UML Profile for Goal-oriented Modelling

Muhammad Rizwan Abid

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science

University of Ottawa
Ottawa, Ontario, Canada
August 2008

Abstract

The Unified Modeling Language (UML) is a standard for visual modelling. We can design abstract models by using its elements. Although the semantic scope of UML elements is very broad, it does not fully address the needs of some modelling domains, including the modelling of goals and non-functional requirements (NFR). To address this problem, UML allows the customization of its metamodel with an extension mechanism called *UML profile*.

Some work has already been done in the area of UML profiles for NFR and goals. In some cases, the proposed solutions were incorrectly or only partially integrated with UML. Sometimes, the profiles were based on metamodels whose nature and accuracy for this domain were unclear. In other cases, the profiling approaches taken were not well supported by tools, which have led to unsatisfactory solutions.

In this thesis, we propose a UML profile for the Goal-oriented Requirement Language (GRL), a goal/NFR notation undergoing standardization at the International Telecommunication Union. Our profile is based on an abstract metamodel of GRL, which has already been successfully tested and implemented in non-UML tools. This profile is also implemented in a UML 2 tool, namely Telelogic G2 4.0, and is well integrated with the rest of UML. Challenges and design decisions for the concrete support of this profile with tools are discussed along the way. The profiling approach used in this thesis is one that has been recommended by modellers and standards developers. Our profile for goal-oriented modelling is also illustrated and validated with several examples.

Acknowledgment

I am first of all thankful to Allah, who has provided me this opportunity of doing a Master of Computer Science degree at the University of Ottawa.

I would like to express my gratitude and special thanks to my supervisors, Dr. Daniel Amyot and Dr. Stéphane Sotèg Somé, who have supported me with invaluable suggestions, knowledge and positive encouragements during my thesis work.

I also wish to offer my special thanks to Gunter Mussbacher for our great discussions and his comments that provided me with inspiration to complete this thesis.

I am also thankful to Telelogic (now IBM), who provided the Tau G2 4.0 tool and technical support for my research work, as well as NSERC for financial support.

Finally, I want to thank my parents, family members, and especially my wife for her understanding throughout my studies and my mother who supported and encouraged me from past to the present.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
BNF	Backus-Naur Form
BWW	Bunge-Wand-Weber ontology
CIM	Computation Independent Model
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
DSM	Domain Specific Modelling
EAI	Enterprise Application Integration
EDOC	Enterprise Distributed Object Computing
EMF	Eclipse Modeling Framework
FR	Functional Requirements
GEF	Graphical Editing Framework
GME	Generic Modeling Environment
GRL	Goal-oriented Requirement Language
GUID	Globally Unique Identifiers
IBM	International Business Machines Corporation
IS	Infrastructure
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
jUCMNav	Java Use Case Maps Navigator
LEL	Language Extended Lexicon
MARTE	Modeling and Analysis of Real-time and Embedded System
MDA	Model Driven Architecture
MDD	Model Driven Development
MEM	Metamodel Extension Mechanism
MOD	Module
MOF	Meta Object Facility
MSC	Message Sequence Chart
NFR	Non-Functional Requirements
OCL	Object Constraint Language
OMG	Object Management Group
POP*	Process, Organisation, Product, and so on
RSA	Rational Software Architecture
SDL	Specification Description Language
SM	Stereotype Mechanism
SIG	Softgoal Interdependency Graph
SS	Superstructure
SysML	System Modeling Language

TCL	Tool Command Language
TTCN	Testing and Test Control Notation
UCM	Use Case Map
UEML	Unified Enterprise Modeling Language
UML	Unified Modeling Language
UML IS	Unified Modeling Language- Infrastructure
UML SS	Unified Modeling Language- Superstructure
URN	User Requirements Notation
URN-FR	User Requirements Notation – Functional Requirements
URN-NFR	User Requirements Notation – Non-Functional Requirements
U2TP	UML Testing Profile
WSDL	Web Service Definition Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Chapter 1. Introduction

1.1. Context and Concepts

The modelling of goals and Non-Functional Requirements has always been a hot topic of discussion in the field of analysis and design. *Goals* are high-level objectives or concerns of a business, stakeholders, or system. They are often used to discover, select, evaluate, and justify requirements for a system. *Functional Requirements* (FR) define functions of the system under development, whereas *Non-Functional Requirements* (NFR) characterize system properties and qualities, such as expected performance, robustness, usability, cost, etc. Goals and NFRs capture essential aspects of systems, which have a significant impact throughout the development process.

The Unified Modeling Language (UML) is the most popular modelling language for software applications. However, many modellers are still unsatisfied with the role of UML in the area of goal and NFR modelling. With the importance of UML in the industry, this deficiency has now become an apparent weakness.

Modellers struggle to define how best to describe and structure goals. While some metamodels[1] for goal modelling exist, such work has often been completed in isolation and has not been done in accordance with standards. Yet, there exists at least one mature metamodel for goal modelling that is undergoing standardization [17]. However, having a standardized goal metamodel is not sufficient, as we still require it to be aligned with the UML metamodel. Such alignment will reduce existing communication and integration problems between goal modellers and UML modellers.

UML does not allow for a direct modification of its metamodel per se. However, there is a generic extension mechanism for tailoring UML to a particular domain. This mechanism is called *UML profiling*. A comprehensible UML profile for goal modelling would represent a mechanism that enables the integration of goals with the rest of UML

[1] A *metamodel* is a model used to describe modeling languages. It defines the modeling concepts, their attributes, and their relationships. Often, metamodels are represented as UML class diagrams.

and establishes directions for developers to assist them in resolving modelling issues in this domain. Such a profile would also help tool vendors to synchronize on one metamodel instead of providing their own solutions in the form of different customized metamodels. In this thesis, a UML profile is created for goal modelling. It aims at improving the ease with which modellers integrate goals with other UML concepts.

In this chapter, the motivation behind this work will be discussed as well as the thesis hypothesis. Finally, there will be a summary of the main contributions and an overview of the other chapters.

1.2. Motivation

The Unified Modeling Language does not address explicitly the modelling of goals and non-functional requirements. In the software engineering community, this deficiency has now become an issue. There has been some work in the area of UML profiles for goal modelling [5][6][9][25], but the solutions proposed suffer from many deficiencies.

Through the study of the related work above, it has come to light that there are some minimal requirements that are considerably important for UML profiles for goal-oriented modelling. These are 1) the integration with UML (i.e. the ability of sharing information between the goal diagram elements and the existing UML elements), 2) diagram pollution avoidance (i.e. preventing the mixing of different diagram constructs), 3) metamodel stability (i.e. the maturity of the underlying goal metamodel) and 4) implementability of the profiling mechanism (i.e. how well the approach used for the creation of a profile is amenable to implementation). These requirements are discussed further in section 2.7.1.

The need to address deficiencies related to these requirements, which are common among current solutions, has motivated the development of a new UML profile for the Goal-oriented Requirement Language (GRL).

1.3. Research Hypothesis

The research hypothesis of this thesis is that UML can be profiled to support goal-oriented modelling with a semantics rooted in a standard metamodel such as that of the

Goal-oriented Requirement Language. The assumption is that a profile based on a mature and a stable metamodel that has been already used by editors and in analysis techniques represents a better alternative to current solutions and that this will ease the support by UML tools, with which resulting models will combine goal-oriented concepts with object-oriented concepts in a way that is comprehensible by the UML community at large.

1.4. Thesis Contributions

The main contributions of this thesis include:

- The creation of a UML profile for GRL, where UML metaclasses are mapped in detail to GRL's metaclasses. Standard guidelines [15] have been followed while defining this profile.
- A proof of concept implementation, which demonstrates the feasibility of supporting such profile in a commercial tool, namely Telelogic Tau G2 4.0.
- Illustration of typical usage of this profile with examples where GRL is used standalone in a model, and then where GRL diagrams are combined with selected UML diagrams in a model.

1.5. Thesis Outline

The rest of this thesis is as follows: Chapter 2 presents background work on UML and GRL and on the standards, profiling technologies, and the tools required for understanding this thesis. An evaluation of related work against our minimal requirements is also included. Chapter 3 is the core of the thesis and defines the UML profile for GRL by providing a detailed mapping of UML metamodel elements to GRL metamodel elements, with corresponding semantics, required attributes, graphical representation, and constraints. Chapter 4 focuses on the implementation of this profile in a commercial UML 2 tool, with a discussion of the challenges faced along the way. Chapter 5 illustrates and validates the profile by creating goal-oriented models with the tool, some of which being compared to similar models created with a non-UML GRL tool (jUCMNav). Chapter 6 summarizes our contributions, provides conclusions and proposes areas for future work.

Chapter 2. Background

The use of UML is very common amongst software engineers, modellers and designers. In most cases, these people take advantage of common types of UML diagrams such as class diagrams and sequence diagrams. In this thesis, one of our main concerns is with the architecture and semantics of UML itself and not only its diagrams. It is important to understand basic concepts of the Unified Modeling Language Superstructure (UML SS) and Unified Modeling Language Infrastructure (UML IS), especially those related to profiles. Section 2.1 provides an overview of UML in this context. Section 2.2 introduces the User Requirements Notation (URN) and its two components, the Goal-oriented Requirement Language (GRL) and the Use Case Map (UCM) notation. Section 2.3 presents details of GRL constructs, their meaning, and how they interlink with each other. Section 2.4 discusses the GRL metamodel, as this will be the target of the profile proposed in this thesis. Section 2.5 provides background information on UML profiling and on tools used for creating UML profiles. In section 2.6, standard guidelines from the International Telecommunication Union for the creation of profiles are presented. This work will adhere to these standards. Section 2.7 provides an overview of existing UML profiles for goal modelling and their limitations.

2.1. Unified Modeling Language (UML)

According to the Object Management Group 2007 (OMG), modelling is the designing of software applications before coding. The only way to attain a complete picture of a system is through a model. A model can help to reveal the extent to which requirements exactly match a system. UML is an OMG standard language which is rich in graphical notations. These notations can be used by modellers to create an abstract model of their system. The latest version of UML (2.1.2, see [22][23]) supports 13 types of diagrams, which are displayed in Figure 1.

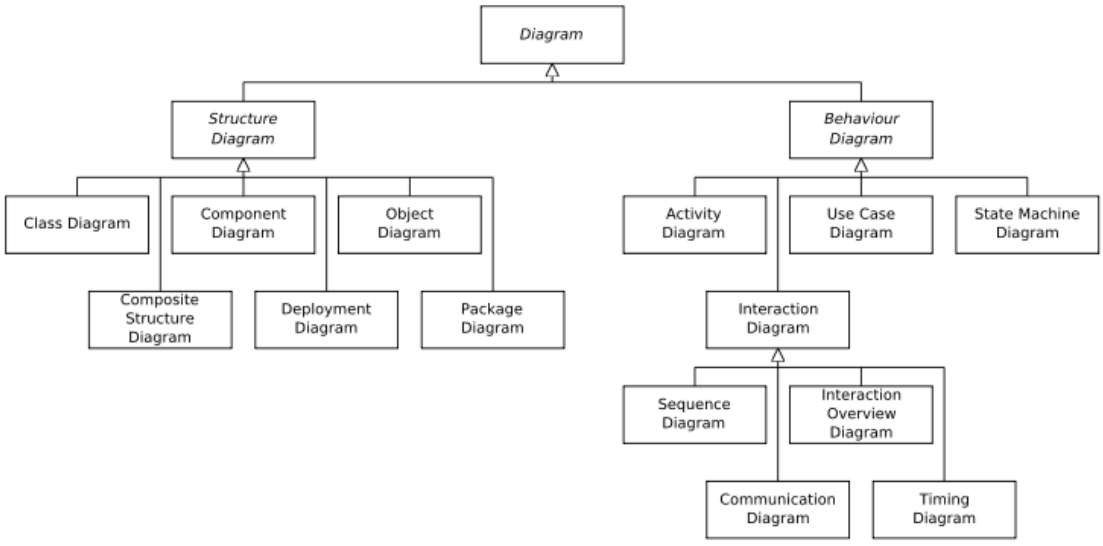

Figure 1 UML Diagrams Presentation in Class Diagram (UML IS)

The UML specification is divided into an Infrastructure (IS) and a Superstructure (SS) documents. These two specifications provide a complete picture of UML and are briefly reviewed below in section 2.1.1 and 2.1.2.

2.1.1 UML Infrastructure

The UML Infrastructure [22] is the component of the UML specification that includes all the constructs that comprise the foundation of this modelling language. The UML has a significant scope, so its constructs and modelling concepts are grouped into different language units. A *language unit* is a group of modelling concepts and constructs that provide users with the power to represent aspects of the system under study, according to a particular paradigm or formalism. The UML Infrastructure is architecturally aligned with the UML Superstructure. This specification is described using a metamodelling approach. The UML infrastructure is based on the *InfrastructureLibrary* package which makes the UML, Meta Object Facility (MOF) [21] and XML Metadata Interchange (XMI) architecturally aligned so that model interchange is fully supported [22]. The InfrastructureLibrary is used to customize the UML metamodel by profiling and creating new languages based on the same metalanguage core as UML. There are two packages in the InfrastructureLibrary (Core and Profiles). The Core package has four other packages dependant on

it. As shown in Figure 2, these include UML, CWM (Common Warehouse Metamodel), Profiles, and MOF.

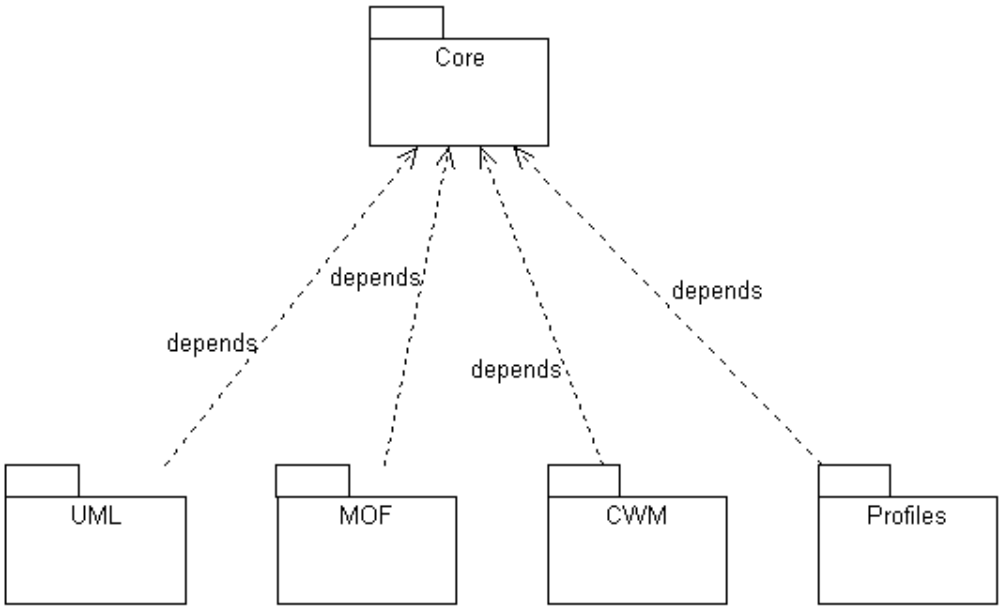

Figure 2 Core Package And its Dependents

The Core package is subdivided into four sub-packages: PrimitiveTypes, Abstractions, Basic, and Constructs. These packages are also subdivided into further sub-packages. It is however not essential to understand the entire infrastructure for this thesis. The Profiles package is neither helpful nor practical when used alone; its functionality is dependent on the Core package and it can be used in combination with either the Core package or the UML package. In the UML infrastructure, the MOF and UML packages are not categorized at the same metalevel.

Metamodel and Layers

A metamodel can be instantiated into a model. These layers are relative to each other and can be used repeatedly. Figure 3, inspired by a figure in the UML infrastructure specification [22], shows a common example that uses four types of layers:

- **Meta-metamodel (M3)**: This is the topmost level; the MOF is one example. The M3 level model has very broad scope and defines concepts that can be used to de-

fine all metamodels. The main characteristic of an M3 level language is that it can define itself by its own constructs and rules, in a circular way (hence, there is no need for a different M4 level). For instance, MOF can define itself.

