ClsResource

$\lceil(ref, name, mdata, importance, impQualitative, actor, available)$
ClsIntentionalElement

$available : Integer$

$available \geq 0$

INIT

$available = 0$

Creating the class: *ClsIntranet*

ClsIntranet

$\lceil(messageout, messagein, send, receive)$
ClsResource
$[Message]$

$nfdependers : \mathbb{P}\ ClsSoftGoal$
$messageout : \mathbb{P}\ Message$
$messagein : \mathbb{P}\ Message$

$messageout \cap messagein = \varnothing$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

send

$\Delta messageout$
$msg? : Message$

$messageout' = messageout \cup \{msg?\}$

receive

$\Delta messagein$
$msg? : Message$

$messagein' = messagein \cup \{msg?\}$

Creating the class: *ClsInternet*

ClsInternet

$\lceil(provider)$
ClsResource
$[Webpage]$

$nfdependers : \mathbb{P}\ ClsSoftGoal$
$openedpages : \mathbb{P}\ Webpage$
$savedpages : \mathbb{P}\ Webpage$
$connected : \mathbb{B}$

$openedpages \subseteq savedpages$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

saveform

$\Delta savedpages$
$myform? : Webpage$

$savedpages' = savedpages \oplus \{myform?\}$

loadpage

$\Delta openedpages$
$myform? : Webpage$

$openedpage' = openedpage \cup \{myform?\}$

C.0.3 The OZ specification of Tasks and ressources linked to applicant

The two classes to specify are: *ClsAccessOwnApp* and *ClsSubmitAppOnline*.

The class *ClsAccessOwnApp*

The list represents in fact a view, accessible by the applicant, of the applications in a database.

ClsAccessOwnApp

$\lceil(apps, SelectApp)$
ClsIntentionalElement
$[Webpage]$

$inet : ClsInternet$
$nfdependers : \mathbb{P}\ ClsSoftGoal$

SelectApp

$\Delta inet.openedpages$

$inet.savedpages \neq \varnothing$
$\exists\, aform \in inet.savedpages \bullet inet.loadpage(aform)$

Addnfdepender

$\Delta nfdependers$
$sgoal? : ClsSoftGoal$

$nfdependers' = nfdependers \oplus \{sgoal?\}$

The variable *apps* contains a list of applications submitted by applicants for different research support programmes. The operation *SelectApp* accesses the list of submitted applications and allowing the applicant to select any of its' applications.

The class: *ClsSubmitAppOnline*

This class is very similar to the class *ClsAccessOwnApp*;

ClsSubmitAppOnline

$\lceil(SubmitApp)$
ClsIntentionalElement
$[Webpage]$

$inet : ClsInternet$

SubmitApp

$\Delta inet.savedpages$
$myform? : Webpage$

$inet.saveform(myform?)$

The operation *SubmitApp* sends the application *app*? back to the database. At this stage of the specification, the operation does not indicate if the older version of the application is replaced by the modified version or not.

C.0.4 The OZ specification of Tasks and ressources linked to Motivator

The classes to specify are: *ClsMeetingToMotivEval*, *ClsSubmitMotivEvalOnline* and *ClsReportOnSubmitedApp*.

The class: *ClsMeetingToMotivEval*

ClsMeetingToMotivEval

$\lceil(meetings, members, applications, organizeAmeeting, inviteAmember, allocateApplications)$

ClsIntentionalElement

$[Meeting, Person, Date, Time]$

$$meetings : Meeting \pfun Place \times Date \times Time$$
$$members : Meeting \pfun Person$$
$$applications : Meeting \pfun Application$$

INIT

$$meetings = \varnothing \ \land \ members = \varnothing \ \land \ applications = \varnothing$$

organizeAmeeting

$$\Delta meetings$$
$$meet? : Meeting$$
$$place? : Place$$
$$date? : Date$$
$$time? : Time$$
$$meetings' = meetings \cup \{meet? \mapsto (place?, date?, time?)\}$$

inviteAmember

$$\Delta members$$
$$memb? : Person$$
$$meet? : Meeting$$
$$members' = members \cup \{meet? \mapsto memb?\}$$

allocateApplications

$$\Delta applications$$
$$apps? : \mathbb{P}\, Application$$
$$meet? : Meeting$$
$$applications' = applications \cup \{meet? \mapsto apps?\}$$

The class: *ClsSubmitMotivEvalOnline*

ClsSubmitMotivEvalOnline

$\lceil(submitEvaluation, submitMotivation)$
ClsIntentionalElement
$[Webpage]$

$inet : ClsInternet$

$inet.connected$

INIT

$evaluations = \varnothing \ \land \ motivations = \varnothing$

submitEvaluation

$\Delta inet.savedpages$
$evalform? : Webpage$

$inet.save(evalform?)$

submitMotivation

$\Delta inet.savedpages$
$motivform? : Webpage$

$inet.save(motivform?)$

The class: *ClsReportOnSubmitedApp*

ClsReportOnSubmitedApp

$\lceil(allreports, newReport)$
ClsIntentionalElement
$[Report, Webpage]$

$report : \mathbb{P}\ Webpage \nrightarrow Report$

$applreports : \mathbb{P}\ Report$

newReport

$\Delta applreports$
$report? : Report$

$\mathrm{dom}(report \rhd \{report?\}) \neq \varnothing$
$applreports' = applreports \oplus \{report?\}$

C.0.5 The OZ specification of Tasks and ressources linked to Administration

The classes to specify are: *ClsGenStatsOnApp*, *ClsProvideInfoGenForms* and *ClsKeepPgDataInDb*.

The class: *ClsGenStatsOnApp*

ClsGenStatsOnApp

$\lceil(msgdata, stat, statistic)$
ClsIntentionalElement
$[Data, Statistic, Webpage]$

$msgdata : Message \nrightarrow Data$
$stat : \mathbb{P}\, Data \nrightarrow Statistic$

$tranet : ClsIntranet$

statistic

$data? : \mathbb{P}\, Data$
$stat! : Statistic$

$\mathrm{dom}(msgdata \rhd data?) \neq \varnothing$
$data? \mapsto stat! \in stat$

The class: *ClsProvideInfoGenForms*

ClsProvideInfoGenForms

$\lceil(msgsection, savesection)$
ClsIntentionalElement
$[Section]$

$msgsection : Section \nrightarrow Message$

$tranet : ClsIntranet$

savesection

$sec? : Section$

$sec? \mapsto msg \in msgsection$
$tranet.sent(msg)$

The class: *ClsKeepPgDataInDb*

ClsKeepPgDataInDb

$\lceil(progmsg, saveprogramme)$
ClsIntentionalElement
$[Programme]$

$progmsg : Programme \pfun Message$

$tranet : ClsIntranet$

---

savePrograme

$prog? : Programme$

$prog? \mapsto msg \in progmsg$
$tranet.sent(msg)$

C.0.6 The OZ specification of Tasks and ressources linked to Server

The classes to specify are: *ClsGenFormSections*, *ClsMaintSubmitedApp* and *ClsMaintDb4SuppPg*.

The class: *ClsGenFormSections*

ClsGenFormSections

$\lceil(sections, pages, onlineforms, generateAnonlineform)$
ClsIntentionalElement
$[Section, Webpage]$

$sections : \mathbb{P}\, Section$
$pages : \mathbb{P}\, Webpage$
$onlineforms : \mathbb{P}(\text{seq}_1\, Section \pfun Webpage)$

$\forall\, oform \in onlineforms \bullet$
$\quad f \in \text{dom}\, oform \Rightarrow f \in sections$
$\quad \text{ran}\, oform \subseteq forms$

INIT

$sections = \varnothing \ \land \ forms = \varnothing$

---

generateAnonlineform

$\Delta onlineforms$
$page? : Webpage$

$\forall\, form \in onlineforms \bullet page? \notin \text{ran}\, form$
$\exists\, Sec : \text{seq}\, Section \mid \{s \mid s \in Sec\} \subseteq sections \bullet$
$\quad onlineforms' = onlineforms \oplus \{Sec \mapsto page?\}$

The class: *ClsMaintSubmitedApp*

ClsMaintSubmitedApp

$\restriction(appdata, appforms, receiveOnlineData)$

ClsIntentionalElement

$[Data, Webpage]$

$formdata : Data \pfun Webpage$

$appdata : \mathbb{P}\, Data$

$appforms : \mathbb{P}\, Webpage$

$inet : ClsInternet$

$\forall\, data : Webpage \bullet data \in appdata \Rightarrow \exists_1\, page : Webpage \bullet$
$\quad data \mapsto page \in formdata$

receiveOnlineData

$\Delta appdata$

$sform? : Webpage$

$sform? \in inet.savedpages$

$\exists\, data : Data \mid data \mapsto sform \in formdata \bullet$
$\quad appdata' = appdata \oplus \{data\}$

The class: *ClsMaintDb4SuppPg*

ClsMaintDb4SuppPg

$\restriction(programmes, data, receiveNewprogdata)$

ClsIntentionalElement

$[Programme, Data]$

$progdata : Data \pfun Programme$

$msgdata : Message \pfun Data$

$programmes : \mathbb{P}\, Programme$

$data : \mathbb{P}\, Data$

$tranet : ClsIntranet$

$\mathrm{ran}(data \lhd progdata) \subseteq programmes$

receiveNewprogdata

$\Delta data$

$prog? : Programme$

$\exists (msg, data) \in tranet.messagein \times Data \mid msg \mapsto datain \in msgdata \bullet$
$\quad datain \mapsto prog?$
$\quad data' = data \oplus \{datain\}$
$\quad programmes' = programmes \oplus \{prog?\}$

C.0.7 Formalising the system class: ClsGrlCaseStudy

ClsGrlCaseStudy

$\lceil(freeelts, actorselts, alllinks)$

$actors : \mathbb{P}\, Identifier$
$freeelts : \mathbb{P}{\downarrow}ClsIntentionalElement$
$alllinks : \mathbb{P}{\downarrow}ClsElementLink$

$\forall\, elt \in free\_elts \bullet elt.actor = \bot$

INIT

$actors = \varnothing \ \land \ freeelts = \varnothing \ \land \ alllinks = \varnothing$

AddActor $[ref]$

$\Delta actors$

$actors' = actors \oplus \{ref\}$

NewFreeelt

$\Delta freeelts$
$elt? : {\downarrow}ClsIntentionalElement$

$elt?.actor.ref = \bot$
$freeelts' = freeelts \oplus \{elt?\}$

NewLink

$\Delta alllinks$
$link? : {\downarrow}ClsElementLink$

$alllinks' = alllinks \oplus \{link?\}$

BoundIntentElt

$elt? \in \{ClsTask, ClsResource\}$
$actor? : Actor$

$elt?.actor = actor?$

Appendix D

Prolog implementation of the Object-Z specification

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% facts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% list of identifiers
givenset([high, medium, low, none], importancetype).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set operations needed for the animation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
element(X,[X|L]).
element(X,[Y|L]):- element(X,L).