- **Metamodel (M2)**: This is an instance of M3, meaning that each of its constructs is an instance of an M3 level construct. M2 metamodels define the concepts and rules of a modeling language. An example of a metamodel is the UML metamodel, which is an instance of MOF. Note that concepts defined in a layer above can also be reused (or inherited) in a layer below. For instance, UML inherits part of MOF as well.
- **User-specified model (M1)**: This is an instance of M2. The language of M2 is used to create a model of a system at M1. An example of an M1 model is a student registration system in a university.
- **Object (M0)**: This is a runtime instance of M1, with concrete values for the attributes of a user-specified model. Example of M0 objects would include specific students and universities, with their names.

According to the example of Figure 3, the meta-metaclass *Class* is defined at the M3 level. *Attribute, Class* and *Instance* are M2 level concepts. Note that *Class* of M3 is from MOF, while *Class* of M2 is from UML. *Student* and *University* are both at M1 level (i.e. the user model) and, finally, *Saeed, Martin* and *UOttawa* are all at the M0 level, which is a runtime instance of M1.

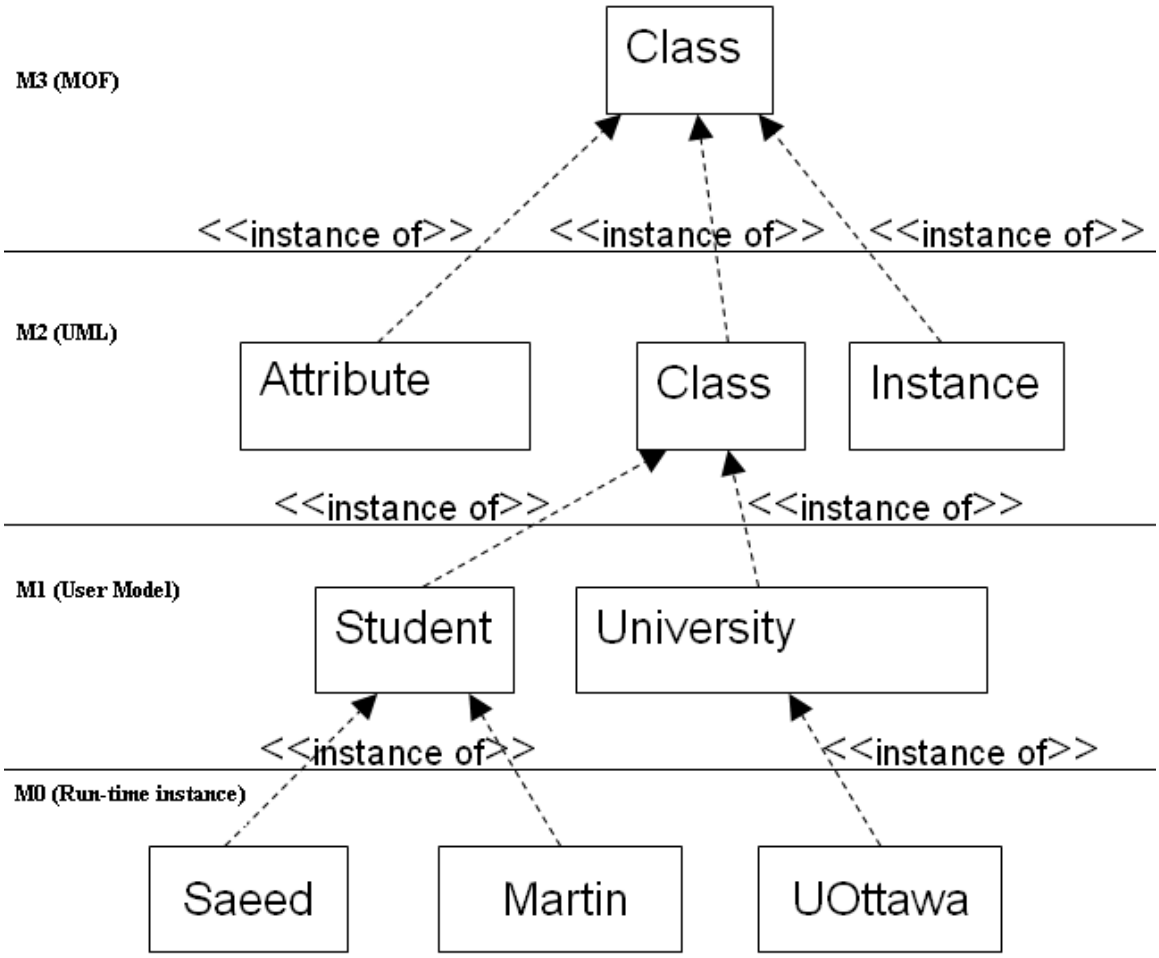

Figure 3 Four Layers of Metamodel

2.1.2 UML Superstructure

The UML Superstructure [23] complements the UML Infrastructure and divides the entire UML into four levels, according to an increasing level of complexity for the modelling concepts. Level 0 is simply an empty package that merges the contents of the Basic package from the UML Infrastructure (Figure 4). Level 0 has elementary concepts that allow it to function as a basic, interoperable language between various groups of modeling tools. Level 1 inherits level 0's components and contains additional features, which include UseCases, Interactions, Structures, Actions and Activities (Figure 5). Level 2 inherits level 1's elements, and the former's additional features include Deployment, State

machine modelling and Profiles (Figure 6). Level 3 is comprised of all previous levels and contains the features that represent the complete UML, which include Modelling Information Flows, Templates and Modelling packaging (0). These levels can be used to establish a compliance level, as tools may support UML up to one of these levels only.

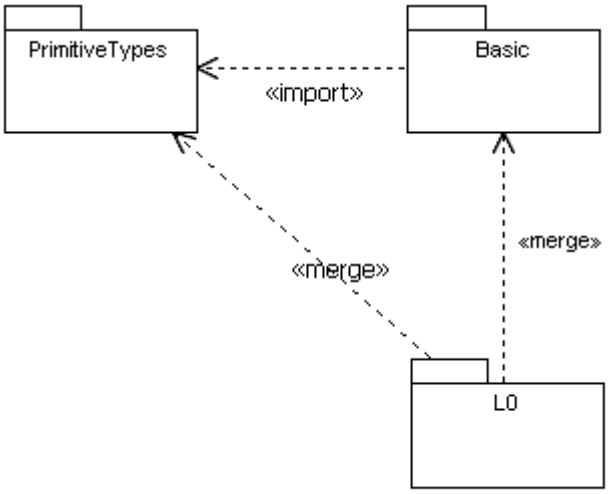

Figure 4 Level 0 (UML Superstructure)

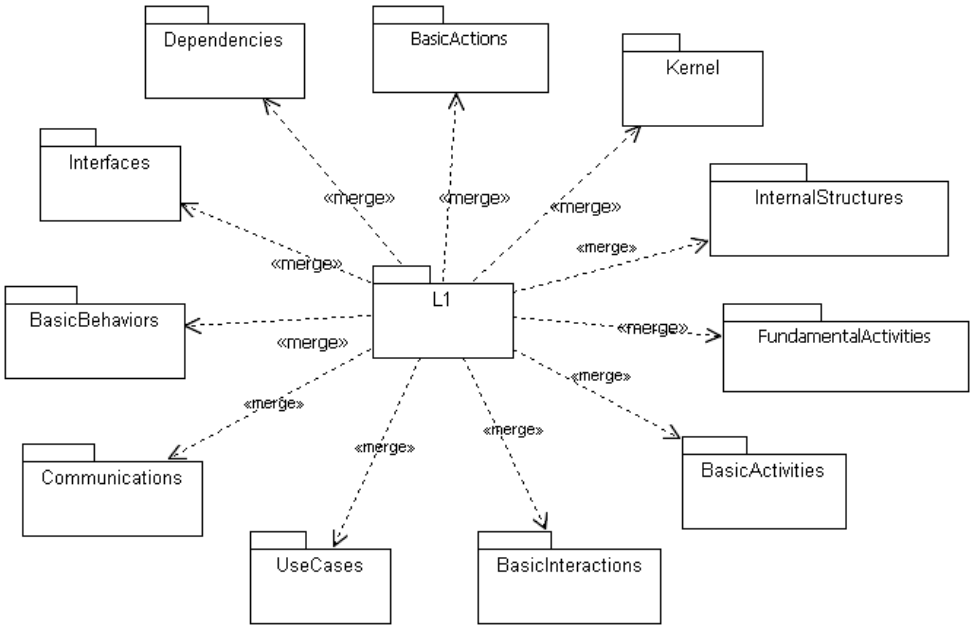

Figure 5 Level 1 (UML Superstructure)

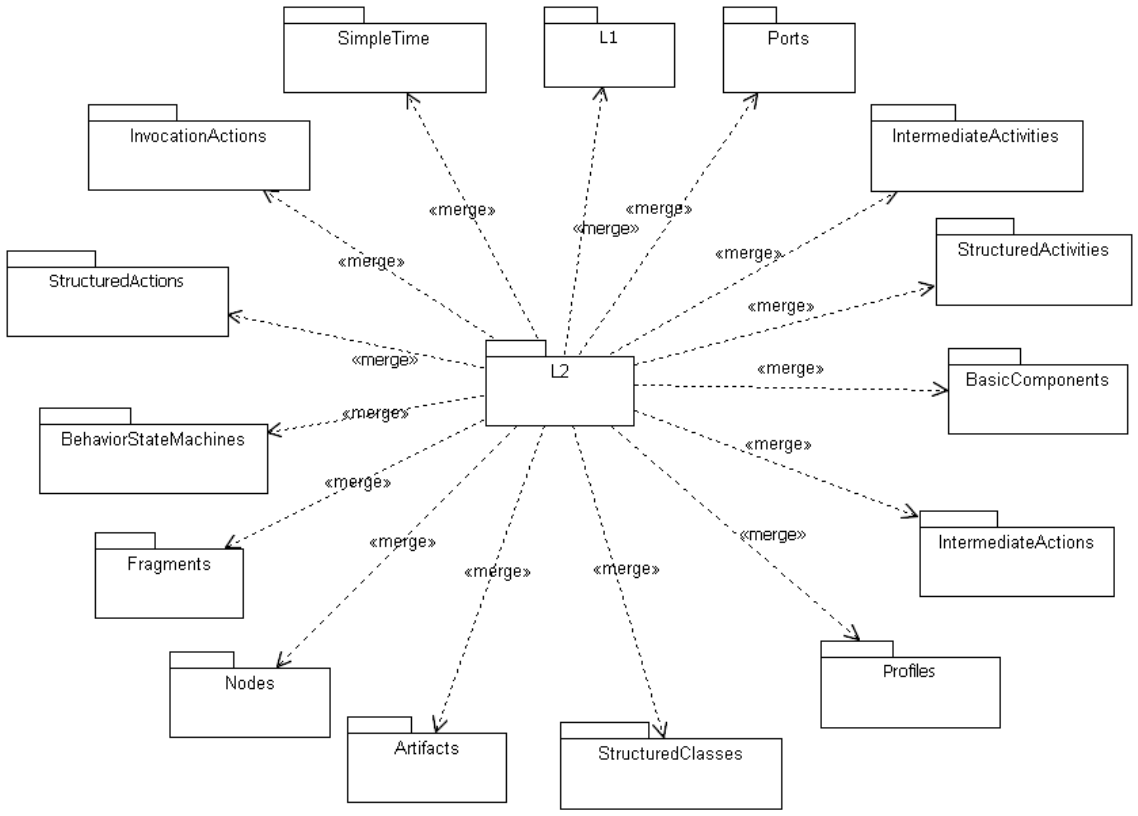

Figure 6 Level 2 (UML Superstructure)

The superstructure also contains the definition of the human-readable notation elements for representing the individual UML modeling concepts as well as rules for combining them into a variety of different diagram types corresponding to different aspects of models, i.e. the 13 types of UML diagrams described in Figure 1.

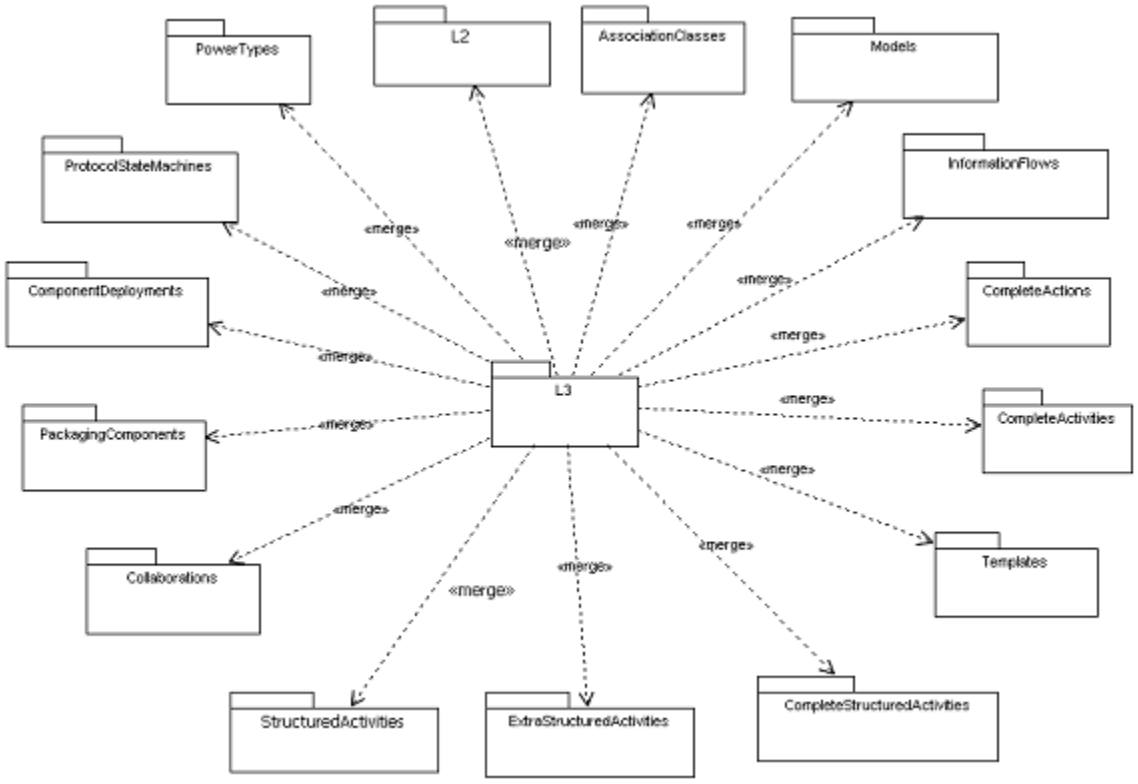

Figure 7 Level 3 (UML Superstructure)

2.2. User Requirements Notation (URN)

The User Requirements Notation (URN) [3][16][27] is being standardized by the International Telecommunication Union (ITU-T) as the Z.150 series of Recommendations. URN can be defined as a graphical modelling language that is used to gather user requirements during the very early stages of design. “It enables modellers to analyze scenarios, goals and NFRs, and also to apply traceability and transformations to other languages such as Message Sequence Charts. URN was initially developed to address the elicitation, modelling, analysis, and validation of functional and non-functional requirements for new or evolving reactive and distributed systems” [3], but it is applicable to a much wider range of application domains.

URN is composed of two complementary notations: the Goal-oriented Requirement Language (GRL), discussed in section 2.3 in detail, and the Use Case Map (UCM) scenario notation. Whereas GRL focuses on the “why” aspects of requirements model,

UCM target the “what” aspects. Draft Recommendation Z.151 [17] describes the details of the definitions, attributes, relationships, constraints, semantics and semantics variations of URN, GRL and UCM for both their abstract and concrete metamodels.

Figure 8 gives a high-level overview of some of the top-level metaclasses of the URN metamodel. In particular, a URN specification may contain GRL and/or UCM specifications, as well as links between their elements. Links and URN model elements can also contain metadata information.

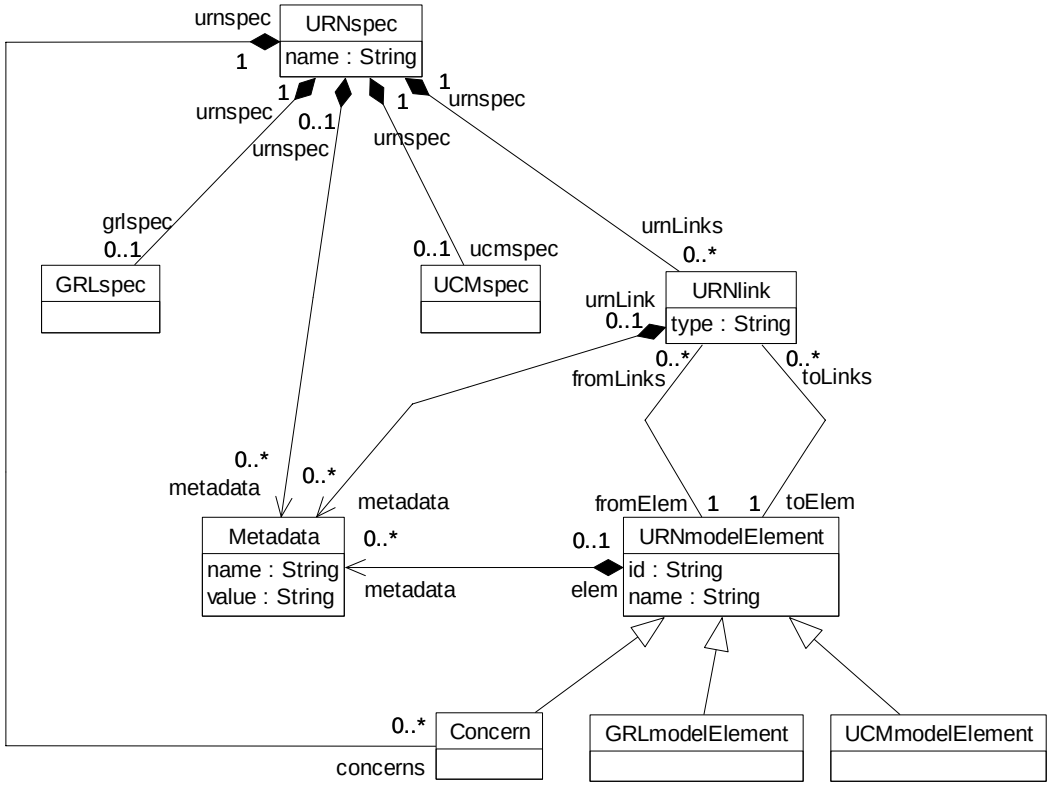

Figure 8 High-Level Overview of the URN Metamodel (Z.151)

The rules and metalanguages used to describe the URN language in the Z.151 document are based on another ITU-T standard, namely Z.111 (Notations to Define ITU-T languages) [14]. Recommendation Z.111 standardizes notations to create abstract and concrete syntaxes of languages, using metagrammars or metamodels.

“In the following a Named element is a meta-class that contains a name attribute of the meta-class Token.

i) *Class* is a Named element that contains one or more *Attribute* items and can participate in binary *Association* relations via one or both of the *AssociationEnd* attributes of each *Association*. A *Class* can inherit the *Attribute* and *AssociationEnd* (and hence *Association*) items from another *Class* (single inheritance only). Each *Class* shall have a *name* with a *Token* value that is distinct from the *Token* value for the *name* of any other *Class* (including predefined primitives), or any *Enumeration*. The *inheritedFrom* attribute (if present) of *Class* references the parent *Class* of a *Class*. An inherited *Class* of a *Class* is either the parent *Class* or any inherited *Class* of the parent *Class*, *transitively*.

ii) *Attribute* is a Named element. It is a meta-class for an ordered element of a *Class* that has a *DataType* and an optional *DefaultValue*. The multiplicity of an *Attribute* is given by *multiplicityMin* and *multiplicityMax*, which have default values of 1 if they are omitted in concrete notation.

iii) *DataType* is a meta-class that is either a *PrimitiveType* or an *EnumeratedType*. *PrimitiveType* is a *DataType* where the type is predefined (one of *Token*, *Nat*, *Boolean*, *Unspecified*) primitive identified by the *Primitive* value of the *name* attribute of the *PrimitiveType*. *EnumeratedType* is a *DataType* where the type is an *Enumeration* identified by *name* attribute.

iv) *Enumeration* is a Named element that represents a (quotation) *DataType* whose values are represented by a set of *literal* elements.

v) *Literal* is a Named element contained in an *Enumeration* that is one of the values of an *Enumeration*.

vi) *DefaultValue* is a meta-class for the optional component of an *Attribute* that identifies a default value of the appropriate type.

vii) *Association* is a meta-class for the relation between *Class* meta-classes. It has two *AssociationEnd* attributes (*source* and *target*) for the logical connection of the *Association* with the related *Class* meta-classes.*AssociationEnd* is a meta-class for the end of an *Association*. It has Boolean attributes that determine if the *AssociationEnd* is composite (*isComposite*), navigable (*isNavigable*) or

ordered (*isOrdered*). If the *AssociationEnd* is navigable (*isNavigable* = true), the *name* shall not be empty, and the *name* (which corresponds to the role name of the *AssociationEnd*) shall be distinct from the *name* of any other *AssociationEnd* that is an *end* of the *Class* that is the *source* of the *AssociationEnd*." [14]. Detail information is in the reference document.

The difference between Z.111 and Z.151 is that both standards exist at different metalevel layers. Z.111 uses an extended subset of Meta-Object Facility to define a simple meta-metamodel (M3) targeting the definition of modelling languages (see Figure 9), whereas Z.151 instantiates it to formalize URN's metamodel (M2) from which user-specified goal and scenario models can be instantiated (M1) and executed with concrete values (M0).

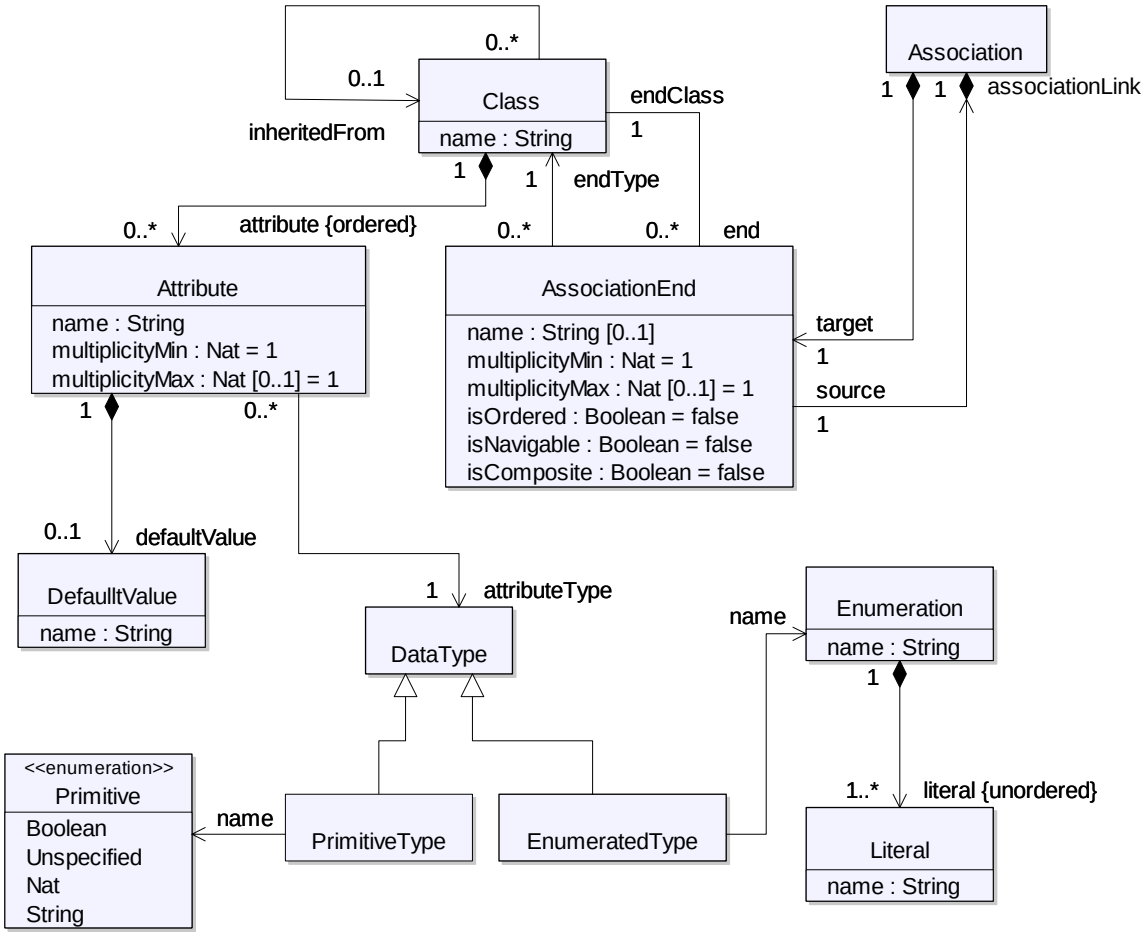

Figure 9 Z.111 Meta-metamodel

2.3. Goal-oriented Requirement Language (GRL)

The GRL [3][17] is a language that focuses primarily on goal modelling. A subset of URN and a graphical language, GRL's major contribution is to provide reasoning for non-functional requirements (high-level goals). GRL also works for functional requirements and has notations for both. Its focus is to design the *why* and the *what* aspects of a model. GRL integrates elements from two other goal-oriented modelling languages, *i** [30] and the NFR framework [4]. The *i** contains modelling elements that support goal, agent, and organization modelling, while the NFR framework provides an evaluation mechanism for goal models. GRL inherits both of these features and has become a very effective modelling language that also has the ability to evaluate any goal model.

GRL divides its modelling elements into three main categories: Intentional Elements, Actor and Links

Intentional Elements

Intentional elements are the constructs which are used to model the system. They carry the intentions of the stakeholders involved. There are five types of intentional elements: Softgoal, Goal, Task, Resource and Belief.

Softgoals are used to model high-level goals that are uncertain or that can never be fully satisfied. They are used to describe qualities and non-functional aspects such as security, robustness, performances and usability [17]. *Goals* are used to describe functional requirements, which are measureable and achievable. *Tasks* describe how goals can be achieved, how certain actions can be performed and how softgoals can be partially satisfied. A task is an action that an actor would ideally want performed in order to determine solutions for both goals and softgoals. *Resources* represent physical or informational entities. A resource status is either available or not available. *Beliefs* are used to capture rationales and are used by modellers to understand their concepts and claims for their described goals.

Actors

An actor represents a stakeholder or a system. It may contain sub-actors and intentional elements. An Actor is an entity that performs actions and makes decisions in order to accomplish goals, perform tasks and satisfy softgoals.

Links

Links are used to connect intentional elements and actors to show their inter-relationships. Different links have different semantics and behaviour. GRL has three types of links which are: Decomposition, Contribution and Dependency

Decomposition links are used to decompose an intentional element into its sub-intentional elements, one or all of which should be achieved or satisfied (depending on the decomposition type) in order for the target intentional element to be satisfied. One exception is that a belief cannot be decomposed either as a source or a target. There are three types of decomposition: *AND*, *XOR (exclusive OR)* and *IOR (inclusive OR)*. One intentional element can use only one type of decomposition at a time. Satisfaction levels can be quantitative or qualitative.

Contributions are links used to describe the positive or negative impact of one or more source intentional elements to their target intentional element. Such links can be used to understand which intentional element is contributing what to another intentional element. This contribution can be quantitative (numeric value) or qualitative (one of Make, Help, SomePositive, Unknown, SomeNegative, Hurt, and Break). *Correlations* are similar to contributions, but they are used for modelling side effects.

Dependency is another type of link indicating that an actor or intentional element depends on another actor or intentional element. All GRL notations are displayed in Figure 10.

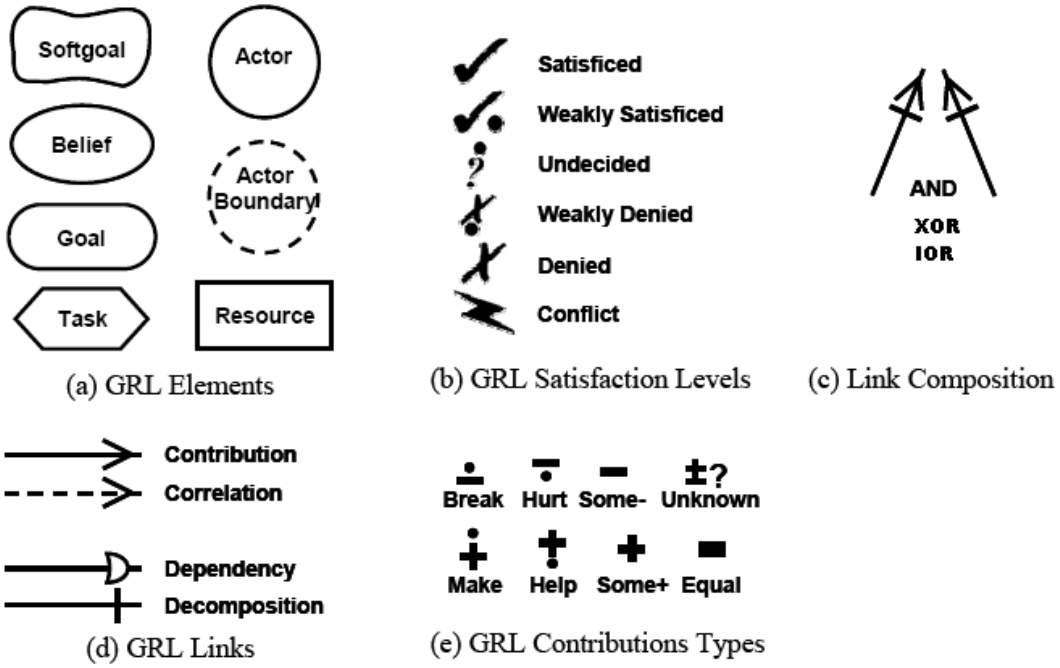

Figure 10 Summary of GRL Notations

Figure 11 is a sample model that illustrates the use of GRL constructs, including the combination of links, actor and intentional elements with different satisfaction values. This diagram is a representation of a teaching assistant (TA) candidate from a TA selection model. TA candidates expect the system to be secure and flexible. According to them, the system should allow one to apply to many courses at a time in a secure way. They do not want to enter the same information repeatedly and also do not want to submit forms by physical means at a particular office. Figure 11 shows that TA candidates can apply for positions through several means: paper-based applications or web-based applications. Web-based applications can be done securely but their availability might be limited, as not all candidates have computers.