% subset
subset([],_).
subset([A|X],Y):-
element(A,Y),subset(X,Y).

% union
union([ ], X, X ).
union(X,[],X):- !.
  union( [ X | R ], Y, Z ) :-
     element( X, Y ),
     !,
```

```
     union( R, Y, Z ).
  union( [ X | R ], Y, [ X | Z ] ) :- union( R, Y, Z ).

% Assumes lists contain no duplicate elements.
  intersection( [ ], X, [ ] ):- !.
  intersection( X, [], []):- !.
  intersection( [ X | R ], Y, [ X | Z ] ) :-
     element( X, Y ),
     !,
     intersection( R, Y, Z ).
  intersection( [ X | R ], Y, Z ) :- intersection( R, Y, Z ).

%partition
partition([],[],[]):- !.

partition(X,Y,Z):-
intersection(X,Y,[]),
union(X,Y,Z).



% Checks if R is a relation from A to B
rel([],_,_):- !.

rel([X/Y|R],A,B):-
element(X,A),
element(Y,B),
rel(R,A,B).

%checks if F is a function
func([],_,_):-!.

func([X/Y|F],A,B):-
element(X,A),
element(Y,B),
not(element(X/_,F)),
func(F,A,B).
```

```
% Domain and range of a relation
dom([],[]).
dom(R,D):-
setof(X,element(X/_,R),D).

ran([],[]).
ran(R,Ran):-
setof(X,element(_/X,R),Ran).

%Remove duplicates on a list
rem_dups([], []).
rem_dups([First | Rest], NewRest) :-
element(First, Rest),
rem_dups(Rest, NewRest).
rem_dups([First | Rest], [First | NewRest]) :-
not(element(First, Rest)),
rem_dups(Rest, NewRest).

%Natural number
nat(0).
nat(X0) :-
   X0 @>0,
   X1 is X0-1,
   nat(X1).

%Save each successful state data into the file 'staterefs.pro'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
save_stateref(SRef):-
open('staterefs.pro',append,Stream,[type(wide_text)]),
pp(Stream,SRef),
close(Stream).
```

D.0.1 Implementing schema_type() using object'references

The class *ClsMetadata*

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma of ClsMetadata
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%

varname(_,refmetadata).
varname(_,name).
varname(_,value).

schema_type([Ref, [Name, Val]], metadata):-

%Variables and partial functions
varname(Ref, refmetadata),
varname(Name,name),
varname(Val, value),

string(Ref),
string(Name),

save_stateref(stateref([Ref,[Name, Val]])),

reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([Ref, [Name, Val]], metadata)

Amzi! Prolog (IDE Only)
Licensed to
Free Version

Interpreting project: phdproject
  Loading Extensions:  aosutils.lsx (always loaded in IDE)
  Consulting Source Files: 'commonclauses.pro', 'ozprologclauses.pro', 'staterefs.pro'
Type 'quit.' to end and [Ctrl]-C to stop user input.

?- schema_type([$md001$,[$Size$,23]], metadata).

yes
?- stateref([md001, LinkedData]).
```

```
LinkedData = [‘Size‘, 23] ;
no
?-
```

The class *ClsModelElement*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma of ClsModelElement
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%)
varname(_, refmodelelt).
varname(_, identifier).

schema_type([RefMe, [RefMd, Id]], modelelt):-
%Variables and partial functions
varname(RefMe, refmodelelt),
varname(Id, identifier),
varname(RefMd, refmetadata),

string(RefMe),
string(Id),
string(RefMd),

stateref([RefMd, X]),
%Process [RefMd, X]

save_stateref(stateref([RefMe, [RefMd, Id]])),

reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([RefMe, [RefMd, Id]], modelelt)

?- schema_type([$me004$, [$md003$, $id0001$]], modelelt).
yes
?- stateref([me004, LinkedData]).
LinkedData = [‘md003‘, ‘id0001‘] ;
no
?-
```

The class *ClsLinkableElement*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma of Linkable Element
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, reflinkableelt).
varname(_, importance).
varname(_, imprtancequalitative).

schema_type([RefLe,[RefMe,Imp, Iqtive]],linkableelt):-

%given set
givenset(Itype, importancetype),

%Variables and partial functions
varname(RefLe, reflinkableelt),
varname(RefMe, refmodelelt),
varname(Imp, importance),
varname(Iqtive, imprtancequalitative),

stateref([RefMe, X]),
%Process [RefMe, X]

element(Imp, Itype),
nat(Iqtive),

save_stateref(stateref([RefLe,[RefMe,Imp,Iqtive]])),
reconsult(staterefs).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([RefLe,[RefMe,Imp, Iqtive]],linkableelt)

?- schema_type([le003, [me004, low, 40]], linkableelt).
yes
?- stateref([le003, LinkedData]).
LinkedData = [me004, low, 40] ;
no
```

```
?-
```

The class *ClsActor*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma of the class ClsActor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_,refactor).

schema_type([RefAc, RefLe], actor):-

%Variables and partial functions
varname(RefAc,refactor),
varname(RefLe, reflinkableelt),

stateref([RefLe,X]),
%process X

save_stateref(stateref([RefAc,RefLe])),
reconsult(staterefs).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([RefAc, RefLe], actor)

?- schema_type([ac003, le001], actor).
yes
?- stateref([ac003, LinkedData]).
LinkedData = le001 ;
no
?-
```

The class *ClsContainableElement*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsGrlContainableElement
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refcontainableelt).
```

```
schema_type([RefCe,[RefLe,RefAc]], containableelt):-

%Variables and partial functions
varname(RefCe,refcontainableelt),
varname(RefLe, reflinkableelt),
varname(RefAc, refactor),

stateref([RefLe,Xle]),
%process Xle

stateref([RefAc,Xac]),
%process Xac

save_stateref(stateref([RefCe,[RefLe,RefAc]])),
reconsult(staterefs).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([RefCe,[RefLe,RefAc]], containableelt)

?- schema_type([ce002, [le003, ac003]], containableelt).
no
?- stateref([le003, LinkedData]).
no
?- stateref([le002, LinkedData]).
LinkedData = [me001, medium, 34] ;
no
?- schema_type([ce002, [le002, ac003]], containableelt).
yes
?- stateref([ce002, LinkedData]).
LinkedData = [le002, ac003] ;
no
?-
```

The class *ClsIntentionalElement*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%State schma for the class ClsIntentionalElement
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refintentionalelt).

schema_type([RefIe,RefCe], intentionalelt):-

%Variables and partial functions
varname(RefIe, refintentionalelt),
varname(RefCe, refcontainableelt),

stateref([RefCe,X]),
%process X

save_stateref(stateref([RefIe, RefCe])),
reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing the schema_type([RefIe,RefCe], intentionalelt)

?- schema_type([ie002, ce001], intentionalelt).
yes
?-
```

The class *ClsBelief*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsBelief
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refbelief).
varname(_, description).

schema_type([RefBl,[RefIe, Desc]], belief):-

%Variables and partial functions
varname(RefBl, refbelief),
varname(RefIe, refintentionalelt),
varname(Desc, description),
```

```
string(Desc),

stateref([RefIe, X]),
%Process RefIe and X here


save_stateref(stateref([RefBl,[RefIe, Desc]])),
reconsult(staterefs).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsResource
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refresource).
varname(_, available).

schema_type([RefRs, [RefIe, Avail]], resource):-

%Variables and partial functions
varname(RefRs, refresource),
varname(RefIe, refintentionalelt),
varname(Avail, available),

nat(Avail),
Avail @>= 0,
stateref([RefIe, X]),
write(X),nl,
%Process RefIe and X

save_stateref(stateref([RefRs, [RefIe, Avail]])),
reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsTask
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, reftask).
```

```
schema_type([RefTa,RefIe], task):-

%Variables and partial functions
varname(RefTa, reftask),
varname(RefIe, refintentionalelt),

stateref([RefIe,X]),
%process X

save_stateref(stateref([RefTa,RefIe])),
reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsGoal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refgoal).

schema_type([RefGo,RefIe], goal):-

%Variables and partial functions
varname(RefGo, refgoal),
varname(RefIe, refintentionalelt),

stateref([RefIe,X]),
%process X

save_stateref(stateref([RefGo,RefIe])),
reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsComplementaryAction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refcomplementaryaction).
varname(_,cnf-phase).
varname(_,cnf-action).
varname(_,cnf-domain).
varname(_,cnf-option).
```

```
schema_type([RefCa,[RefIe,Phase, Action, Domain, Option]], complementaryaction):-

%Variables and partial functions
varname(RefCa, refcomplementaryaction),
varname(RefIe, refintentionalelt),
varname(Phase, cnf-phase),
varname(Action, cnf-action),
varname(Domain, cnf-domain),
varname(Option, cnf-option),

string(Phase),
string(Action),
string(Domain),
string(Option),

stateref([RefIe,X]),
%process X

save_stateref(stateref([RefCa,[RefIe,Phase, Action, Domain, Option]])),
reconsult(staterefs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%State schma for the class ClsSoftGoal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
varname(_, refsoftgoal).
varname(_, cnfactions).

schema_type([RefSg, [RefIe, RefCas]], softgoal):-

%Variables and partial functions
varname(RefSg, refsoftgoal),
varname(RefIe, refintentionalelt),
varname(RefCas, cnfactions),

stateref([RefIe, X]),
%Process RefIe and X
```

```
save_stateref(stateref([RefSg, [RefIe, RefCas]])),
reconsult(staterefs).
```

D.0.2 Upward validation

typechecking with CZT 1.5.0

```
line 74 column 9 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol id
line 82 column 5 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 91 column 2 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol COLON
line 103 column 10 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 104 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 105 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 106 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 107 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 108 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 108 column 27 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol SPOT
line 111 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 111 column 42 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 111 column 65 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 111 column 90 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 117 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 119 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 125 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 127 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 132 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 134 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 139 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 140 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 149 column 10 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 150 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 151 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 152 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 153 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 154 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 154 column 27 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol SPOT
line 157 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 157 column 42 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 157 column 65 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 157 column 90 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 163 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 165 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
```

```
line 171 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 173 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 178 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 180 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 185 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 186 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 195 column 10 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 196 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 197 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 198 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 199 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 200 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 200 column 27 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol SPOT
line 203 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 203 column 42 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 203 column 65 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 203 column 90 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 209 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 211 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 217 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 219 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 224 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 226 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 231 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 232 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 241 column 10 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 242 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 243 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 244 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 245 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 246 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 246 column 27 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol SPOT
line 249 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 249 column 42 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 249 column 65 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 249 column 90 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 255 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 257 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 263 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 265 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 270 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 272 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 277 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 278 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 282 column 9 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol id
```

```
line 283 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \defs
line 283 column 26 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \bbar
line 283 column 39 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \bbar
line 283 column 49 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \bbar
line 291 column 9 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol id
line 295 column 9 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol id
line 301 column 12 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 306 column 9 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol id
line 309 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 310 column 12 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 310 column 24 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 310 column 61 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \bbar
line 312 column 8 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 312 column 20 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 312 column 57 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \bbar
line 312 column 76 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 319 column 9 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol id
line 323 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 339 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 340 column 12 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 341 column 11 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 342 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 343 column 11 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \int
line 346 column 0 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol DELTA
line 348 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 349 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 354 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 355 column 24 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \uni
line 360 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 361 column 22 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \uni
line 371 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 372 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 373 column 12 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 375 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 376 column 12 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \subs
line 379 column 0 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol DELTA
line 381 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 382 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 387 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 388 column 24 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 393 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 394 column 24 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \uni
line 409 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 412 column 0 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol DELTA
line 413 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
```

line 415 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \exi
line 420 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 421 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 436 column 0 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol DELTA
line 438 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 453 column 30 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \prod
line 453 column 41 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \prod
line 466 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 467 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \uni
line 467 column 33 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 473 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 474 column 17 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \uni
line 474 column 29 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 478 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 480 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 481 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \uni
line 481 column 39 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 491 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 498 column 11 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol DOT
line 500 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 507 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 517 column 8 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 520 column 12 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 525 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 527 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 539 column 5 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 545 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 547 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 549 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 565 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 566 column 5 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 583 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 584 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 600 column 10 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 601 column 7 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 602 column 13 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 602 column 20 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \seqone
line 603 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 604 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 604 column 27 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol SPOT
line 605 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \zimp
line 606 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \subs
line 614 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 615 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 616 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \exi

```
line 616 column 22 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \cbar
line 616 column 32 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \cbar
line 616 column 50 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \subs
line 617 column 29 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 617 column 41 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 629 column 8 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 630 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 632 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 633 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 633 column 5 in "C:\Users\Thesis\phdAppendC.tex":
         Syntax error in variable declaration at token data;
         an expression is expected after token COLON
line 633 column 36 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \zimp
line 633 column 42 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \exione
line 634 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 639 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 641 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \exi
line 641 column 5 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol data
line 641 column 15 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \cbar
line 641 column 25 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 642 column 23 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 655 column 11 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 656 column 6 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 658 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 659 column 27 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \subs
line 664 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 665 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \exi
line 665 column 20 in "C:\Users\Thesis\phdAppendC.tex":
         Syntax error in variable declaration at token messagein;
         an expression is expected after token COLON
line 665 column 43 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \cbar
line 665 column 52 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 666 column 11 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \map
line 667 column 17 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 668 column 29 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 685 column 7 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 686 column 9 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 686 column 15 in "C:\Users\Thesis\phdAppendC.tex":
         Syntax error in variable declaration at token POLY;
         an expression is expected after token COLON
line 687 column 10 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \pset
line 687 column 16 in "C:\Users\Thesis\phdAppendC.tex":
         Syntax error in variable declaration at token POLY;
         an expression is expected after token COLON
line 688 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
```

```
line 689 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \all
line 689 column 42 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 692 column 53 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \
line 695 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \begin
line 695 column 7 in "C:\Users\Thesis\phdAppendC.tex": Syntax error at symbol
         genschemaAddActorid
line 697 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 698 column 17 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 699 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \end
line 703 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 704 column 14 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \perp
line 705 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 710 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
line 711 column 21 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \fovr
line 716 column 0 in "C:\Users\Thesis\phdAppendC.tex": Unknown latex command \ST
```

Calculating the precondition with Z/Eves

```
Allegro CL for Windows Copyright (C) 1992-1997, Franz Inc., Berkeley, CA, USA.
All rights reserved.
=>
Reading file C:\Users\Dongmc\User Data\PHD Package\Thesis\zevesch6.tex
given sets GrlE, OzE
schema Grlmodel
... axiom Grlmodel\$declarationPart
schema InitGrlmodel
... axiom InitGrlmodel\$declarationPart
theorem CanInitGrlmodel
... theorem CanInitGrlmodel
schema Ozspec
... axiom Ozspec\$declarationPart
schema InitOzspec
... axiom InitOzspec\$declarationPart
theorem CanInitOzspec
... theorem CanInitOzspec
declaration of ozgrle
... axiom ozgrle\$declaration
schema StateOzGrl
... axiom Grlmodel\$thetasEqual
... axiom Grlmodel\$inSet
```

```
... axiom Grlmodel\$thetaInSet
... axiom Grlmodel\$setInPowerSet
... axiom Grlmodel\$member
... axiom Grlmodel\$thetaMember
... axiom Grlmodel\$declaration
... axiom Ozspec\$thetasEqual
... axiom Ozspec\$inSet
... axiom Ozspec\$thetaInSet
... axiom Ozspec\$setInPowerSet
... axiom Ozspec\$member
... axiom Ozspec\$thetaMember
... axiom Ozspec\$declaration
... axiom Grlmodel\$select\$grlelements
... axiom Grlmodel\$select\$links
... axiom Ozspec\$select\$ozelements
... axiom Ozspec\$select\$related
... axiom StateOzGrl\$declarationPart
schema InitStateOzGrl
... axiom InitStateOzGrl\$declarationPart
theorem CanInitStateOzGrl
... theorem CanInitStateOzGrl
schema FormaliseGrlmodel
... schema \Delta StateOzGrl
... axiom Delta\$StateOzGrl\$declarationPart
... axiom StateOzGrl\$thetasEqual
... axiom StateOzGrl\$select\$ozgrl
... axiom FormaliseGrlmodel\$declarationPart
theorem preFormaliseGrlmodel
... theorem preFormaliseGrlmodel
Done.
=> try \forall StateOzGrl; grlm?:Grlmodel \spot \pre FormaliseGrlmodel; prove by reduce;
Beginning proof of ...

=> try \forall StateOzGrl; grlm?:Grlmodel \spot \pre FormaliseGrlmodel; prove by reduce;
Beginning proof of ...

               StateOzGrl \\
```

```
        \land grlm? \in Grlmodel \\
\implies
 (\exists
    ozgrl':
      \power (\lblot grlelements: \power GrlE,
                     links: \power (GrlE \cross GrlE) \rblot
              \cross \lblot ozelements: \power OzE,
                            related: \power (OzE \cross OzE) \rblot);
    ozspec!:
      \lblot ozelements: \power OzE, related: \power (OzE \cross OzE) \rblot
    @ FormaliseGrlmodel)
Which simplifies
with invocation of \Delta StateOzGrl, FormaliseGrlmodel, StateOzGrl
when rewriting with mapDef, weakening
forward chaining using Delta\$StateOzGrl\$declarationPart,
FormaliseGrlmodel\$declarationPart, KnownMember\$declarationPart, knownMember,
StateOzGrl\$declarationPart, `[internal items]`
with the assumptions `&oplus$declaration`, StateOzGrl\$select\$ozgrl,
ozgrle\$declaration, select\_2\_1, select\_2\_2, `&map$declaration`,
pfun\_type, Ozspec\$declaration, Grlmodel\$declaration, `[internal items]` to
...

              ozgrl \in Grlmodel \pfun Ozspec \\
        \land (\forall grl: Grlmodel; oz: Ozspec | (grl, oz) \in ozgrl
                 @ (\forall grle: GrlE | grle \in grl.grlelements
                      @ (\exists oze: OzE
                           @ (      oze \in oz.ozelements \\
                              \land (grle, oze) \in ozgrle)))) \\
        \land grlm? \in Grlmodel \\
\implies
 (\exists
    @       ozgrl \oplus \{(grlm?, ozspec!)\}
            \in \power (\lblot grlelements: \power GrlE,
                               links: \power (GrlE \cross GrlE) \rblot
                        \cross \lblot ozelements: \power OzE,
                                       related:
                                         \power (OzE \cross OzE) \rblot) \\
```

```
      \land ozspec!
            \in \lblot ozelements: \power OzE,
                       related: \power (OzE \cross OzE) \rblot \\
      \land ozgrl \oplus \{(grlm?, ozspec!)\} \in Grlmodel \pfun Ozspec \\
      \land (\forall
               grl\_\_0: Grlmodel; oz\_\_0: Ozspec
                 | (grl\_\_0, oz\_\_0) \in ozgrl \oplus \{(grlm?, ozspec!)\}
               @ (\forall grle\_\_0: GrlE | grle\_\_0 \in grl\_\_0.grlelements
                    @ (\exists oze\_\_0: OzE
                         @ (      oze\_\_0 \in oz\_\_0.ozelements \\
                            \land (grle\_\_0, oze\_\_0) \in ozgrle)))) \\
      \land ozspec! \in Ozspec)
Rearranging gives ...

               ozgrl \in Grlmodel \pfun Ozspec \\
         \land grlm? \in Grlmodel \\
         \land (\forall grl: Grlmodel; oz: Ozspec | (grl, oz) \in ozgrl
                  @ (\forall grle: GrlE | grle \in grl.grlelements
                       @ (\exists oze: OzE
                            @ (      oze \in oz.ozelements \\
                               \land (grle, oze) \in ozgrle)))) \\
\implies
 (\exists
    ozspec!:
      \lblot ozelements: \power OzE, related: \power (OzE \cross OzE) \rblot
    @       ozspec! \in Ozspec \\
      \land
       ozgrl \oplus \{(grlm?, ozspec!)\}
       \in \power (\lblot grlelements: \power GrlE,
                           links: \power (GrlE \cross GrlE) \rblot
                    \cross \lblot ozelements: \power OzE,
                                  related: \power (OzE \cross OzE) \rblot) \\
      \land ozgrl \oplus \{(grlm?, ozspec!)\} \in Grlmodel \pfun Ozspec \\
      \land (\forall
               grl\_\_0: Grlmodel; oz\_\_0: Ozspec
                 | (grl\_\_0, oz\_\_0) \in ozgrl \oplus \{(grlm?, ozspec!)\}
               @ (\forall grle\_\_0: GrlE | grle\_\_0 \in grl\_\_0.grlelements
```

```
                    @ (\exists oze\_\_0: OzE
                         @ (      oze\_\_0 \in oz\_\_0.ozelements \\
                            \land (grle\_\_0, oze\_\_0) \in ozgrle)))))
Which simplifies
when rewriting with overrideInPfun, unitInPfun, applicationInDeclaredRangeFun,
rel\_type, Ozspec\$setInPowerSet, CrossSubsetCross2, inPowerSelf,
Grlmodel\$setInPowerSet, unitInRel, weakening, rel\_sub, power\_sub,
tupleInCross2
forward chaining using KnownMember\$declarationPart, knownMember,
‘[internal items]‘
with the assumptions relDefinition, fun\_type, ‘&oplus$declaration‘,
ozgrle\$declaration, select\_2\_1, select\_2\_2, pfun\_type,
Ozspec\$declaration, Grlmodel\$declaration, ‘[internal items]‘ to ...

               ozgrl \in Grlmodel \pfun Ozspec \\
         \land grlm? \in Grlmodel \\
         \land (\forall grl: Grlmodel; oz: Ozspec | (grl, oz) \in ozgrl
                  @ (\forall grle: GrlE | grle \in grl.grlelements
                       @ (\exists oze: OzE
                            @ (      oze \in oz.ozelements \\
                               \land (grle, oze) \in ozgrle)))) \\
\implies
 (\exists
    ozspec!:
      \lblot ozelements: \power OzE, related: \power (OzE \cross OzE) \rblot
    @       ozspec! \in Ozspec \\
      \land (\forall
               grl\_\_0: Grlmodel; oz\_\_0: Ozspec
                 | (grl\_\_0, oz\_\_0) \in ozgrl \oplus \{(grlm?, ozspec!)\}
               @ (\forall grle\_\_0: GrlE | grle\_\_0 \in grl\_\_0.grlelements
                    @ (\exists oze\_\_0: OzE
                         @ (      oze\_\_0 \in oz\_\_0.ozelements \\
                            \land (grle\_\_0, oze\_\_0) \in ozgrle)))))
Proving gives ...

               ozgrl \in Grlmodel \pfun Ozspec \\
         \land grlm? \in Grlmodel \\
```

```
        \land (\forall grl: Grlmodel; oz: Ozspec | (grl, oz) \in ozgrl
                @ (\forall grle: GrlE | grle \in grl.grlelements
                     @ (\exists oze: OzE
                          @ (      oze \in oz.ozelements \\
                             \land (grle, oze) \in ozgrle)))) \\
\implies
 (\exists
    ozspec!:
      \lblot ozelements: \power OzE, related: \power (OzE \cross OzE) \rblot
    @       ozspec! \in Ozspec \\
      \land (\forall
               grl\_\_0: Grlmodel; oz\_\_0: Ozspec
                 | (grl\_\_0, oz\_\_0) \in ozgrl \oplus \{(grlm?, ozspec!)\}
               @ (\forall grle\_\_0: GrlE | grle\_\_0 \in grl\_\_0.grlelements
                    @ (\exists oze\_\_0: OzE
                         @ (      oze\_\_0 \in oz\_\_0.ozelements \\
                            \land (grle\_\_0, oze\_\_0) \in ozgrle)))))
Command had no effect.
=>
```