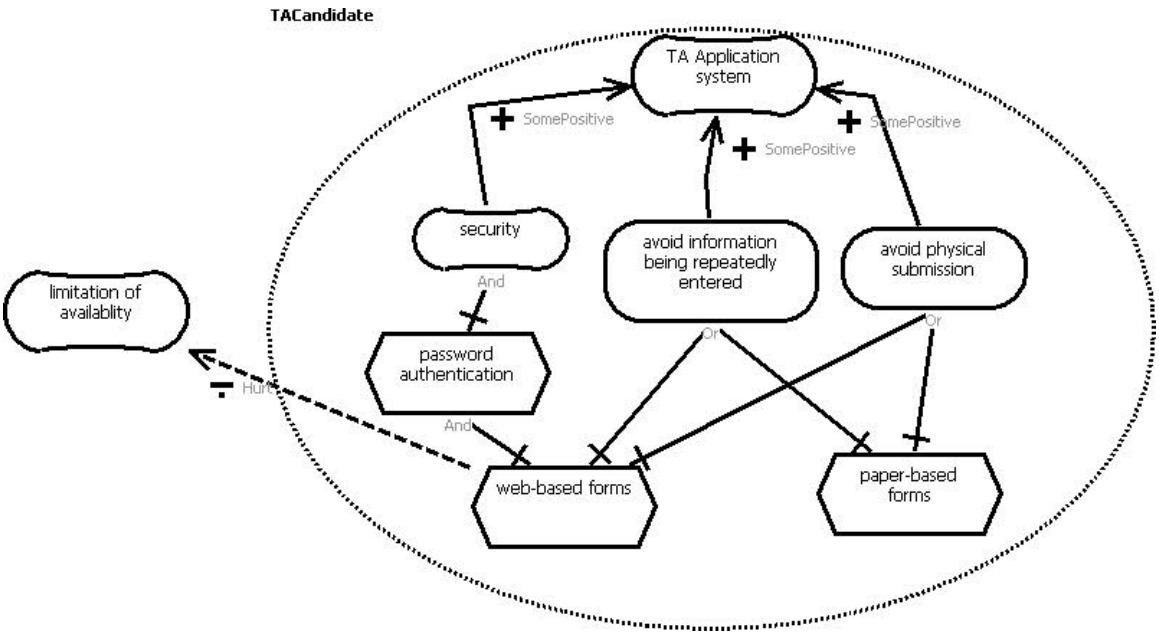

Figure 11 GRL Model Sample

GRL Strategies

GRL strategies are also a powerful feature of GRL. The process is initiated by a modeller, who assigns initial satisfaction values to some of the intentional elements (usually leaves in the GRL graph). These values then propagate in the GRL model through links, hence computing satisfaction levels for the other connected intentional elements. The strategy effect is shown by a coloring scheme applied to different intentional elements as demonstrated in Figure 12. This evaluation method is effective at an early stage of system modelling. It helps to analyze different alternatives and tradeoffs for the system and the stakeholders involved.

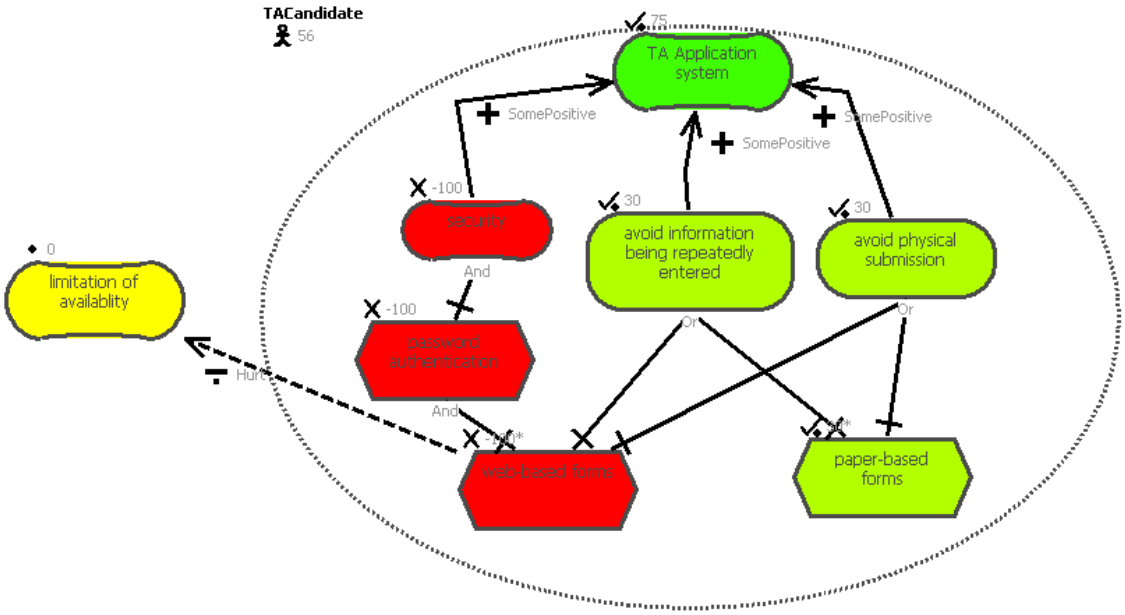

Figure 12 GRL Strategies

2.4. Metamodel of Goal-oriented Requirement Language (GRL)

A *model* is an abstraction of a system built with an intended purpose in mind. A model should be able to answer questions in place of the actual system [8]. A model that describes a set of related models is called a *metamodel*. A model is an instance of its metamodel. OMG's MOF is a standard meta-metamodel for model-driven engineering, with standard instances such as UML and CWM. Application modellers use UML diagrams in order to create their specific models. Z.111 is GRL's meta-metamodel, but the GRL metamodel itself is illustrated graphically with a UML class diagram.

An important part of this thesis examines which classes from the UML metamodel can be extended to map to the GRL metamodel classes, without violating inherited constraints.

2.4.1 Scope

The GRL metamodel provides an opportunity for all modellers (including software engineers and requirements engineers) to model requirements, especially non-functional requirements, for their proposed systems and to evaluate them by using strategies for preci-

sion, validity and comprehensiveness. In this thesis, we focus on the GRL models elements themselves, and leave aside the notion of strategy, as the latter relates more to the evaluation of the former.

2.4.2 Overview

The GRL core diagram in Figure 13 represents the GRL metamodel. It complements the URN metamodel extract shown in Figure 8.

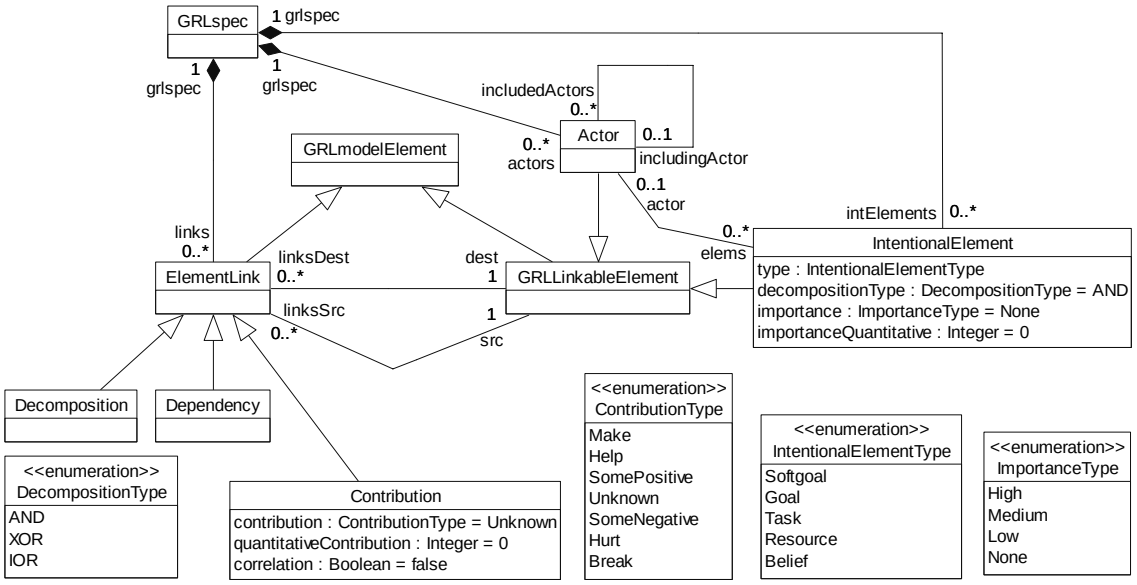

Figure 13 Metamodel of GRL [17]

This metamodel formalizes all the GRL concepts and constructs introduced earlier. Unlike what the name “abstract metamodel” may imply, there are no abstract classes in the abstract metamodel; there are only concrete classes. However, it is considered abstract because it does not include graphical or layout information relevant to the concrete notation. All of the constraints, semantics, attributes and notations are discussed in detail in Chapter 3. Here we only discuss how these classes interact with each other, and how such metamodel should be read.

The Actor class has a self-association “includingActor” of multiplicity [0..1] and “includedActors” of multiplicity [0..*]. This association specifies that Actor may or may not contain one or more sub-Actors and all included Actors have a unique including Ac-

tor. The Actor class has an association “elems” with IntentionalElement, with multiplicity [0..*], meaning that an Actor may or may not contain IntentionalElements. The GRLSpec contains the Actors, which are generalized by GRLLinkableElement.

The IntentionalElement class has an association with the Actor class, which means that an intentional element may belong to at most one actor. The IntentionalElements are also GRLLinkableElements and are contained by GRLSpec. Note that there are various types of IntentionalElement.

The ElementLink class, also contained by GRLspec class, is a generalization of Decomposition, Dependency and Contribution. The ElementLink class has two associations with GRLLinkableElement class, one describing the source and the other the target.

The DecompositionType, ContributionType, IntentionalElementType and ImportanceType are enumeration classes used to capture values for the attributes of some of the classes mentioned above.

GRLspec aggregates the IntentionalElements, Actors and ElementLinks (Figure 13) but is itself contained by URNspec, as shown in Figure 8.

2.5. UML Profile

2.5.1 Definition

A UML profile [18] is an extension mechanism provided by UML to customize it’s metamodel. UML does not allow direct modifications of its metamodel, but the use of a UML profile enables one to reuse and extend the UML metamodel constructs with new concepts. One common interpretation of a UML profile is “the extension mechanism of a UML metamodel for a particular domain” [23]. The different constructs used to define a profile are listed below with their definitions:

- **Stereotype**: A construct which is similar to a class construct in UML’s metamodel. According to the UML Superstructure Profile package, stereotypes are specific metaclasses.
- **Tagged value**: A construct similar to an attribute construct of a class in UML’s metamodel. According to the UML Superstructure Profile package, tagged values are standard meta-attributes.

- **Constraints**: Restrictions required in a particular domain.

2.5.2 The Purpose of UML Profiles

The purpose of introducing UML profiles is to give modellers the means to customize the UML metamodel without directly changing it. Modellers can utilize stereotypes, tag values and constraints for this customization. However, some restrictions still apply, as the modeller can introduce new constraints, but cannot violate any inherited constraint from the original UML metamodel. Being able to customize the UML metamodel through a UML profile is an effective method to improve and narrow the scope of the original UML metamodel for a specific domain.

In many cases, like in the finance or manufacturing domains, there is no need to create profiles because the original UML constructs already fulfill modelling requirements. The profile's most prominent use is when a new, domain-specific language is being introduced. In order for a community to define and integrate a new language, a profile can be created to extend the UML metamodel.

There already exist multiple standard UML profiles available on the OMG Web site [20], including: Systems Modeling Language (SysML); Common Object Request Broker Architecture (CORBA); Enterprise Application Integration (EAI); Enterprise Distributed Object Computing (EDOC); Profile for Modeling and Analysis of Real-time and Embedded System (MARTE); Schedulability, Performance and Time; Software Radio; Voice; and Testing.

2.5.3 UML Profile Creation

There are different ways of creating a UML profile. These different ways are [18]:

Stereotype Mechanism (SM)

This is a very common and straightforward method of creating a UML profile. The stereotype mechanism is an extension of the basic UML elements and is supported by most UML tools. This method is comprised of several steps:

- Assigning a new name to an extended metaclass, which will be represented as a *stereotype* of UML;

- Adding new attributes in the stereotype, which are called *tags*;
- Adding new *constraints* to the stereotype. However, we cannot weaken existing UML metamodel constraints;
- Assigning a new appearance to the stereotype.

Some limitations still exist for this method, including that the modeller may pollute the diagram by mixing domain-specific diagram constructs with predefined UML diagram constructs, hence possibly hurting the understandability of models.

Metamodel Extension Mechanism (MEM)

This mechanism [29], which includes the functionalities of SM, is a less common method of creating a UML profile. It allows the extension of non-basic UML elements and imposes restrictions that allows for the sole use of domain-specific assigned diagrams to stereotypes. While it is more flexible, the MEM is a more complex approach of profiling and is supported by fewer tools than the SM.

2.5.4 Profiling Related Tools

There are a number of tools which provide support for creating UML profiles. Table 1 lists popular commercial UML tools.

Table 1 Tools supporting UML Profiles

Company Name	Tool Name
@-portunity	Blueprint SM
ARTiSAN software	ARTiSAN Studio
IBM	Rational Software Architecture (RSA)
Telelogic (IBM)	Tau G2
MID	Innovator
No Magic	MagicDraw
MDWorkbench-Sodius	MDWorkbench
Objecteering software	Objecteering 6.1

There exist other tools and technologies for domain specific modelling (DSM) that are not based on UML. These include the Generic Modelling Environment (GME)

[11], Xactium's XMF-Mosaic [28], and the Eclipse Modeling Framework (EMF) [7]. There has been previous research which has used a number of factors to analyze which tool is more suitable for prototyping an editor for an evolving graphical language, and a subset of GRL was used as an example [2]. "These evaluation factors are *Graphical completeness* (Can we represent all the notation elements?), *Editor usability* (Does the editor generated support undo/redo, load/save, simple manipulation of notation elements and properties, etc?), *Effort* (How much time and effort is required to learn the approach and produce DSML tools?), *Language evolution* (How are older models handled when the language or metamodel evolves?), *Integration with other languages* (How can we support additional languages(e.g., Use Case Maps in combination with GRL) or integrate with other tools?), *Analysis capabilities* (Can we easily analyze or transform models produces with the graphical editor?)"[2]. UML profiling tools were also part of this study. One of the conclusions was that Telelogic Tau G2 was superior to IBM's RSA [10] in terms of support for UML profiles in that context. In particular, Tau G2 supports both the stereotype mechanism and the metamodel extension mechanism discussed earlier, whereas RSA only supports the former.

2.6. ITU-T Support for Profiles

There is an ITU-T standard document, Recommendation Z.119 [15], which provides guidelines for creating a UML profile for a modelling language. According to these guidelines, a profile can be expressed by the following headings and descriptions:

- **Scope**: The span of the profile, which states the rules for conformance, as well as the notation guidelines for UML.
- **References**: A valid source that provides information about a specific topic. Examples include the relevant ITU-T system design languages, the UML SS, the Object Constraint Language, and if applicable, MOF.
- **Definitions**: The terms and explanations provided in the standard.
- **Abbreviations and acronyms**.
- **Conventions**: Includes name resolution, the drawing style, and formal notations for mapping, translation and transformation.

The main text of the profile is also the subject of many guidelines:

- **Defining the profile**
  - **Stereotype summary**: Ideally comprised of a UML diagram that provides the stereotype definitions and explanations of which class of the UML metamodel maps with which class of a particular language metamodel (like in our case, the GRL metamodel), the stereotype summary can also be presented in a table.
  - **Stereotypes**
  - **UML Metamodel**: This should be an appendix to the profile document that summarizes the relevant parts of the UML metamodel. This is not strictly necessary, but may make the profile document easier to understand.
- **Structure of stereotype sub-clauses**
  - **Attribute**s: the stereotype characteristics that the modellers select from the extended UML metaclass and any new required characteristics. Modellers can set default or initial values, as well as data types.
  - **Constraints**: the rules that are passed down from the extended metaclass. Modellers are also allowed to apply new rules as required.
  - **Semantics**: The mapping between the UML metaclass and the stereotype, as well as the corresponding ITU-T language element. Items for consideration include the new class concept in the new model, as well as its original meaning in the UML metamodel.
  - **Notations**: The creation of new representations if there is any change from the inheriting class.
  - **References**: It specifies where to find related sections of the UML SS and ITU-T Recommendation(s). While strictly this is not needed, it will be invaluable for readers so that the relevant parts of UML SS and ITU-T Recommendation(s) can be found and the effect of the profile understood.

One of the profiles standardized by ITU-T is the Z.109 UML profile for SDL (Specification Description Language) [13]. GRL is very different from SDL. While SDL [12] is a language used for specifications and behaviour descriptions of telecommunica-

tion systems, GRL deals with modelling and evaluating goals and requirements. As the SDL profile mostly extends the UML metaclasses without conceptually altering them, there remains a need to discuss the semantics, value and constraints of every single attribute that the metaclass possesses or inherits from its parent classes.