Bibliography

[1] ITU-T, Recommendation Z.151 (10/12), User Requirements Notation (URN)-Language definition., October 2012. URL http://www.itu.int/rec/T-REC-Z.151-201210-I/enlastaccessed-04/2015.

[2] Tawfig Abdelaziz, Mohamed Elammari, and Rainer Unland. Visualizing a Multiagent-Based Medical Diagnosis System Using a Methodology Based on Use Case Maps. In Gabriela Lindemann, J. Denzinger, Ingo Timm, and Rainer Unland, editors, *Multiagent System Technologies*, volume 3187 of *Lecture Notes in Computer Science*, pages 545–559. Springer Berlin / Heidelberg. ISBN 978-3-540-23222-3. URL http://dx.doi.org/10.1007/978-3-540-30082-3_15.

[3] Matoussi Abderrahman, Frédériç Gervais, and Régine Laleau. A Goal-Based Approach to Guide the Design of an Abstract Event-B Specification. In *Proceedings of the 2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS '11, pages 139–148, 2011. ISBN 978-0-7695-4381-9.

[4] Jean-Raymond Abrial. Formal methods in industry: achievements, problems, future. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 761–768, New York, NY, USA, 2006. ACM. ISBN 1-59593-375-1. URL {http://doi.acm.org/10.1145/1134285.1134406}.

[5] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, 12:447–466. ISSN 1433-2779.

[6] M Ahmad, J Araujo, N Belloir, J.-M. Bruel, C Gnaho, R Laleau, and F Semmak. Self-adaptive systems requirements modelling: Four related approaches comparison. In *Comparing Requirements Modeling Approaches Workshop (CMA@RE), 2013 International*, pages 37–42, July 2013. doi: 10.1109/CMA-RE.2013.6664183.

[7] Edward B. Allen. Typesetting Technical Reports that Include Z Specifications Using LaTeX, 2006.

[8] D Ameller, C Ayala, J Cabot, and X Franch. Non-functional Requirements in Architectural Decision Making. *Software, IEEE*, 30(2):61–67, mar 2013. ISSN 0740-7459. doi: 10.1109/MS.2012.176.

[9] Daniel Amyot. Use Case Maps: Quick Tutorial, September 1999.

[10] Daniel Amyot. Introduction to the user requirements notation: learning by example. *Computer Networks*, 42(3):285–301, June 2003. ISSN 1389-1286.

[11] Daniel Amyot. Use Case Maps as a Feature Description Notation. In *FIREworks Feature Constructs Workshop*, pages 27–44. Springer-Verlag, May, 2000.

[12] Daniel Amyot and R. Andrade. Description of Wireless Intelligent Network Services with Use Case Maps. In *SBRC'99, 17th Brazilian Symposium on Computer Networks*, Salvador, Brazil, May 1999.

[13] Daniel Amyot and Gunter Mussbacher. URN: Towards a New Standard for the Visual Description of Requirements. In *Proceedings of the 3rd International Conference on Telecommunications and Beyond: The Broader Applicability of SDL and MSC*, SAM'02, pages 21–37, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-00877-2. URL `http://0-dl.acm.org.oasis.unisa.ac.za/citation.cfm?id=1765408.1765411`.

[14] Daniel Amyot and Gunter Mussbacher. User Requirements Notation: The First Ten Years, The Next Ten Years (Invited Paper). *Journal of Software*, 6(5), 2011. URL `http://ojs.academypublisher.com/index.php/jsw/article/view/0605747768`.

[15] Daniel Amyot, Luigi Logrippo, and Michael Weiss. Generation of test purposes from use case maps. *Comput. Netw.*, 49(5):643–660, December 2005. ISSN 1389-1286. URL `http://dx.doi.org/10.1016/j.comnet.2005.05.006`.

[16] Daniel Amyot, Sepideh Ghanavati, Jennifer Horkoff, Gunter Mussbacher, Liam Peyton, and Eric Yu. Evaluating goal models within the goal-oriented requirement language. *Int. J. Intell. Syst.*, 25(8):841–877, August 2010. ISSN 0884-8173.

[17] Daniel Amyot, Rouzbahan Rashidi-Tabrizi, Gunter Mussbacher, Jason Kealey, Etienne Tremblay, and Jennifer Horkoff. Improved GRL Modeling and Analysis with jUCMNav 5. In *iStar*, pages 137–139, 2013.

[18] Daniel Amyot, R. J. A. Buhr, T. Gray, and L. Logrippo. Use Case Maps for the Capture and validation of Distributed Systems Requirements. In *ISRE'99, Fourth International Symposium on Requirements Engineering*, pages 44–53, Limerick, Ireland, June 1999.

[19] Amyot, Daniel and Roy, Jean-François and Weiss, Michael. UCM-Driven Testing of Web Applications. In *SDL Forum'05*, pages 247–264, 2005.

[20] Annie I. Anton. Goal-based requirements analysis. In *Proceedings of the 2Nd International Conference on Requirements Engineering (ICRE '96)*, ICRE '96, pages 136–, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7252-8. URL `http://dl.acm.org/citation.cfm?id=850944.853130`.

[21] Annie I. Anton and Colin Potts. The use of goals to surface requirements for evolving systems. In *International Conference on Software Engineering*, pages 157–166, Kyoto, April 1998. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=671112`.

[22] Sara Baase and Allen Van Gelder. *Computer algorithms : Introduction to design and analysis*. Addison-Wesley, Delhi, 2000. ISBN 0-201-61244-5. URL `www.awlonline.com/cs`.

[23] Martin L Barrett. Putting Non-functional Requirements to Good Use. *J. Comput. Sci. Coll.*, 18(2):271–277, December 2002. ISSN 1937-4771. URL `http://0-dl.acm.org.oasis.unisa.ac.za/citation.cfm?id=771322.771361`.