2.7. Related Work on Goal Modelling Profiles

2.7.1 Requirements for a Profile for Goal-oriented Modelling

Goal-oriented profiles shall support different functionalities and satisfy the quality criteria expressed by the following four requirements:

R1. **Integration with UML**: This is the ability of sharing information between the goal-oriented modelling diagram and the existing UML diagrams. It is apparent that UML diagrams share some information and knowledge with each other. For example, an instance of a class from a class diagram can be used as a lifeline in a sequence diagram. Hence, in this specific case, there must be an evaluation of how the GRL diagram integrates with other UML diagrams.

R2. **Diagram Pollution Avoidance**: Different diagrams offer different viewpoints on the same model. In most UML tools, whenever a UML diagram is created, a separate editor for each diagram type, with a customized toolbar, is used in order to maintain the diagram "purity". The allocation of unique toolbars and editors for each diagram prevents the "pollution" or mixing of different diagram constructs. In order for modellers to use GRL diagrams, based on GRL profile, without polluting other diagrams, and vice-versa, it is important that the profile promotes this feature.

R3. **Metamodel Stability**: A metamodel is considered stable if it is standardized or if it has reached a good level of maturity through tool support and a community of users. The stability of the underlying metamodel is very important for the evaluation of any goal-oriented modelling profile.

R4. **Implementability of the Profiling Mechanism**: The profiling mechanism used must ease the implementation and proper tool support of the profile. This

is important as some profiles are expressed in vague terms or in ways that are incompatible with UML tools.

These requirements will be used to evaluate the work of others and later, in Section 5.3, to evaluate our own work.

2.7.2 Evaluation of Related Work

In this section, we will analyze four research works [5][6][9][25] close to our work. Evaluation and analysis of these works are based on their level of satisfaction to our stipulated requirements.

Research Work 1: A Template-based Analysis of GRL

A template-based analysis of GRL has been conducted by Dallons *et al.* in [6]. However, there is no actual profile for goal modelling in this work. The authors considered three concrete syntaxes for GRL: 1) a textual syntax (expressed in Backus-Naur Form (BNF), 2) a graphical syntax (also expressed in BNF, but augmented with topological information) and 3) an Extensible Markup Language (XML) syntax (expressed as an XML Document Type Definition (DTD)). Other considerations include informal semantic definitions of constructs, examples of GRL models and a tutorial. A conclusion reached by [6] is that a GRL metamodel should be comprised of four different concepts shown in Figure 14 - Figure 17. These are: Actor, IntentionalElement, NonIntentionalElements and IntentionalRelationship.

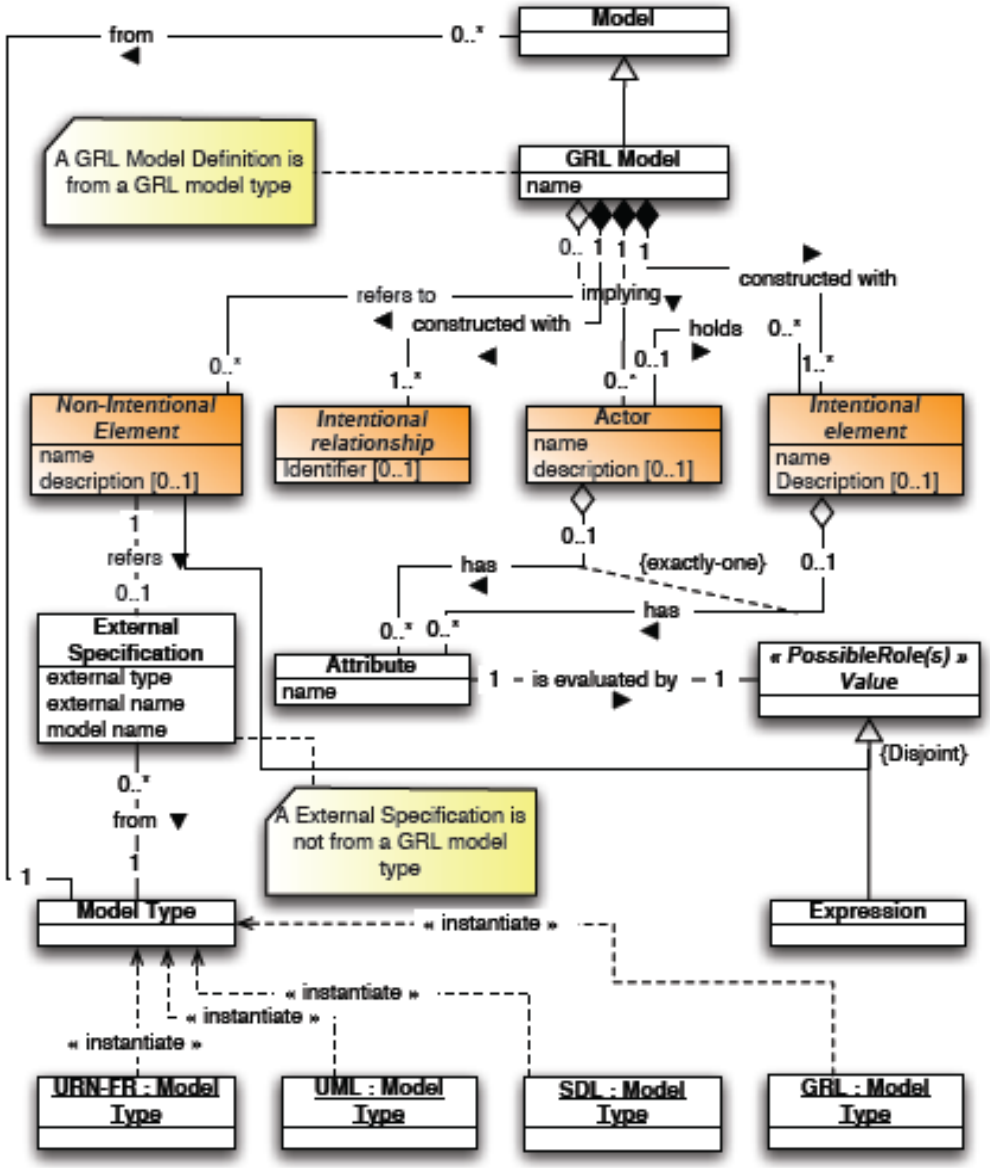

Figure 14 Top Level View of GRL Metamodel [6]

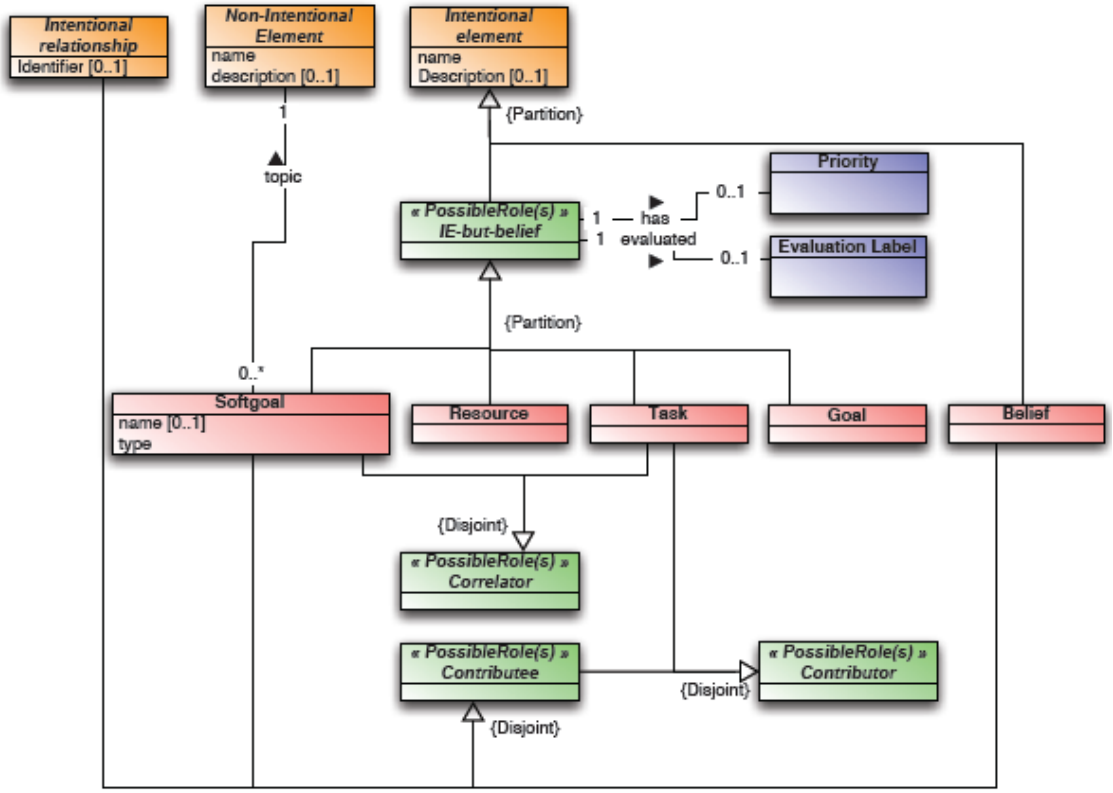

Figure 15 GRL Metamodel: Zoom on Intentional Elements [6]

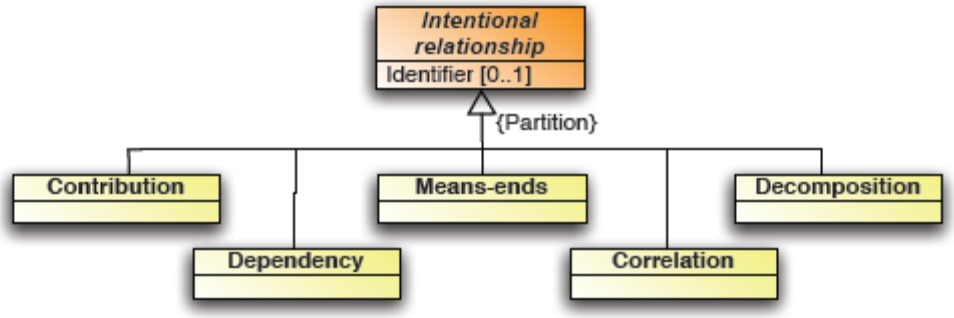

Figure 16 Types of Intentional Relationships [6]

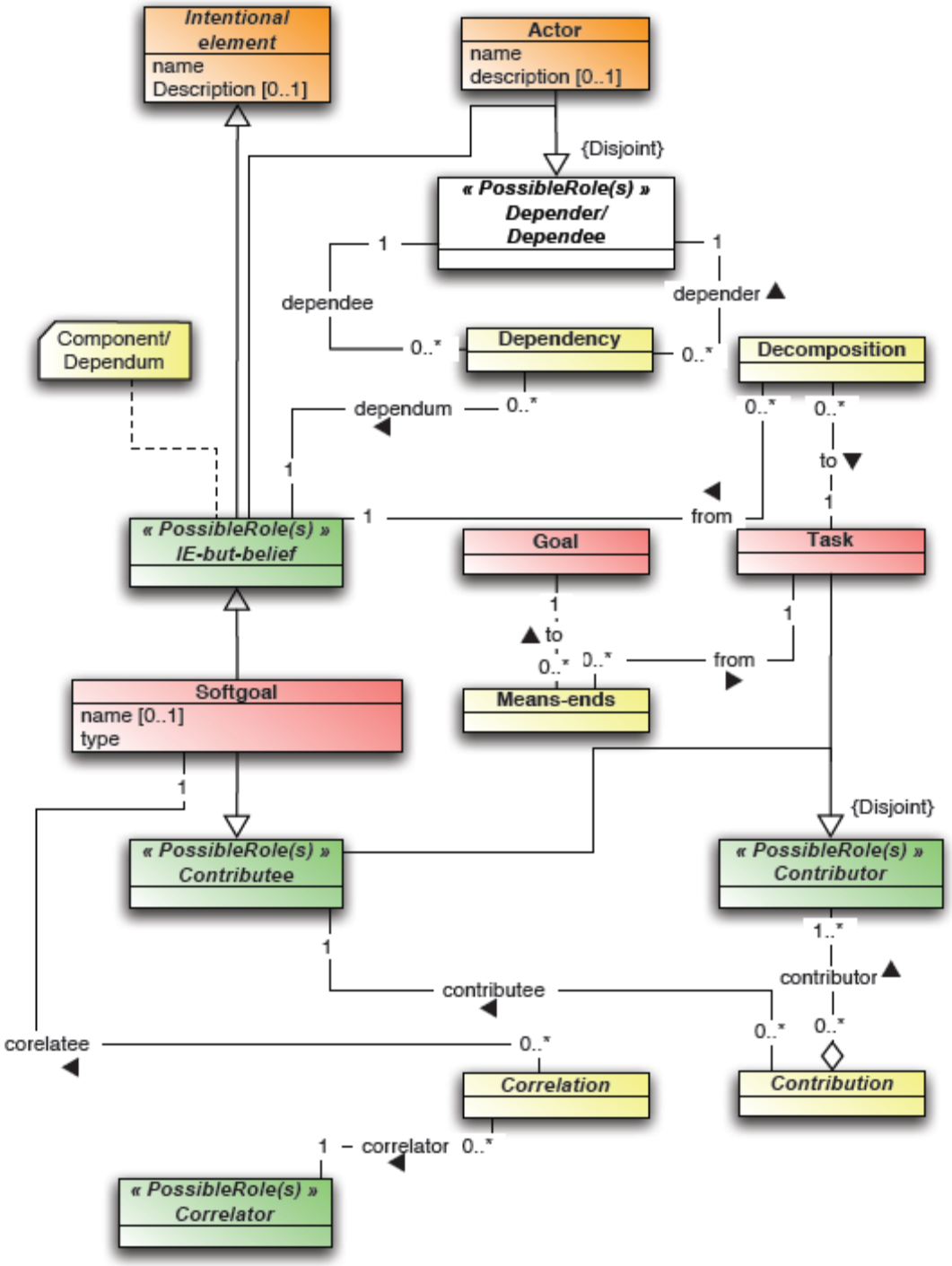

Figure 17 GRL Metamodel: Zoom on Intentional Relationships [6]

The authors of [6] applied an already established template-based analysis on their meta-model. “The template was proposed by Opdahl and Hendeson Sellers as a means to sys-

tematize the description of Enterprise Modelling Language (EML) constructs. It can be used for various purposes like comparing and integrating EML constructs or, simply, for better understanding them. Translating models from one EML to another is another possible use. In Version 1.1 of the template, each construct is defined by filling in the following sections: *Preamble* (General issues are specified here, i.e., constructs, diagram type, language name and version, acronyms and external resources.), *Presentation* (such issues are lexical information (icons, line styles), syntax and layout conventions are specified here.), *Semantics* (It requires the analyst to answer some questions which are in detail discussed in [6]), *Open issues* (All the issues that the template failed to address should be mentioned here.)" [6] The assessment concluded that the GRL specification is very vague and the scope of the definitions is very broad. They noted confusion with the definition and semantics of Actor. They raised a number of questions including whether there should be subtypes of Actor, the difference between roles and actors, as well as issues concerning whether tasks and goals could be decomposed. Additionally, they provided suggestions to improve the GRL textual syntax.

Their research does not demonstrate any integration with the existing UML diagrams. The research was based on a metamodel that was not valid or stable. It can be considered more as a theoretical effort as there is no mention of a practical implementation of the profile in any tool. In the evaluation part below, we give an analysis of how the work in [6] satisfies our requirements.

Evaluation

R1. Integration with UML: It can be concluded that this work does not fulfill requirement R1 since the research does not demonstrate any integration with the existing UML diagrams.

R2. Diagram Pollution Avoidance: This requirement is not satisfied because of the lack of a practical implementation.

R3. Metamodel Stability: The metamodel does not satisfy the requirement for stability. As mentioned in [6], a GRL metamodel did not exist at the time of the work. So the proposed metamodel is a supposed metamodel with no guarantee of validity.

R4. Implementability of the Profiling Mechanism: This requirement is not satisfied because the constructs were only verbally mapped. In order for this requirement to be satisfied, one needs to implement the profile through the use of profile supporting UML tools after mapping it.

Research Work 2: UML Profile for Enterprise Goal Modelling

Figure 18 shows a UML profile proposed by Grangel *et al*. [9] for enterprise goal modelling at the Computation Independent Model (CIM) level. "It is used to represent domain and system requirements. It is based on business models and shows the enterprise from a holistic point of view that is independent of the computation" [9]. The concepts are divided in two parts: CIM-Knowledge and CIM-Business. The top-level general vision of the enterprise and its knowledge are discussed in the first part. The second part involves details of the enterprise knowledge, through the examination of its business representation. The depiction adheres to organisational, structural and behavioural models.

Abstraction Level	Metamodel	UML Profile	Model	Diagram
CIM-Knowledge	Knowledge	for KM	Knowledge	Blocks Ontological Knowledge
CIM-Business	Organisation	for GM	Organisation	Goals
		for OSM		Organisational Structure
		for AM		Analysis
		for BRM		Business Rules
	Structure	for SM	Structure	Product Resource
	Behaviour	for BM	Behaviour	Process Service

Figure 18 Proposal for Enterprise Knowledge Modelling [9]

The work in [9] is based on the Unified Enterprise Modeling Language (UEML), and the Process, Organisation, Product, and so on (POP*[2]). They proposed a metamodel for goal modelling shown in Figure 19:

[2] In *POP**, * means so on

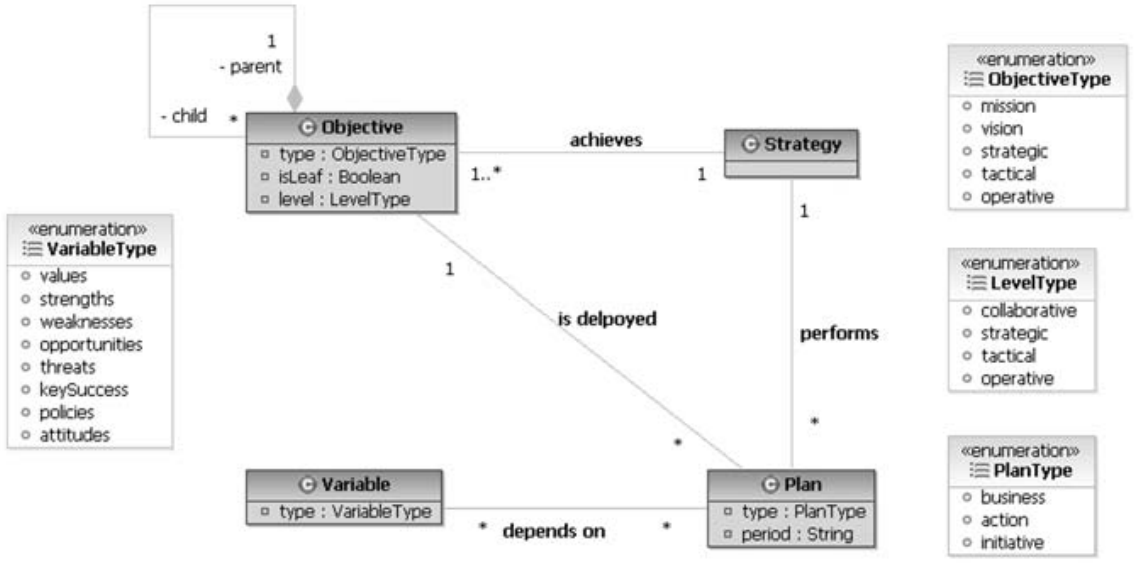

Figure 19 Goal Metamodel: Organizational Metamodel Excerpt [9]

The metamodel distinguishes four different conceptual constructs: "*Objective* (this represents any target that enterprises want to achieve, it is possible to define it at different hierarchical levels, such as strategic, tactical and operative levels. At the strategic level, this construct is also used to represent the enterprise's mission and vision.), *Strategy* (this represents how the enterprise wants to achieve the objectives proposed at the strategic level), *Plan* (this represents the organisation of the work at different hierarchal levels in order to accomplish the objectives and strategy defined in the enterprise) and *Variable* (this represents any factor that is able to influence the execution of the plans defined in the organisation)" [9].

The profile has been implemented in IBM Rational Software Modeler Development Platform and in MagicDraw UML 12.0. Figure 20 shows stereotypes created for the implementation of the profile. Below is an evaluation of this work from our perspective.

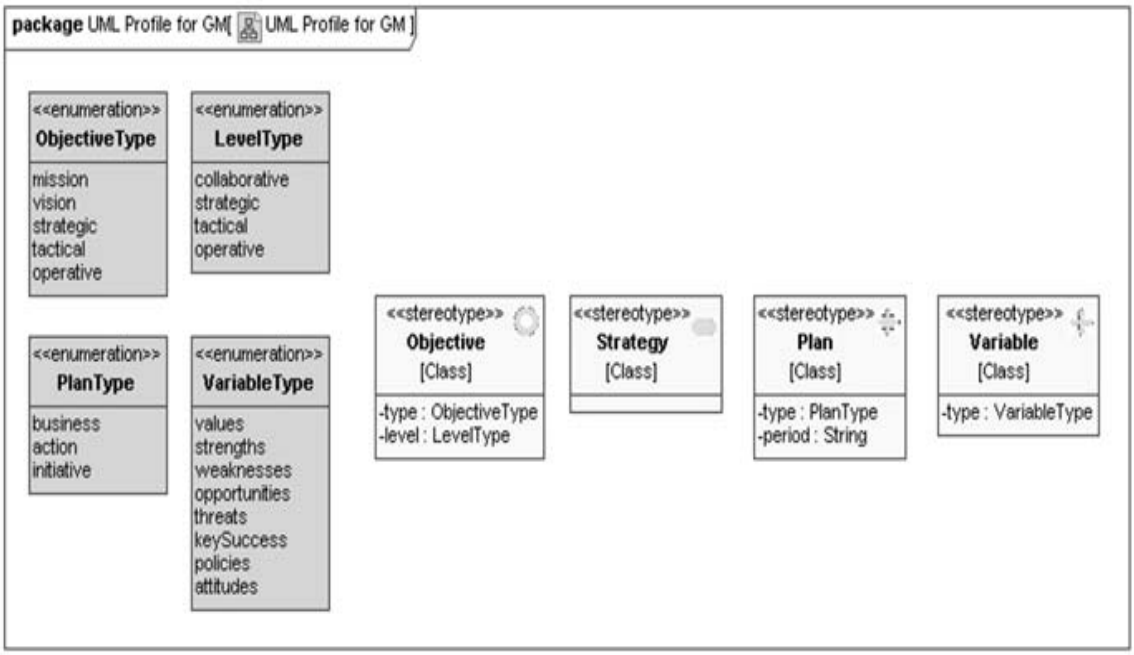

Figure 20 Diagram of the UML Profile for Enterprise Goal Modelling [9]

Evaluation

R1. Integration with UML: This requirement can be satisfied by [9]. It is supported by most tools to reuse the constructs of UML diagrams. It is apparent that they created stereotypes, so they can be reused in other UML diagrams.

R2. Diagram Pollution Avoidance: The implementation is based on SM, which does not allow for the creation of a separate editor. The modellers are therefore forced to use a different diagram editor (most likely a Class diagram editor) that allows for their metamodel constructs to become polluted through the mixing with other UML diagram constructs (e.g. with Class diagram constructs).

R3. Metamodel Stability: We cannot assert that the metamodel fulfills all aspects of Goal-oriented Modelling, since there is no pre-existing editor based on their proposed metamodel. Their metamodel is neither a standard nor in an ongoing process of standardization.

R4. Implementability of the Profiling Mechanism: This requirement is partially satisfied, as the profile is implemented in the IBM Rational Software Modeller Development Platform and MagicDraw UML 12.0 through the use of Stereo-

type Mechanism (whereas Metamodel Extension Mechanism is the recommended approach for profiling [18][29]). Our claim of partial satisfaction of this requirement by their work is motivated by the use of the simple stereotype mechanism for implementation. As mentioned in [18][29], the metamodel extension mechanism is recommended.

Research Work 3: UML Profile for Softgoal by Use Case Driven Approach

Supakkul and Chung [25] proposed a metamodel for NFR concepts, and its integration with the UML metamodel through the use of the UML profile extension mechanism. The integration of UML and NFR notations occurs in a Use Case diagram. NFRs are represented as softgoals and associated with appropriate Use Case model elements as depicted in Figure 21.

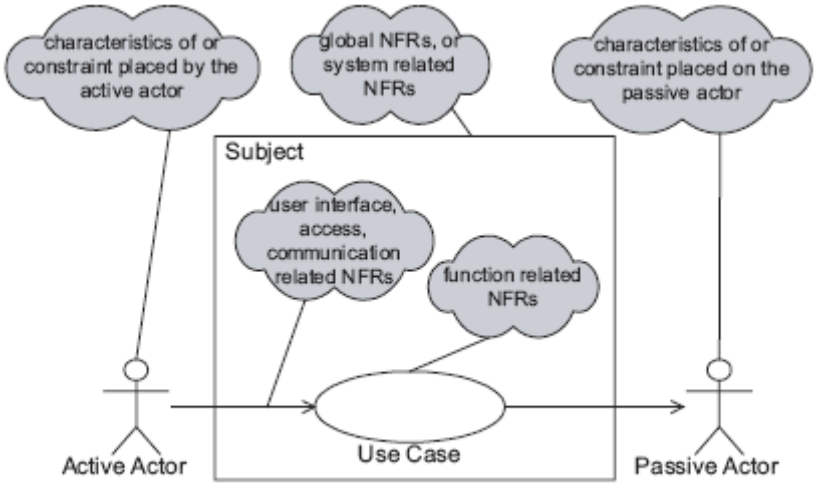

Figure 21 NFR Association Points in UseCase Model for NFR Types [25]

The profile is also referred to as *Softgoal Profile* because NFRs are considered as softgoals. It consists in two parts:

- **Softgoal Interdependency Graph (SIG) subprofile**: A representation of the concepts used in the NFR framework."The NFR Framework is a goal-oriented approach for addressing NFRs. In this framework, they represent NFRs as NFR softgoals to be satisficed."[25]

- **Procedure subprofile**: A representation of the concepts that are related to the method, correlation rule and evaluation procedures. The Procedure subprofile may import the SIG subprofile for use in its stereotypes.

The modeller can use these profile stereotypes with the combination of UML standard notations. The authors implemented their work and used the London ambulance dispatch system as a case study.

Evaluation

R1. Integration with UML: The authors of [25] did not mention which mechanism was considered for their profile implementation. From their diagrams and references, it appears that they used SM. As a result, they fulfill our stipulated requirement.

R2. Diagram Pollution Avoidance: The constructs can be polluted by the UML diagram constructs. The authors mention that their profile allows users to model using a combination of UML standard notations and their stereotypes.

R3. Metamodel Stability: The metamodel is only partially stable, as it is inspired from the NFR framework which does not fully address the goal-modelling domain. Furthermore, there is no editor solely based on the metamodel.

R4. Implementability of the Profiling Mechanism: We are not able to assess the work based on this criterion. It is not clear from [25] which tool the authors used for the implementation of their work, or whether it is purely a theoretical concept.

Research Work 4: Using UML to Reflect Non-Functional Requirements

Cysneiros *et al*. [5] based their study on the idea that there should be an integration of NFRs with functional requirements. They remarked that there are two cycles to the software development process: one which covers functional requirements and the second, which comprises non-functional requirements. There is no integration of these two cycles. They concluded that there is a requirement for a junction point that would integrate these two independent cycles. Figure 22 shows their proposed strategy for integration.

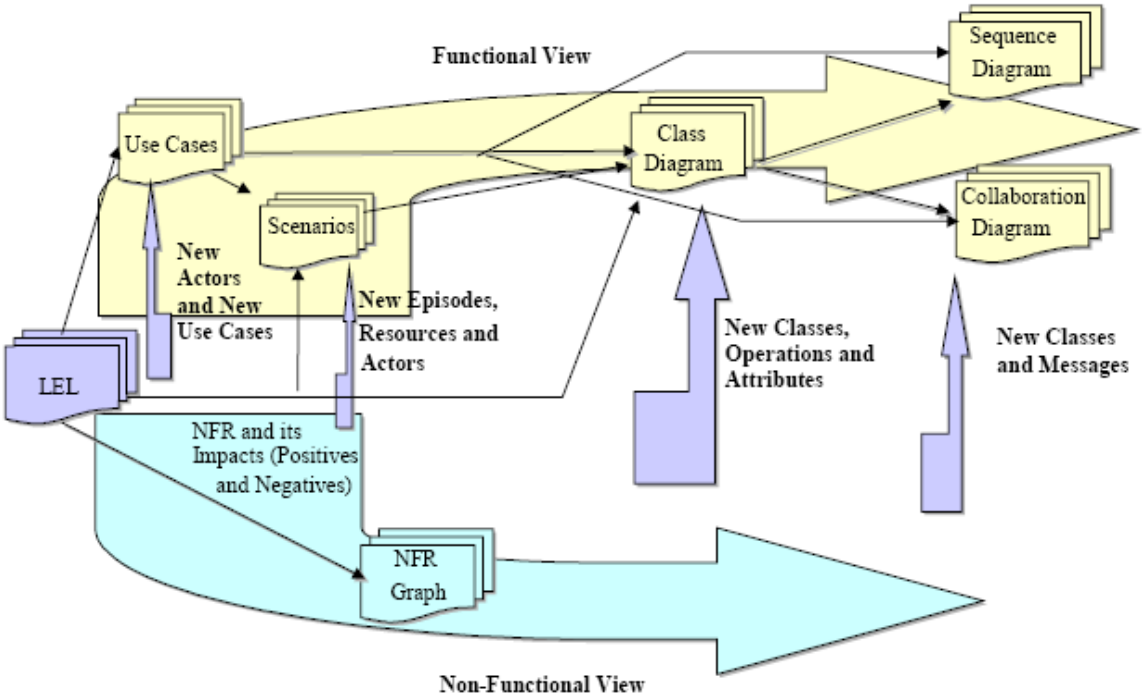

Figure 22 An Strategy Overview for Dealing with NFR [5]

In this strategy, the integration point is called Language Extended Lexicon (LEL). Functional and non functional views are joined at that point. The junction point is used to register the words or phrases unique to a specific field of the application. A LEL can hold information from both functional and non functional requirements. "To build the NFR model one must search all entries of the LEL looking for notions that express the need for an NFR. For each NFR found, create an NFR graph expressing all the operationalizations that are necessary to satisfice this NFR. It resulted, at end, a set of NFR graphs that represent the non functional aspects of the system"[5]. Cysneiros *et al*. claim that their proposed solution covers all diagrams. For instance, Figure 23 shows how NFR graphs are integrated with Class diagrams. The term *integration* means in this context that "every root of each NFR graph must refer to an LEL symbol and every class of the Class diagram must be named using an LEL symbol" [5]. Cysneiros *et al*. also proposed some heuristics for the convergence process.

- They created a stereotype <<NFR>> for classes which are used to satisfy a NFR.
- They represented a link to the non-functional view beside each operation that has been included to satisfy an NFR.

- Each operation may have pre- and post conditions.
- An attribute can be added to satisfy a NFR and it may use the same expression as in the operations to create a link to the non-functional view.

The approach has been validated through case studies.