[24] Saeed Ahmadi Behnam, Daniel Amyot, Alan J. Forster, Liam Peyton, and Azalia Shamsaei. Goal-driven development of a patient surveillance application for improving patient safety. In Gilbert Babin, Peter Kropf, Michael Weiss, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *E-Technologies: Innovation in an Open World*, volume 26 of *Lecture Notes in Business Information Processing*, pages 65–76. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01187-0. URL `{http://dx.doi.org/10.1007/978-3-642-01187-0\_6}`.

[25] J. C. Bicarregui, J. S Fitzgerald, P. G. Larsen, and J. C. Woodcock. Industrial practice in formal methods: A review. In *Proceedings of the 2nd World Congress on Formal Methods*, FM '09, pages 810–813, 2009. ISBN 978-3-642-05088-6.

[26] M. Bittner and F. Kammuller. Translating Fusion/UML to Object-Z. In *Formal Methods and Models for Co-Design, 2003. MEMOCODE '03. Proceedings. First ACM and IEEE International Conference on*, pages 49–50, June 2003. doi: 10.1109/MEMCOD.2003.1210087.

[27] Barry W. Boehm. Verifying and validating software requirements and design specifications. *IEEE software*, 1(1):75, 1984.

[28] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Weseley, 2nd edition, 2005. ISBN 0-321-26797-4.

[29] F. Bordeleau and R. J. A. Buhr. The UCM-ROOM Design Method: from Use Case Maps to Communicating State Machines. In *Conference on the Engineering of Computer-Based System*, Monterey, USA, 1997.

[30] Jonathan Bowen. *Formal Specification and Documentation Using Z: A Case Study Approach*. International Thomson Computer Press, London / Boston, 1996. ISBN 1-85032-230-9.

[31] Jonathan P. Bowen and Michael G. Hinchey. Ten Commandments of Formal Methods ... Ten Years Later. *Computer*, 39:40–48, January 2006. ISSN 0018-9162. URL `http://portal.acm.org/citation.cfm?id=1110638.1110672`.

[32] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, (3):203–236. ISSN 1387-2532. doi: 10.1023/B:AGNT.0000018806.20944.ef.

[33] Frederick P. Jr. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *Information Processing '86, IFIP 1986*, 1986.

[34] R. J. A. Buhr. Use Case Maps: A New Model to Bridge the Gap Between Requirements and Design. *SCE 95- Contribution to the OOPSLA 95 Use Case Map Workshop*, pages 1–4, 1995.

[35] R. J. A. Buhr. Understanding Macroscopic Behavior Patterns with Use-Case Maps. In Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson, editors, *Building Application Frameworks - Object-Oriented Foundations of Framework Design*, pages 415–439. John Wiley & Sons, New York, 1999.

[36] R. J. A. Buhr. Use Case Maps as Architectural Entities for Complex Systems. In IEEE, editor, *Transaction on Software Engineering*, pages 1131–1155, December 1998.

[37] R. J. A. Buhr and R. S. Casselman. *Use Case Maps for Object-Oriented Systems*. Printice Hall, USA, 1999.

[38] R. J. A. Buhr, Daniel Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. Features-Interaction Visualisation and Resolution in an Agent Environment. In K. Kimbler and L. G. Bouma, editors, *FIW'98, Fifth International Workshop on Feature Interaction in Telecommunications and Software Systems*, Lund, Sweden, 1998. IOS Press, 135-149.

[39] R. J. A. Buhr, Daniel Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. High Level, Multi-agent Prototypes from a Scenario-Path Notation: A Feature-Interaction Example. In H.S. Nwana (Eds) and D.T. Ndumu, editors, *PAAM'98, Third Conference on Practical Application of Intelligent Agents and Multi-Agents*, London, UK, 277-295, March 1998.

[40] Ron Burback. *SOFTWARE ENGINEERING METHODOLOGY:THE WATERSLUICE*. PhD thesis, Stanford University: Department of Computer Science, USA, December 1998. Principal adviser-Gio Wiederhold.

[41] Michael Butler. Decomposition Structures for Event-B. In Michael Leuschel and Heike Wehrheim, editors, *Integrated Formal Methods*, volume 5423 of *Lecture Notes in Computer Science*, pages 20–38. Springer Berlin / Heidelberg. ISBN 978-3-642-00254-0.

[42] Gustavo Cabral and Augusto Sampaio. Automated formal specification generation and refinement from requirement documents. *Journal of the Brazilian Computer Society*, 14(1):87 – 106, 03 2008. ISSN 0104-6500. doi: 10.1007/BF03192554. URL `http://www.scielo.br/scielo.php?script=sci\_arttextandpid=S0104-65002008000100008andnrm=iso`.

[43] Kai-Yuan Cai. Non-Functional Computing: Towards a More Scientific Treatment to Non-Functional Requirements. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 2, pages 493–494, July 2007. doi: 10.1109/COMPSAC.2007.156.

[44] Rafael Capilla, Muhammad Ali Babar, and Oscar Pastor. Quality requirements engineering for systems and software architecting: methods, approaches, and tools. *Requirements Engineering*, (4):255–258. ISSN 0947-3602. doi: 10.1007/s00766-011-0137-9.

[45] David Carrington and Graeme Smith. Extending Z for Object-Oriented Specifications. 5th Australian Software Engineering Conference, (Sydney), May, 1990.

[46] Anis Charfi, Benjamin Schmeling, Andreas Heizenreder, and Mira Mezini. Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In *2006 European*

Conference on Web Services (ECOWS'06), pages 23–34. IEEE. ISBN 0-7695-2737-X. doi: 10.1109/ECOWS.2006.32.

[47] S Chawla and S Srivastava. A Goal based methodology for Web specific Requirements Engineering. In *Information and Communication Technologies (WICT), 2012 World Congress on*, pages 173–178, October 2012. doi: 10.1109/WICT.2012.6409070.

[48] Sang-soo Choi, So-yeon Kim, and Gang-soo Lee. Enhanced Misuse Case Model: A Security Requirement Analysis and Specification Model. In *Proceedings of the 2006 International Conference on Computational Science and Its Applications - Volume Part V*, ICCSA'06, pages 618–625, Berlin, Heidelberg. Springer-Verlag. ISBN 3-540-34079-3, 978-3-540-34079-9. doi: 10.1007/11751649_68.

[49] L Chung and S Supakkul. Representing NFRs and FRs: A goal-oriented and use case driven approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3647 LNCS:29–41, 2005. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-33745127388\&partnerID=40\&md5=29aa326709f89c31664c72a65767782c`.

[50] Lawrence Chung and Brian A Nixon. Dealing with non-functional requirements: three experimental studies of a process-oriented approach. In *Proceedings of the 17th international conference on Software engineering*, pages 25–37. ACM, 1995.

[51] Kendra Cooper, Lirong Dai, and Yi Deng. Performance modeling and analysis of software architectures: An aspect-oriented {UML} based approach. *Science of Computer Programming*, (1):89–108. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2004.10.007.

[52] Agostino Cortesi and Francesco Logozzo. Abstract Interpretation-based Verification of Non-functional Requirements. In *Proceedings of the 7th International Conference on Coordination Models and Languages*, COORDINATION'05, pages 49–62, Berlin, Heidelberg. Springer-Verlag. ISBN 3-540-25630-X, 978-3-540-25630-4. doi: 10.1007/11417019_4.

[53] L M Cysneiros, K K Breitman, C Lopez, and H Astudillo. Querying Software Interdependence Graphs. In *Software Engineering Workshop, 2008. SEW '08. 32nd Annual IEEE*, pages 108–112, October 2008. doi: 10.1109/SEW.2008.28.

[54] Lirong Dai and Kendra Cooper. Modeling and performance analysis for security aspects. *Science of Computer Programming*, (1):58–71. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2005.11.006.

[55] Jose Luis de la Vara, Juan Sánchez, and Óscar Pastor. Business process modelling and purpose analysis for requirements analysis of information systems. In Zohra Bellahsène and Michel Léonard, editors, *Advanced Information Systems Engineering: 20th International Conference, CAiSE 2008 Montpellier, France, June 16-20, 2008 Proceedings*, pages 213–227. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-69534-9. doi: 10.1007/978-3-540-69534-9_17.

[56] Dursun Delen, Nikunj P. Dalal, and Perakath C. Benjamin. Integrated modeling: the key to holistic understanding of the enterprise. *Commun. ACM*, 48(4):107–112, April 2005. ISSN 0001-0782.

[57] John Derrick and Eerke Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. Formal Approaches to Computing and Information Technology. Springer-Verlag, London, UK, 2001. ISBN 1-85233-245-X. URL `http://www.cs.kent.ac.uk/people/staff/eab2/refbook/`.

[58] Rachida Djouab, Alain Abran, and Ahmed Seffah. An ASPIRE-based method for quality requirements identification from business goals. *Requirements Engineering, Springer London*, pages 1–20, 2014. ISSN 0947-3602. doi: 10.1007/s00766-014-0211-1.

[59] G Dobson, S Hall, and G Kotonya. A domain-independent ontology for non-functional requirements. In *Proceedings - ICEBE 2007: IEEE International Conference on e-Business Engineering - Workshops: SOAIC 2007; SOSE 2007; SOKM 2007*, pages 563–566, 2007. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-47349085554\&partnerID=40\&md5=98af8ab72df9d3b853cff788bee81ac6`.

[60] Cyrille Dongmo. Towards the Formalisation of Use Case Maps(UCMs). Master's thesis, School of Computing, University of South Africa(Unisa), 2011. URL `{http://uir.unisa.ac.za/handle/10500/5621}`.

[61] Cyrille Dongmo and John A. van der Poll. A Four-Way Framework for Validating a Specification. In Paula Kotze, Aurona Gerber, Alta van der Merwe, and Nicola Bidwell, editors, *SAICSIT*, pages 46–59. ACM PRESS, 2010. ISBN 978-1-60558-950-3.

[62] Cyrille Dongmo and John Andrew Van der Poll. An application of a four-way framework for validating a specification: Animating an object-z specification using prolog. In *Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology-CCIT 2014*, . ISBN 978-1-63248-051-4.

[63] Cyrille Dongmo and John Andrew Van der Poll. Exploiting enterprise organograms to facilitate goal/requirements elicitation. In *Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology-CCIT 2014*, . ISBN 978-1-63248-051-4.

[64] Cyrille Dongmo and John Andrew van der Poll. Addressing the Construction of Z and Object-Z with Use Case Maps (UCMs). *International Journal of Software Engineering and Knowledge Engineering*, 24(02):285–327, 2014. doi: 10.1142/S0218194014500120.

[65] Cyrille Dongmo and John Andrew Van der Poll. An application of a four-way framework for validating a specification: Animating an object-z specification using prolog. *International Journal of Software Engineering and Research Methodology*, 2(1):10–19, 2015. ISSN 2374 - 1619. URL `http://www.seekdl.org/journal_page_papers.php?jourid=131&issueid=148`.

[66] Cyrille Dongmo and John Andrew Van der Poll. Exploiting enterprise organograms to facilitate goal/requirements elicitation. *International Journal of Software Engineering and Research Methodology*, 2(1):20–29, 2015. ISSN 2374 - 1619. URL `http://www.seekdl.org/journal_page_papers.php?jourid=131&issueid=148`.

[67] E. P. Doolan. Experience with fagan's inspection method. *Softw., Pract. Exper.*, 22(2): 173–182, 1992. ISSN 1097-024X. URL `http://dblp.uni-trier.de/db/journals/spe/spe22.html#Doolan92`.

[68] Roger Duke and Gordon Rose. *Formal Object-Oriented Specification Using Object-Z*. Macmillan, Basingstoke, 2000. ISBN 0333801237.

[69] Roger Duke, Paul King, Gordon A. Rose, and Graeme Smith. The Object-Z Specification Language. In Timothy D. Korson, Vijay Vashnavi, and Bertrand Meyer, editors, *TOOLS (5)*, pages 465–484. Prentice Hall, 1991. ISBN 0-13-923178-1.

[70] H.a Espinoza, H.a Dubois, S.a Gérard, J.b Medina, D.C.c Petriu, and M.c Woodside. Annotating UML models with non-functional properties for quantitative analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3844 LNCS:79–90, 2006. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-33745648549\&partnerID=40\&md5=9657a0ed9eafcf1e3b416c1d0e6e31a0`.

[71] Michael E Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 38(2-3):258–287, June 1999. ISSN 0018-8670.

[72] W M Farid. The NORMAP Methodology: Lightweight Engineering of Non-functional Requirements for Agile Processes. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 322–325, December 2012. doi: 10.1109/APSEC.2012.23.

[73] W M Farid and F J Mitropoulos. Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes. In *Southeastcon, 2012 Proceedings of IEEE*, pages 1–7, March 2012. doi: 10.1109/SECon.2012.6196988.

[74] Ye Fei and Zhu Xiaodong. An XML-Based Software Non-Functional Requirements Modeling Method. In *Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference on*, pages 2–380, August 2007. doi: 10.1109/ICEMI.2007.4350695.

[75] John S. Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef. *Vienna Development Method*. John Wiley and Sons, Inc. ISBN 9780470050118.

[76] Leonardo Freitas. Proving theorems with z/eves. *Appendix A*, 1(1), 2004. URL `https://www.cs.york.ac.uk/ftpdir/pub/leo/york-msc-2007/information/zeves/tutorials/CRG-3.pdf`.

[77] Peter B. Galvin, Greg Gagne, and Abraham Silberschatz. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 9th edition, 2013. ISBN 1118093755, 9781118093757.

[78] Susan Gerhart, Dan Craigen, and Ted Ralston. Experience with formal methods in critical systems. *High-Integrity System Specification and Design*, page 413, 2012.

[79] C Gnaho, F Semmak, and R Laleau. An overview of a SysML extension for goal-oriented NFR modelling: Poster paper. In *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, pages 1–2, May 2013. doi: 10.1109/RCIS.2013.6577734.

[80] Martin Gogolla and Mark Richters. On Combining Semi-Formal and Formal Object Specification Techniques. In *Recent trends in algebraic development techniques: 12th international workshop, WADT97*, pages 238–252. Springer, 1998.

[81] Gemma Grau and Xavier Franch. A Goal-oriented Approach for the Generation and Evaluation of Alternative Architectures. In *Proceedings of the First European Conference on Software Architecture*, ECSA'07, pages 139–155, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75131-9, 978-3-540-75131-1. URL `http://0-dl.acm.org.oasis.unisa.ac.za/citation.cfm?id=2394215.2394230`.

[82] Jeff Gray and Stephen Schach. Constraint animation using an object-oriented declarative language. In *Proceedings of the 38th Annual on Southeast Regional Conference*, ACM-SE 38, pages 1–10, 2000. ISBN 1-58113-250-6.

[83] Sarthak Grover and Nigamanth Sridhar. GenQA: Automated Addition of Architectural Quality Attribute Support for Java Software? In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 483–487, New York, NY, USA. ACM. ISBN 978-1-60558-166-8. doi: 10.1145/1529282.1529385.

[84] R.a Guizzardi, F.-L.b Li, A.c Borgida, G.a Guizzardi, J.b Horkoff, and J.b Mylopoulos. An ontological interpretation of non-functional requirements. *Frontiers in Artificial Intelligence and Applications*, 267:344–357, 2014. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84907001173\&partnerID=40\&md5=b07611f375aa422d42bee63e9ba2fde0`.

[85] Anthony Hall. *Industrial-Strength Formal Methods in Practice*, chapter Using Formal Methods to Develop an ATC Information System, pages 207–229. Springer London, London, 1999. ISBN 978-1-4471-0523-7. doi: 10.1007/978-1-4471-0523-7_10.

[86] Brahim Hamid and Christian Percebois. A modeling and formal approach for the precise specification of security patterns. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 95–112. ISBN 9783319048963. doi: 10.1007/978-3-319-04897-0-7.

[87] W. Hasselbring. *Z User Workshop, Cambridge 1994: Proceedings of the Eighth Z User Meeting, Cambridge 29–30 June 1994*, chapter Animation of Object-Z Specifications with a Set-Oriented Prototyping Language, pages 337–356. Springer London, London, 1994. ISBN 978-1-4471-3452-7. doi: 10.1007/978-1-4471-3452-7_20. URL `http://dx.doi.org/10.1007/978-1-4471-3452-7_20`.

[88] Jameleddine Hassine. Describing and assessing availability requirements in the early stages of system development. *Software & Systems Modeling*, pages 1–25. ISSN 1619-1366. doi: 10.1007/s10270-013-0382-0.

[89] Les Hatton. Does OO Sync with How We Think? *IEEE Softw.*, 15(3):46–54, 1998. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/52.676735.

[90] Anne Elisabeth Haxthausen. *An Introduction to Formal Methods for the Development of Safety-critical Applications*. 2010. This report is a delivery to The Danish Government's railway authority, Trafikstyrelsen, as a part of the Public Sector Consultancy service offered by the Technical University of Denmark.

[91] Andrea Herrmann and Barbara Paech. MOQARE: misuse-oriented quality requirements engineering. *Requirements Engineering*, 13(1):73–86, 2008. ISSN 0947-3602. doi: 10.1007/s00766-007-0058-9.

[92] Erik Hofstee. *Constructing a Good Dissertation: A Practical Guide to Finishing a Master's, MBA or PhD on Schedule*. EPE, 2006. ISBN 0-9585007-1-1.

[93] Gerard Horgan and Souheil Khaddaj. Use of an adaptable quality model approach in a production support environment. *Journal of Systems and Software*, (4):730–738. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2008.10.009.

[94] IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. Technical report, IEEE, 1998. URL `http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=720574`.

[95] Jonathan Jacky. *The way of Z: practical programming with formal methods*. Cambridge University Press, New York, NY, USA, 1996. ISBN 0-521-55976-6.

[96] Kealy Jason. Enhanced Use Case Map Analysis and Transformation Tooling. Master's thesis, 2007. Ottawa-Carleton Institute for Computer Science, Canada.

[97] Mouton Johann. *How to succeed in your Master's and Doctoral Studies: A South African Guide and Resource Book*. Van Schaik, 1064 Arcadia Street, Hatfield, Pretoria, South Africa, 1st edition, 2001. ISBN 0 627 02484 X.

[98] Wendy Johnston. A Type Checker for Object-Z. Technical report, Department of Computer Science, The University of Queensland Australia, 1996.

[99] Hyo Taeg Jung and Gil-Haeng Lee. A systematic software development process for non-functional requirements. In *2010 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 431–436. IEEE, 2010.

[100] Chanwit Kaewkasi and Wanchai Rivepiboon. WWM: A Practical Methodology for Web Application Modeling. In *COMPSAC*, pages 603–608. IEEE Computer Society, 2002. ISBN 0-7695-1727-7.

[101] Erik Kamsties. *Engineering and Managing Software Requirements*, chapter Understanding Ambiguity in Requirements Engineering, pages 245–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-28244-0. doi: 10.1007/3-540-28244-0_11.

[102] M Kassab, O Ormandjieva, and M Daneva. An Ontology Based Approach to Non-functional Requirements Conceptualization. In *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, pages 299–308, September 2009. doi: 10.1109/ICSEA.2009.50.

[103] Suntae Kim, Dae-Kyoo Kim, Lunjin Lu, and Soo-Yong Park. A Tactic-Based Approach to Embodying Non-functional Requirements into Software Architectures. In *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 139–148. IEEE, September 2008. ISBN 978-0-7695-3373-5. doi: 10.1109/EDOC.2008.18.

[104] M. Kirikova and J. Bubenko. *Enterprise Modelling: Improving the Quality of Requirements Specifications*. Report series - Department of Computer & Systems Sciences. DSV, 1994. URL `https://books.google.co.za/books?id=G1ZzMwAACAAJ`.

[105] Barbara Kitchenham. Procedures for performing systematic reviews. Technical report, Keele University and NICTA, 2004.

[106] Alexei Lapouchnian. Goal-Oriented Requirements Engineering: An Overview of the Current Research. Technical report, Department of Computer Science, University of Toronto, Toronto, Canada, Jun 2005.

[107] Jonathan Lee and Kuo-Hsun Hsu. Modeling software architectures with goals in virtual university environment. *Information and Software Technology*, (6):361–380. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/S0950-5849(02)00021-6.

[108] Isaac Lera, Carlos Juiz, and Ramon Puigjaner. Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems. *Science of Computer Programming*, (1):27–37. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2005.11.003.

[109] David Lightfoot. *Formal Specification Using Z*. Grassroots Series. Palgrave, 2nd edition, 2001.