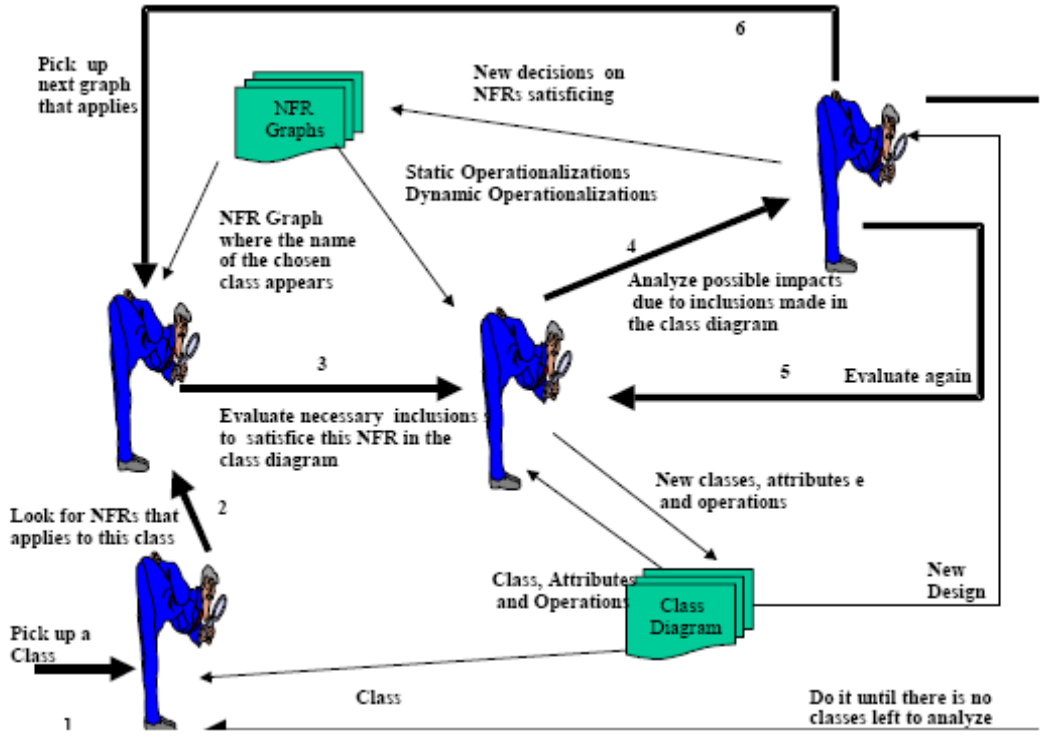

Figure 23 The Class Diagram Integration Process [5]

Evaluation

R1.Integration with UML: This work does not meet the integration criteria.

R2.Diagram Pollution Avoidance: This work cannot be evaluated according to this requirement because the proposed strategy differs from UML profiling. They did not create any stereotype.

R3.Metamodel Stability: For the same reason as above, the work cannot be evaluated according to this requirement.

R4.Implementability of the Profiling Mechanism: For the same reason as above, the work cannot be evaluated according to this requirement.

Overall Analysis of Previous Work

Table 2 gives the overall view of previously discussed research works and their satisfaction level with our stipulated requirements. It is observed that none of the discussed approaches satisfies all four requirements. Some requirements are not applicable (N/A) with their work (and we suspect they are not satisfied) and some are partially satisfied.

Table 2 Overview of Previous Work Contributions

	Integration with UML	**Diagram Pollution Avoidance**	**Metamodel Stability**	**Tool Support**
A Template based analysis of GRL	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied
UML profile for Enterprise Goal Modelling	Satisfied	Not Satisfied	Not Satisfied	Partially Satisfied
UML profile for Softgoal by use case driven approach	Satisfied	Not Satisfied	Partially Satisfied	N/A
Using UML to reflect non-functional requirements	Not Satisfied	N/A	N/A	N/A

2.8. Chapter Summary

In this chapter, we provided a review of the UML architecture, examined the different levels of metamodelling, discussed URN in detail and reviewed some related work on goal-oriented modelling profiles.

The UML architecture is divided into two parts known as UML infrastructure and UML superstructure. The different levels of metamodel are referred to as M3, M2, M1 and M0. The MOF is at the meta-metamodel level (M3), the UML at the metamodel level (M2), user-specified models are at level M1, and objects at level M0. The URN includes the UCM notation and GRL. The former is used for modelling scenarios and functional requirements, while GRL exists specifically for modelling goals and non-functional requirements. GRL's main elements include Intentional elements, Actors and Element

links. We also discussed the GRL metamodel and presented GRL strategies used for model evaluation.

Profiling was discussed as a UML feature that is used to customize the UML metamodel because modellers are prevented from directly customizing the UML meta-model. There are two approaches for creating a profile: a Stereotype Mechanism and a Metamodel Extension Mechanism. The ITU-T has published a Recommendation on the creation of UML profiles for modelling languages.

We gave a brief analysis of four goal modelling and profiling approaches related to our work and compared them against four requirements. While there have been some interesting contributions, existing approaches do not fulfill our established criteria for a goal-oriented modelling profile.

Chapter 3. UML Profile for GRL

In chapter two, we discussed prerequisites for writing a GRL profile in detail. The current chapter embodies the core work of this thesis, including how we map the UML metamodel to the original GRL metamodel. In section 3.1, we discuss naming conventions which are used in this profile. Section 3.2 provides a summary of the GRL metaclasses and their extensions from the UML metaclasses in tabular form. In section 3.3, we separately map each GRL metaclass with a UML metaclass. We also discuss GRL metaclasses semantics, attributes, constraints and notations. Section 3.4 then provides a global overview of the GRL profile.

GRL model elements have an *id* attribute that is unique inside the model and that does not change as new versions of the model are created. It is used to enable traceability from between versions of a GRL model and between a GRL model and external requirements or other types of models. GRL model elements also have *names* but these *names* can be modified from one version to the next, and their uniqueness has a scope that is smaller than the model (e.g., actor names are all different, but the name of an actor can be the same as the name of an intentional element).

3.1. Conventions, Names and Template

3.1.1 Conventions

This work uses the conventions defined in ITU-T Rec. Z.119 [15]. These conventions are repeated below:

- A *term* in this profile is a sequence of characters making up either an English word or a concatenation of English words. The sequence of characters indicates the meaning of the term.
- An *underlined term* refers to a UML term or a term defined by a stereotype in this profile. A term beginning with a *capital letter* denotes a metaclass.

- A term preceded by the word "*stereotype*" designates a UML stereotype according to the stereotype concept defined in the UML Superstructure specification documentation (usually in a phrase “The stereotype X extends the metaclass X” where X is a term). The stereotype is required when its multiplicity is [1..1] (that is the derived attribute isRequired of the Extension association between the extended metaclass and the stereotype is true). If the multiplicity of the stereotype is [0..1], the stereotype is not required.
- The convention for a term enclosed in << and >> is extended to allow GRL qualifiers to be used. The convention for a term enclosed in < and > is extended to allow GRL concrete syntax to be used. A convention on the meaning of terms in italics is added.
    - A term enclosed in << and >> refers to a stereotype described by this profile. These terms are not underlined.
    - A term in italic in a stereotype description refers to a GRL abstract syntax item.
    - A metaclass enclosed in << and >> and preceded by “GRL” is a GRL metamodel class and is not underlined. Here the term in <<and>> is the name of a GRL metaclass.
- All owned attributes of stereotypes are in the attribute section and all inherited attributes are in the semantics section.

In addition, the prefix “/” is often used to indicate a UML association role (e.g., /superClass).

3.2. Stereotype Summary

The following table lists the stereotypes with the UML metaclass that each stereotype extends. A description of the stereotypes is provided in section 3.3.

Table 3 Stereotype, Metaclass Mapping Information

Stereotype	Stereotyped UML metaclass
GRLspec	Model
GRLmodelElement	NamedElement
GRLLinkableElement	Class
Actor	Class
IntentionalElement	Class
IntentionalElementType	Enumeration
ImportanceType	Enumeration
ElementLink	Relationship
Contribution	Association
ContributionType	Enumeration
Dependency	Association
Decomposition	Association
DecompositionType	Enumeration

3.3. Structure of the Goal-oriented Requirement Language (GRL) Profile

3.3.1 GRLspec

The stereotype GRLspec is mapped from the metaclass Model with multiplicity [0..1].

Note: The stereotype GRLspec is intended to work as a container for GRL specifications.

Attributes

No attributes.

Constraints

No constraints.

Semantics

None. It is a structural concept.

Notation

There is no notation for <<GRLspec>> Model.

References

- UML SS: 17.3.1 Model (from Models).

3.3.2 GRLmodelElement

The stereotype GRLmodelElement extends the metaclass NamedElement with multiplicity [0..1].

Note: This stereotype contains GRL concept metadata to associate with other model elements.

Attributes

Stereotype attributes:

- id: String — Defines the identifier of the <<GRLmodelElement>> NamedElement.

Constraints

- <<GRLmodelElement>> NamedElement has a unique id tag within the URN specification.
- Each <<GRLmodelElement>> NamedElement must have a unique name.

Semantics

Attribute *name*: String of <<GRLmodelElement>> NamedElement is mapped with the name of NamedElement.

Notation

There is no notation for <<GRLmodelElement>> NamedElement.

References

- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies).

3.3.3 GRLLinkableElement

The stereotype GRLLinkableElement extends the metaclass Class with multiplicity [0..1]

Note: The stereotype GRLLinkableElement is intended for generalizing GRL <<Actor>> and GRL <<IntentionalElement>>.

Attributes

Stereotype attributes:

- id: String Defines the identifier of the <<GRLLinkableElement>> Class.

Constraints

- <<GRLLinkableElement>> Class has an id tag that must be unique within the URN specification.
- Each <<GRLLinkableElement>> Class must have a unique name.

Semantics

The attribute *name*:String of <<GRLLinkableElement>> Class is mapped with the attribute name of NamedElement.

<<GRLLinkableElement>> Class contain common properties with both GRL<<Actor>> and GRL<<IntentionalElement>>.

Notation

There is no notation for <<GRLLinkableElement>> Class.

References

- UML SS: 7.3.7 Class (from Kernel)
- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies)

3.3.4 Actor

The stereotype Actor extends the metaclass Class with multiplicity [0..1].

Note: In this profile, Class behaves as a new GRL<<Actor>> concept. Here, GRL <<Actor>> behaves as a container and stakeholder. A GRL<<Actor>> can depend on other actors and intentional elements and can also contain other actors and intentional elements.

Attributes

Stereotype attributes:

- id: String Defines the identifier of the <<Actor>> Class.

Constraints

- An <<Actor>> Class cannot include itself, either directly or indirectly.
- An <<Actor>> Class has an id tag that must be unique within the URN specification.
- Each <<Actor> Class must have a unique name.
- An <<Actor>> Class has attribute isLeaf (inherited from RedefineableElement) = True. It is therefore not possible to further specialize <<Actor>> Class.

Semantics

Class describes a set of objects that share the same specifications of features, constraints and semantics. Its features are comprised of attributes and operations. In the same way, <<Actor>> Class has attributes and operations. Instances of Property are represented by attributes, which map to the <<Actor>> Class attributes.

The nestedClassifier is an association owned by Class. This association has on the other end a Classifier. This association is used for referencing all the Classifiers that are defined within the Class. GRL<<Actor>> owns an association *"elems"* with GRL<<IntentionalElement>>. As GRL<<Actor>> can contain GRL<<IntentionalElement>>s, that same behaviour maps nestedClassifier with *elems*.

GRL<<Actor>> also owns an association *includingActor* [0..1] and *includedActors* [0..*]. This association provides the reference of including GRL<<Actor>> which have other GRL<<Actor>>s included. It helps to find out which GRL<<Actor>> contains other GRL<<Actor>>s. This same behaviour is the reason to map nestedClassifier with *includingActor* with multiplicity [0..1] and *includedActors* with multiplicity [0..*].

Attribute *name*: String of GRL<<Actor>> is mapped with name of Class, which is inherited from NamedElement.

Notation

An <<Actor>> Class is represented by a "dashed circle" with the accompanying actor name, as shown in Figure 24.

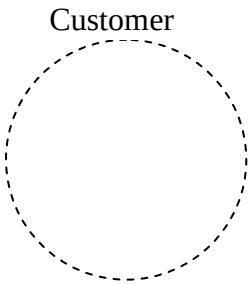

Figure 24 Actor

References

- UML SS: 7.3.7 Class (from Kernel).
- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies).
- UML SS: 7.3.46 RedefineableElement (from Kernel).

3.3.5 IntentionalElement

The stereotype IntentionalElement extends the metaclass Class with multiplicity [0..1].

Note: In this profile, Class is introduced with the concept of a linkable element used by the model. Stereotype IntentionalElement can be decomposed into sublevels. It can also be evaluated by assigning a qualitative and quantitative importance level. Stereotype IntentionalElement has both, inherited and owned attributes.

Attributes

Stereotype attributes:

- id: String — Defines the identifier of the <<IntentionalElement>> Class.
- type: IntentionalElementType — This is *enumeration* data type. It defines the different types of GRL<<IntentionalElement>> like *Softgoal*, *Goal*, *Task*, *Resource* and *Belief*.
- decompositionType: DecompositionType — This is the *enumeration* data type. Its possible values are *AND*, *XOR* and *IOR*. Its default or initial value is *AND*. It defines the different type of *decomposition* when GRL<<IntentionalElement>> is the source of the decomposition link.

• importance: ImportanceType	This is the *enumeration* data type. Its possible values are *High, Medium, Low,* and *None*. Its default value is *None*. It is used to evaluate the importance level of the intentional element quality to its owning actor when specified.
• importanceQuantitave: Integer	Defines the evaluation of the quantitative importance of GRL<<IntentionalElement>> on its GRL<<Actor>>. Its value ranges from 0 to 100, with 0 as default.

Constraints

- <<IntentionalElement>> Class has a tag importanceQuantitative which value must be ≥ 0 and ≤ 100.
- <<IntentionalElement>> Class has a tag id which must be unique within URN specification.
- Each <<IntentionalElement>> Class must have a unique name

Semantics

The <<IntentionalElement>> Class has an association with GRL <<Actor>>. It specifies the reasons for including particular behaviours, information and structural aspects in a system's requirements. There are different types of intentional elements corresponding to different types of behaviour and information elements. These various types have different notations and are very flexibly linked to each other.

The <<IntentionalElement>> Class has a tag importance that captures an actor's level of interest in the included intentional element. However, it is not mandatory that modellers use both the importance and importanceQuantitative tags. The selection depends on a modeller's requirements for the desired analysis type, either qualitative, quantitative, or mixed.

Class is given its attribute name by inheritance from NamedElement. The name attribute maps to the *name*: String attribute of <<IntentionalElement>> Class.

The nestedClassifier is an association owned by Class. This association has on the other end a Classifier. This association is used for referencing all the Classifiers that are defined within the Class. GRL<<IntentionalElement>> also owns an association a*ctor* with multiplicity [0..1]. This association provides the reference of GRL<<Actor>> which contains other GRL<<IntentionalElement>>s. It helps to find out which GRL<<Actor>> owns which GRL<<IntentionalElement>>s. This same behaviour is the reason to map nestedClassifier with a*ctor* with multiplicity [0..1].

decompositionType is only effective when GRL<<IntentionalElement>> has GRL<<ElementLink>> of type GRL<<Decomposition>> and the value of *type* is not *Belief*.

Notation

An <<IntentionalElement>> Class has different types as mentioned above in semantics. Each type has a separate notation:

- Softgoal:

Figure 25 Softgoal

- Goal:

Figure 26 Goal

- Task:

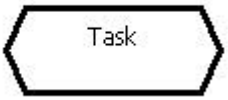

Figure 27 Task

- Resource:

Figure 28 Resource

- Belief:

Figure 29 Belief

References

- UML SS: 7.3.7 Class (from Kernel).
- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies).

3.3.6 IntentionalElementType

The stereotype IntentionalElementType extends the metaclass Enumeration with multiplicity [0..1].

Note: The stereotype is also a user-defined data type. An instance of stereotype IntentionalElementType is a value of GRL<<IntentionalElementType>> attribute *type*. There are five values: *Softgoal, Goal, Task, Resource,* and *Belief.*

Attributes

No attributes.

Constraints

No constraints

Semantics

The <<IntentionalElementType>> Enumeration instance has different values taken by an attribute *type* of GRL<<IntentionalElementType>>. These values have their own meaning and are used according to modelling requirements.

Goal, Softgoal, Task, Resource and *Belief* are all instance values of <<IntentionalElementType>> Enumeration with extensive use in goal-oriented modelling.

Every instance of <<IntentionalElementType>> Enumeration is a value that is mapped with EnumerationLiteral.

Notation

There is no notation of <<IntentionalElementType>> Enumeration.