[110] Lin Liu and Eric Yu. Designing information systems in social context: A goal and scenario modelling approach. *Inf. Syst.*, 29(2):187–203, April 2004. ISSN 0306-4379.

[111] Shaoying Liu and Hao Wang. An automated approach to specification animation for validation. *J. Syst. Softw.*, 80(8):1271–1285, August 2007. ISSN 0164-1212.

[112] Xiaoqing (Frank) Liu, Manooch Azmoodeh, and Nektarios Georgalas. Specification of Non-functional Requirements for Contract Specification in the NGOSS Framework for Quality Management and Product Evaluation. In *Fifth International Workshop on*

Software Quality (WoSQ'07: ICSE Workshops 2007), pages 7–7. IEEE, May . ISBN 0-7695-2959-3. doi: 10.1109/WOSQ.2007.12.

[113] Yi Liu, Zhiyi Ma, and Weizhong Shao. Integrating Non-functional Requirement Modeling into Model Driven Development Method. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 98–107, November 2010. doi: 10.1109/APSEC.2010.21.

[114] Grzegorz Loniewski, Etienne Borde, Dominique Blouin, and Emilio Insfran. Model-Driven Requirements Engineering for Embedded Systems Development. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pages 236–243. IEEE, September . ISBN 978-0-7695-5091-6. doi: 10.1109/SEAA.2013.48.

[115] Francisca Losavio, Nicole Levy, Parinaz Davari, and François Colonna. Pattern-based Architectural Design Driven by Quality Properties: A Platform to Model Scientific Calculation. In *Proceedings of the 2Nd European Conference on Software Architecture*, EWSA'05, pages 94–112, Berlin, Heidelberg. Springer-Verlag. ISBN 3-540-26275-X, 978-3-540-26275-6. doi: 10.1007/11494713_7.

[116] Glenn H. MacEwen. Specification prototyping. In *Proceedings of the Workshop on Rapid Prototyping*, pages 112–119, 1982. ISBN 0-89791-094-X.

[117] Petra Malik and Mark Utting. CZT: A Framework for Z Tools. *ZB2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users, Guildford, UK*, pages 65–84, 2005.

[118] Tegegne Marew, Joon-Sang Lee, and Doo-Hwan Bae. Tactics based approach for integrating non-functional requirements in object-oriented analysis and design. *Journal of Systems and Software*, 82(10):1642–1656, October 2009. ISSN 01641212. doi: 10.1016/j.jss.2009.03.032.

[119] Tim McComb and Graeme Smith. Animation of Object-Z Specifications Using a Z Animator. In *SEFM*, pages 191–200. IEEE Computer Society, 2003. ISBN 0-7695-1949-0.

[120] W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[121] Timothy Meline. Selecting studies for systematic review: Inclusion and exclusion criteria. *Contemporary Issues in Communication Science and Disorders*, 33(21-27), 2006.

[122] Janne Merilinna, Anton Yrjönen, and Tomi Räty. NFR+ framework method to support bi-directional traceability of non-functional requirements. *Computer Science - Research and Development*, pages 1–15. ISSN 1865-2034. doi: 10.1007/s00450-012-0205-5.

[123] Christel Michael and Kang Kyo. Issues in Requirements Elicitation (CMU/SEI-92-TR-012). Technical report, Software Engineering Institute, Carnegie Mellon University, 1992. URL `http://www.sei.cmu.edu/library/abstracts/reports/92tr012.cfm`.

[124] A. Miga. Application of Use Case Maps to System Design with tool Support. Master's thesis, Carleton University, Canada, 1998.

[125] Andrew Miga, Daniel Amyot, Bordeleau Francis, Donald Cameron, and Murray Woodside. Deriving Message Sequence Charts from Use Case Maps Scenario Specifications. In *Maps Scenario Specifications. 10th SDL Forum*, pages 268–287. Springer, 2001.

[126] Tim Miller, Leo Freitas, Petra Malik, and Mark Utting. Czt support for z extensions. In Judi Romijn, Graeme Smith, and Jaco van de Pol, editors, *IFM*, volume 3771 of *Lecture Notes in Computer Science*, pages 227–245. Springer, 2005. ISBN 3-540-30492-4. URL `http://dblp.uni-trier.de/db/conf/ifm/ifm2005.html#MillerFMU05`.

[127] H. Miyazaki and A. Tanaka. Study on representation of security aspects in each viewpoint using UML for ODP. In *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC*, 2007. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-52049119736\&partnerID=40\&md5=ea08d51208a28e89c249489f9f57bcf6`.

[128] Lakhoua M.N. and M. Rahmouni. Investigation of the Methods of Enterprise Modeling. *African Journal of Business Management*, 5(16):6845–6852, August 2011.

[129] Farid Mokhati and Yahia Menassel. Towards formalising use case maps in maude strategy language: application to multi–agent systems. *International Journal of Computer Applications in Technology*, 47(2-3):138–151, 2013.

[130] David Mole. Z - An Introduction to Formal Methods, by Antoni Diller, Wiley, 2nd Edition, 1994 (Book Review). *Softw. Test., Verif. Reliab.*, 4(3):191, 1994.

[131] Haralambos Mouratidis, Paolo Giorgini, and Gordon Manson. Modelling secure multiagent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems - AAMAS '03*, page 859, New York, New York, USA, July . ACM Press. ISBN 1581136838. doi: 10.1145/860575.860713.

[132] Haralambos Mouratidis, Paolo Giorgini, and Gordon Manson. When security meets software engineering: a case of modelling secure information systems. *Information Systems*, (8):609–629, December . ISSN 03064379. doi: 10.1016/j.is.2004.06.002.

[133] G Mussbacher, S Ghanavati, and Daniel Amyot. Modeling and Analysis of URN Goals and Scenarios with jUCMNav. In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 383–384, aug 2009. doi: 10.1109/RE.2009.56.

[134] Gunter Mussbacher. Models in software engineering. chapter Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models, pages 305–316. 2008. ISBN 978-3-540-69069-6.

[135] Gunter Mussbacher and Daniel Amyot. Goal and Scenario Modeling, Analysis, and Transformation with jUCMNav. In *ICSE Companion*, pages 431–432. IEEE, 2009. ISBN 978-1-4244-3494-7.

[136] Gunter Mussbacher, Daniel Amyot, and Michael Weiss. Visualizing Early Aspects with Use Case Maps. In Awais Rashid and Mehmet Aksit, editors, *Transactions on Aspect-Oriented Software Development III*, volume 4620 of *Lecture Notes in Computer Science*, pages 105–143. Springer Berlin / Heidelberg. ISBN 978-3-540-75161-8.

[137] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using non-functional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18:483–497, 1992.

[138] Mehrnaz Najafi and Hassan Haghighi. An Integration of UML-B and Object-Z in Software Development Process. In Khaled Elleithy and Tarek Sobh, editors, *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering*, volume 152 of *Lecture Notes in Electrical Engineering*, pages 633–648. Springer New York, 2013. ISBN 978-1-4614-3534-1. doi: 10.1007/978-1-4614-3535-8_53.

[139] Katsuhiko Nakamura. Introduction to logic programming by Christopher J. Hogger. *New Generation Computing*, 3(4):487–487, 1985. ISSN 1882-7055. doi: 10.1007/BF03037083.

[140] Q L Nguyen. Non-functional requirements analysis modeling for software product lines. In *Modeling in Software Engineering, 2009. MISE '09. ICSE Workshop on*, pages 56–61, May 2009. doi: 10.1109/MISE.2009.5069898.

[141] Martin S Olivier. *Information Technology Research — A Practical Guide for Computer Science and Informatics*. Van Schaik, Pretoria, South Africa, 2nd edition, 2009. ISBN 9780627027581.

[142] Gerard O'Regan. *Mathematical Approaches to Software Quality*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 184628242X.

[143] Carlos Otero. *Software engineering design: theory and practice*. CRC Press, 2012.

[144] Tim Parker. *TOZE-A Graphical Editor for the Object-Z Specification Language with Syntax and Type Checking Capabilities.* PhD thesis, University of Wisconsin-La Crosse, 2008. Promoter-Kotze, P. and Promoter-Labuschagne, W. A.

[145] Ajit K Patankar and Sadashiv Adiga. Enterprise integration modelling: a review of theory and practice. *Computer Integrated Manufacturing Systems*, 8(1):21 – 34, 1995. ISSN 0951-5240. doi: 10.1016/0951-5240(95)92810-H.

[146] Catherine Pickering and Jason Byrne. The benefits of publishing systematic quantitative literature reviews for phd candidates and other early-career researchers. *Higher Education Research & Development*, 33(3):534–548, 2014.

[147] Christophe Ponsard and Emmanuel Dieul. From requirements models to formal specifications in B. In *ReMo2V*, volume 241 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. URL `http://dblp.uni-trier.de/db/conf/caise/remo2v2006.html\#PonsardD06`.

[148] P M S Poon, Tharam S Dillon, and E Chang. Transformation of QoS data into XML characterising data communication in real time distributed systems. In *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*, pages 204–209, June 2004. doi: 10.1109/INDIN.2004.1417330.

[149] Eltjo R. Poort, Nick Martens, Inge van de Weerd, and Hans van Vliet. *How Architects See Non-Functional Requirements: Beware of Modifiability*, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-28714-5. doi: 10.1007/978-3-642-28714-5_4.

[150] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991. ISBN 0-13-478702-1.

[151] Shengchao Qin and Guanhua He. Linking Object-Z with Spec#. In *ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems*, pages 185–196, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2895-3. doi: http://dx.doi.org/10.1109/ICECCS.2007.27.

[152] Prasad Rajagopal, Roger Lee, Thomas Ahlswede, Chia-Chu Chiang, and Dale Karolak. A new approach to software requirements elicitation. In *Proceedings of the Sixth*

International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks, SNPD-SAWN '05, pages 32–42, 2005. ISBN 0-7695-2294-7.

[153] Venkataraman Ramesh, Robert L Glass, and Iris Vessey. Research in computer science: an empirical study. *Journal of systems and software*, 70(1):165–176, 2004.

[154] Gil Regev and Alain Wegmann. Where do goals come from: the underlying principles of goal-oriented requirements engineering. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 253–362, 2005. ISBN 0-7695-2425-7.

[155] C Rohleder. Representing Non-functional Requirements on Services - A Case Study. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pages 359–366, July 2012. doi: 10.1109/COMPSACW.2012.71.

[156] N S Rosa, P R F Cunha, and G R R Justo. ProcessNFL: a language for describing non-functional properties. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3676–3685, January 2002. doi: 10.1109/HICSS.2002.994496.

[157] Nelson Souto Rosa and Paulo Roberto Freire Cunha. An approach for reasoning and refining non-functional requirements. *J. Braz. Comp. Soc.*, 10(1):59–81, 2004. URL `http://dblp.uni-trier.de/db/journals/jbcs/jbcs10.html#RosaC04`.

[158] Simone Röttger and Steffen Zschaler. Tool Support for Refinement of Non-functional Specifications. *Software & Systems Modeling*, (2):185–204. ISSN 1619-1366. doi: 10.1007/s10270-006-0024-x.

[159] Jean-François Roy, Jason Kealey, and Daniel Amyot. Towards Integrated Tool Support for the User Requirements Notation. In Reinhard Gotzhein and Rick Reed, editors, *SAM*, volume 4320 of *Lecture Notes in Computer Science*, pages 198–215. Springer, 2006. ISBN 3-540-68371-2.

[160] Omar Salman. Animation of Z specifications by translation to Prolog. *Doğuş Üniversitesi Dergisi*, 1(1):155–167, 2011.

[161] Pere P Sancho, Carlos Juiz, Ramon Puigjaner, Lawrence Chung, and Nary Subramanian. An Approach to Ontology-aided Performance Engineering Through NFR Framework. In *Proceedings of the 6th International Workshop on Software and Performance*, WOSP '07, pages 125–128, New York, NY, USA. ACM. ISBN 1-59593-297-6. doi: 10.1145/1216993.1217014.

[162] A. M. Sen and K. Hemachandran. Goal Oriented Requirements Engineering: A Literature Survey. *Assam University Journal of Science & Technology: Physical Sciences and Technology*, 6(II):16–25, 2010.

[163] Graeme Smith. *The Object-Z specification language*. Kluwer Academic, Boston, 2000. ISBN 0792386841.

[164] Graeme Smith. State-Based Formal Methods for Distributed Processing: From Z to Object-Z. Technical report, Software Verification Research Center, The University of Queensland Australia, 2001.

[165] Graeme Smith and John Derrick. Abstract specification in Object-Z and CSP. In *Formal Methods and Software Engineering, volume 2495 of LNCS*, pages 108–119. Springer, 2002.

[166] Graeme Smith, Florian Kammller, and Thomas Santen. Encoding Object-Z in Isabelle/HOL. In *International Conference of Z and B Users (ZB 2002), volume 2272 of LNCS*, pages 82–99. Springer-Verlag, 2002.

[167] Colin Snook and Michael Butler. Using a graphical design tool for formal specification. In *Proceedings of the 13th Annual Workshop of the Psychology of Programming Interest Group*, pages 311–321, 2001.

[168] Colin Snook and Michael Butler. UML-B and Event-B: an integration of languages and tools. In *The IASTED International Conference on Software Engineering - SE2008*, February 2008. URL `http://eprints.soton.ac.uk/264926/`.

[169] Ian Sommerville. *Software Engineering*. Addison-Wesley, 8th edition, 2007.

[170] Jim Mike Spivey. *The Z notation: a reference manual*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1992. ISBN 0-13-978529-9.

[171] J. Sun, P. Loucopoulos, and L. Zhao. Representing and elaborating quality requirements: The QRA approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8217 LNCS:446–453, 2013. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84894198355`.

[172] Sam Supakkul and Lawrence Chung. A UML profile for goal-oriented and use case-driven representation of NFRs and FRs. In *Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on*, pages 112–119, August 2005. doi: 10.1109/SERA.2005.19.

[173] Fathi Taibi, Jacob K. Daniel, and Fouad Mohammed Abbou. On checking the consistency of object-z classes. *SIGSOFT Software Engineering Notes*, 32, July 2007. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/1281421.1281433.

[174] Chris Taylor, John Derrick, and Eerke Boiten. A Case Study in Partial Specification: Consistency and Refinement for Object-Z. pages 177–185, September 2000.

[175] Ian Toyn and John A. Mcdermid. CADiZ: An Architecture for Z Tools and its Implementation. *Software - Practice and Experience*, 25:305–330, 1995.

[176] John Andrew Van der Poll. *Automated support for set-theoretic specifications*. PhD thesis, University of South Africa: School of Computing (South Africa), 2001. Promoter-Kotze, P. and Promoter-Labuschagne, W. A.

[177] John Andrew van der Poll and Paula Kotzé. Enhancing the Established Strategy for Constructing a Z Specification. *SACJ*, (No. 35):118–131, 2005.

[178] John Andrew van der Poll, Kotzé Paula, Ahmed Seffah, Thiruvengadam Radhakrishnan, and Asmaa Alsumait. Combining UCMs and Formal Methods for Representing and Checking the Validity of Scenarios as User Requirements. *SAICSIT'03*, pages 111–113, 2003.

[179] Axel van Lamsweerde. Formal specification: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 147–159, 2000. ISBN 1-58113-253-0.

[180] Axel van Lamsweerde. Requirements Engineering: from craft to discipline. In Mary Jean Harrold and Gail C. Murphy, editors, *SIGSOFT FSE*, pages 238–249. ACM, 2008. ISBN 978-1-59593-995-1.

[181] F. Vernadat. Enterprise Modeling and Integration (EMI): Current status and research perspectives. *Annual Reviews in Control*, (1):15–25. ISSN 13675788. doi: 10.1016/S1367-5788(02)80006-2.

[182] Andrws Vieira, Pedro Faustini, Luigi Carro, and Érika Cota. NFRs Early Estimation Through Software Metrics. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 329–332, San Jose, CA, USA, 2015. EDA Consortium. ISBN 978-3-9815370-4-8. URL `http://dl.acm.org/citation.cfm?id=2755753.2755827`.

[183] Jaya Vijayan and G. Raju. Requirements elicitation using paper prototype. In Taihoon Kim, Haeng-Kon Kim, MuhammadKhurram Khan, Akingbehin Kiumi, Wai-chi

Fang, and Dominik lzak, editors, *Advances in Software Engineering*, volume 117 of *Communications in Computer and Information Science*, pages 30–37. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-17577-0. doi: 10.1007/978-3-642-17578-7_4.

[184] H Wada, J Suzuki, and K Oba. A Feature Modeling Support for Non-Functional Constraints in Service Oriented Architecture. In *Services Computing, 2007. SCC 2007. IEEE International Conference on*, pages 187–195, jul 2007. doi: 10.1109/SCC.2007.5.

[185] Hai H. Wang, Terry Payne, Nick Gibbins, and Ahmed Saleh. Formal Specification of OWL-S with Object-Z: The Dynamic Aspect. In Boualem Benatallah, Fabio Casati, Dimitrios Georgakopoulos, Claudio Bartolini, Wasim Sadiq, and Claude Godart, editors, *Web Information Systems Engineering WISE 2007*, volume 4831 of *Lecture Notes in Computer Science*, pages 237–248. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-76992-7. doi: 10.1007/978-3-540-76993-4_20.

[186] Alan Wassyng and Mark Lawford. *FME 2003: Formal Methods: International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003. Proceedings*, chapter Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project, pages 133–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-45236-2. doi: 10.1007/978-3-540-45236-2_9.

[187] Bo Wei, Zhi Jin, and Lin Liu. A Formalism for Extending the NFR Framework to Support the Composition of the Goal Trees. *2010 Asia Pacific Software Engineering Conference*, pages 23–32, November . doi: 10.1109/APSEC.2010.13.

[188] M. M. West and B. M. Eaglestone. Software development: two approaches to animation of z specifications using prolog. *Software Engineering Journal*, 7(4):264–276, July 1992. ISSN 0268-6961. doi: 10.1049/sej.1992.0027.

[189] Margaret Mary West. *Issues in Validation and Executability of Formal Specifications in the Z Notation*. PhD thesis, University of Leeds: School of Computing (UK), 2002. URL `http://etheses.whiterose.ac.uk/id/eprint/1305`.

[190] Margaret Mary West. Correctness Criteria for the Animation of Z Specifications via a Logic Programming Language. Technical report, University of Huddersfield, Queensway, HD1 4DH, UK, 2007.

[191] Roel Wieringa and Eric Dubois. Integrating semi-formal and formal software specification techniques. *Information Systems*, 23(3-4):159–178, 1998. URL `http://doc.utwente.nl/61832/`.

[192] Kirsten Winter and Roger Duke. Model Checking Object-Z Using ASM. In *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, pages 165–184, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43703-7. URL `http://dl.acm.org/citation.cfm?id=647983.743685`.

[193] Jim Woodcock. Calculating Properties of Z specifications. *SIGSOFT Softw. Eng. Notes*, 14(5):43–54, 1989. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/71633.71634.

[194] Jim Woodcock and Jim Davis. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-948472-8.

[195] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, October 2009. ISSN 0360-0300.

[196] Lihua Xu, Hadar Ziv, Thomas A Alspaugh, and Debra J Richardson. An architectural pattern for non-functional dependability requirements. *Journal of Systems and Software*, (10):1370–1378. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2006.02.061.

[197] Bin Yin and Zhi Jin. Extending the Problem Frames Approach for Capturing Non-functional Requirements. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 432–437. IEEE, May . ISBN 978-1-4673-1536-4. doi: 10.1109/ICIS.2012.47.

[198] Lin Yu and Eric Liu. From requirements to architectural design–using goals and scenarios. In *First International Workshop From Software Requirements to Architectures-STRAW*, volume 1, page 22, 2001.

[199] Liming Zhu and I Gorton. UML Profiles for Design Decisions and Non-Functional Requirements. In *Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent, 2007. SHARK/ADI '07: ICSE Workshops 2007. Second Workshop on*, page 8, May 2007. doi: 10.1109/SHARK-ADI.2007.14.

[200] Didar Zowghi and Chad Coulin. *Requirements Elicitation: A Survey of Techniques, Approaches, and Tools.*, pages 19–46. Springer Berlin Heidelberg, 2005. doi: 10.1007/3-540-28244-0_2.

Index