References

- UML SS: 7.3.16 Enumeration (from Kernel).
- UML SS: 7.3.17 EnumerationLiteral (from Kernel).

3.3.7 ImportanceType

The stereotype ImportanceType extends the metaclass Enumeration with a multiplicity of [0..1].

Note: This stereotype ImportanceType is a user-defined data type. Its literal values are used by the attribute *importance* of GRL<<IntentionalElement>>. These literal values are *High, Medium, Low* and *None*.

Attributes

No attributes.

Constraints

No constraints.

Semantics

<<ImportanceType>> Enumeration has different literal values which are used by attribute *importance* of GRL<<IntentionalElement>>.

Every literal of <<ImportanceType>> Enumeration is a value which is mapped with EnumerationLiteral.

Notation

There is no notation for <<ImportanceType>> Enumeration.

References

- UML SS: 7.3.16 Enumeration (from Kernel).
- UML SS: 7.3.17 EnumerationLiteral (from Kernel).

3.3.8 ElementLink

The stereotype ElementLink extends the metaclass Relationship with a multiplicity of [0..1].

Attributes

Stereotype attributes:

- id: String — Defines the identifier of the <<ElementLink>> Relationship.
- name: String — The name of the <<ElementLink>> Relationship.

Constraints

- <<ElementLink>> Relationship has a Tag id that must be unique within the URN specification.
- Each <<ElementLink>> Relationship must have a unique name.
- As <<ElementLink>> Relationship is used to connect GRL<<GRLLinkableElement>>s, the source and destination GRL<<GRLLinkableElement>>s must be different.

Semantics

The purpose of <<ElementLink>> Relationship is to show the intentional relationship among GRL<<GRLLinkableElements>> which include GRL<<Actor>> and GRL<<IntentionalElement>>.

The relatedElement is a derived union (of all elements) used in Relationship. On the other end of this association is Element, which has a children association with Class. This association specifies the elements related by the Relationship. GRL<<ElementLink>> has associations named *dest* and *src* with GRL<<GRLLinkableElement>>. They also bear the same functionality. So relatedElement can be mapped with *dest* and *src.*

<<ElementLink>> Relationship is a directed link that is used to connect a source GRL<<Actor>> or a source GRL<<IntentionalElement>> to a different destination.

Notation

There is no notation for <<ElementLink>> Relationship. However, its subclasses have their own notations, which are used according to modelling requirements.

References

- UML SS: 7.3.47 Relationship (from Kernel).

3.3.9 Contribution

The stereotype Contribution extends the metaclass Association with multiplicity [0..1]. **Note**: The stereotype Contribution is a link which illustrates how a source GRL<<IntentionalElement>> helps with the satisfaction of a destination GRL<<IntentionalElement>>.

Attributes

Stereotype attributes:

- id: String — Defines the identifier of the <<Contribution>> Association.
- contribution: ContributionType — An *enumeration* datatype. Its possible values are *Make, Help, SomePositive, Unknown, SomeNegative, Hurt,* and *Break*. Its default value is *Unknown.* This attribute assigns a qualitative value of contribution to GRL<<IntentionalElement>>.
- quantitativeContribution: Integer — A primitive datatype, its default value is 0. This attribute assigns a quantitative value of contribution to GRL<<IntentionalElement>>.
- correlation: Boolean — The link is a GRL<<Contribution>> when the value is true and a *correlation,* which is also like a *contribution* when the value is false. The side effect is also shown in the later case. The default value is false.

Constraints

- <<Contribution>> Association has a tag id that must be unique within the URN specification.
- Each <<Contribution>> Association must have a unique name.
- Only GRL<<IntentionalElement>> can be a source or a destination of <<Contribution>> Association.
- <<Contribution>> Association cannot link any GRL<<Actor>>.
- An instance of <<Contribution>> Association is a link with two ends, a source and a destination. A GRL<<IntentionalElement>> used as destination must not be a *resource* or a *belief.*
- The upper and lower range of attribute *quantitativeContribution* must be ≥ -100 and ≤ 100.

Semantics

Attribute *name*: String of <<Contribution>> Association is mapped with the name of the Class, which is inherited from NamedElement.

A <<Contribution>> Association is a primary required effect in goal-oriented modelling. A *correlation* is a side effect and not a primary requirement.

A <<Contribution>>Association can be used to define the qualitative and quantitative impact level put by a source GRL<<IntentionalElement>> on a destination GRL<<IntentionalElement>>.

Correlations behave in the same way as <<Contribution>> Association, but their consideration includes the side effects between the different categories of GRL<<IntentionalElement>> and also between the different categories of GRL<<IntentionalElement>> and GRL<<Actor>>.

Additional constraints may be applied by modellers according to specific modelling requirements on <<Contribution>> Association.

Notation

A *contribution* is a solid arrow, while a *correlation* is a dashed arrow, as shown in Figure 30 and Figure 31:

Figure 30 Contribution

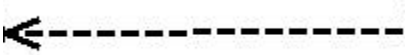

Figure 31 Correlation

References

- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies)
- UML SS: 7.3.3 Association (from Kernel)

3.3.10 ContributionType

The stereotype ContributionType extends the metaclass Enumeration with multiplicity [0..1].

Note: It is intended to assign some qualitative values to the attribute *contribution* of stereotype Contribution. Its user-defined literal values are *Make, Help, SomePositive, Unknown, SomeNegative, Hurt* and *Break.*

Attributes

No attributes.

Constraints

No constraints.

Semantics

<<ContributionType>> Enumeration is a user-defined data type. Each of its literal values is mapped with EnumerationLiteral, which is a *value*. This *value* is used to assign the qualitative contribution to the GRL<<IntentionalElement>> through a link. All of the values have their own notational presentation and weight based on a positive or negative sense.

Notation

The notations for <<ContributionType>>Enumeration are:

- **Make**: This is a positive contribution that sufficiently affects stakeholder satisfaction when evaluated; see Figure 32.

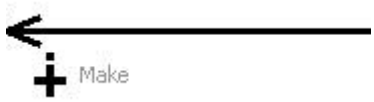

Figure 32 Contribution Type: Make

- **Help**: This is also a positive but insufficient contribution. See Figure 33.

Figure 33 Contribution Type: Help

- **SomePositive**: This is a positive contribution, but the level of its impact or extent is not known; see Figure 34.

Figure 34 Contribution Type: SomePositive

- **Unknown**: It is known that there is some contribution, but it is unknown whether the impact of that contribution is positive or negative. This contribution type has no particular symbolic representation and uses the same notation as the GRL<<Contribution>> arrow.
- **SomeNegative**: This is a negative contribution with an unknown extent. See Figure 35.

Figure 35 Contribution Type: SomeNegative

- **Hurt**: This contribution is negative and it is insufficient. See Figure 36.

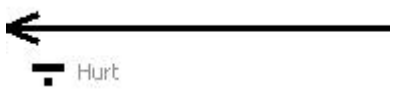

Figure 36 Contribution Type: Hurt

- **Break**: This contribution is negative and sufficient. See Figure 37.

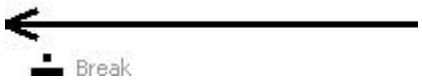

Figure 37 Contribution Type: Break

References

- UML SS: 7.3.16 Enumeration (from Kernel).
- UML SS: 7.3.17 EnumerationLiteral (from Kernel).

3.3.11 Dependency

The stereotype Dependency extends the metaclass Association with a multiplicity of [0..1].

Note: The stereotype Dependency is used to create a link that expresses the dependencies between GRL<<Actors> for GRL<<IntentionalElements>>.

Attributes

Stereotype attributes:

- id: String Defines the identifier of the <<Dependency>> Association.

Constraints

- <<Dependency>> Association has a tag id that must be unique within the URN specification.
- Each <<Dependency>> Association must have a unique name.
- GRL<<IntentionalElement>> *Belief* can never be the source or destination of a <<Dependency>> Association.

- A <<Dependency>> Association is a link that is used to join at least one of the linkable elements GRL<<Actor>> or GRL<<IntentionalElement>>, that are contained in a GRL<<Actor>>.

Semantics

Attribute *name*: String of <<Dependency>> Association is mapped with name of Class, which is inherited from NamedElement.

<<Dependency>> Association is a construct that enables reasoning about how GRL<<Actors>> depend on each other to achieve their *Goals*.

Different configurations of <<Dependency>> Association can be used. Some scenarios are discussed below:

- [source GRL<<Actor>> → depends → GRL<<IntentionalElement>> (not contained in any GRL<<Actor>>) →destination GRL<<Actor>] means that Source GRL<<Actor>> depends on the destination GRL<<Actor>> for the GRL<<IntentionalElement>> which is not contained in any GRL<<Actor>>. See Figure 38.

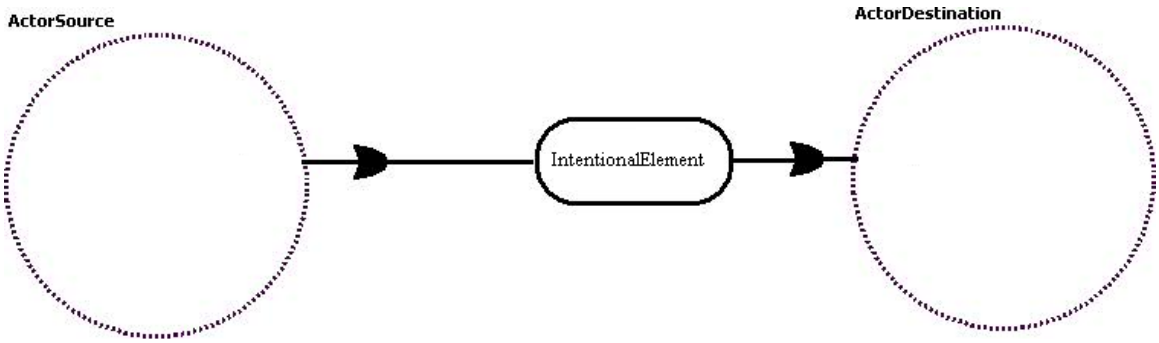

Figure 38 Dependency Scenario 1

- [source GRL<<Actor>> → depends → GRL<<IntentionalElement>> (contained by destination GRL<<Actor>>] means that Source GRL<<Actor>> depends on the destination GRL<<Actor>> for the GRL<<IntentionalElement>> which is contained inside the destination GRL<<Actor>>. See Figure 39.

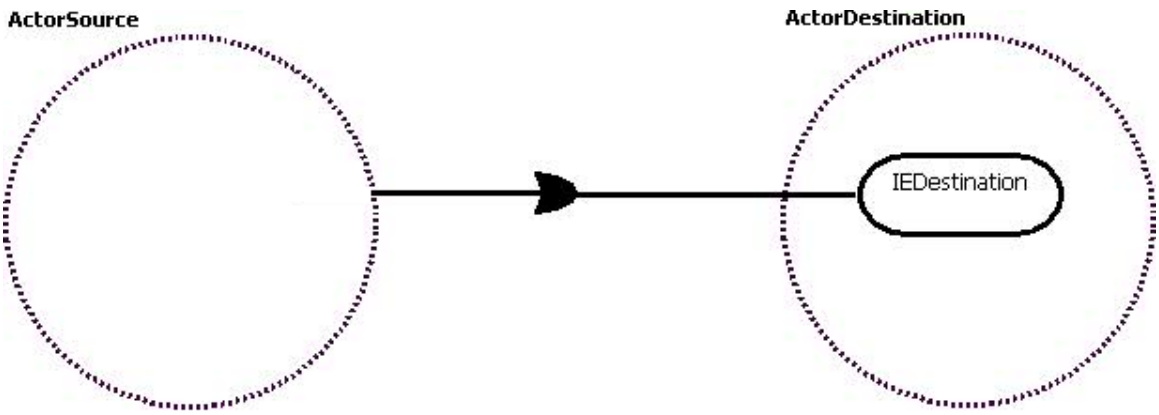

Figure 39 Dependency Scenario 2

- [GRL<<IntentionalElement>> (which is contained by source GRL<<Actor>>) → depends → GRL<<IntentionalElement>> (not contained in any GRL<<Actor>>) → depends → destination GRL<<Actor>>], meaning that GRL<<IntentionalElement>>, which is contained by source GRL<<Actor>>, depends on the destination GRL<<Actor>> for the GRL<<IntentionalElement>> which is not contained by any GRL<<Actor>>. See Figure 40.

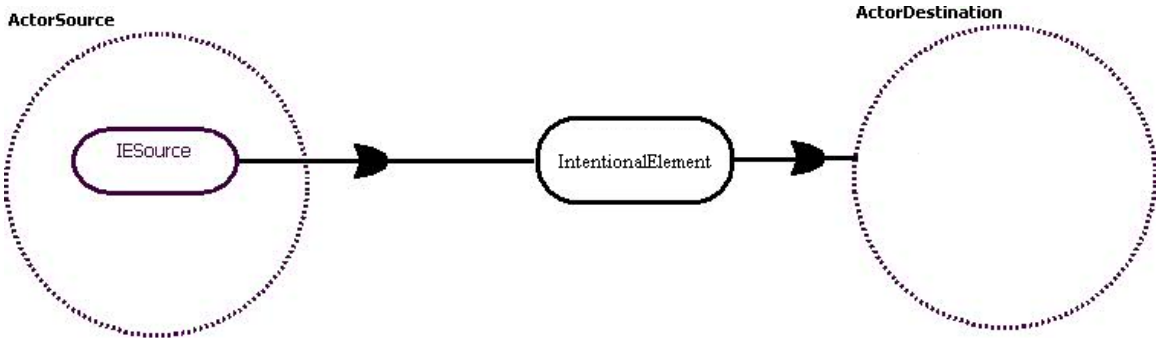

Figure 40 Dependency Scenario 3

- [sourceGRL<<IntentionalElement>> (contained by source GRL<<Actor>>) → depends → destination GRL<<IntentionalElement>> (contained by destination GRL<<Actor>>)], meaning that source GRL<<IntentionalElement>> which is contained by the source GRL<<Actor>> depends on the destination GRL <<In-

tentionalElement>>, which is contained by destination GRL<<Actor>>. See Figure 41.

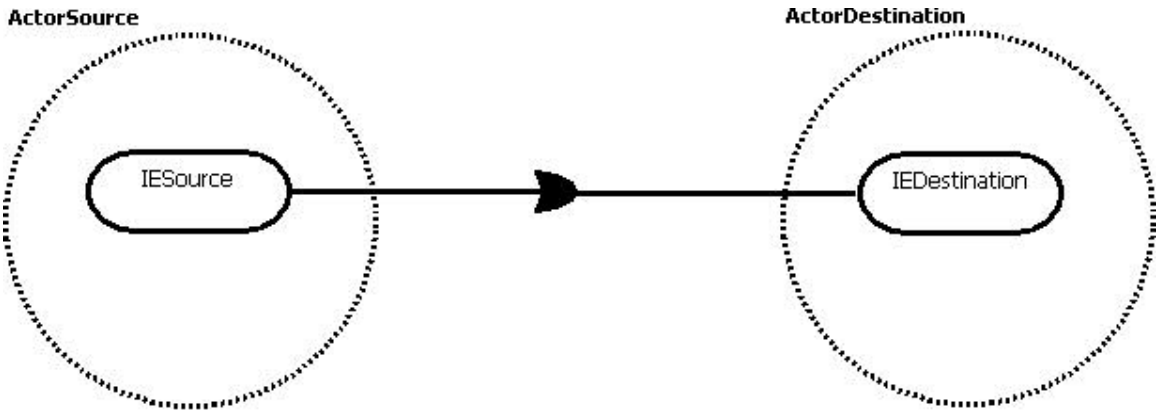

Figure 41 Dependency Scenario 4

Notation

It is as shown in Figure 42:

Figure 42 Dependency

References

- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies)
- UML SS: 7.3.3 Association (from Kernel)

3.3.12 Decomposition

The stereotype Decomposition extends the metaclass Association with multiplicity [0..1]. **Note**: The purpose of a GRL<<ElementLink>> is to define what a source GRL<<IntentionalElement>> requires to be satisfied in order for a target GRL<<IntentionalElement>> to be satisfied. An instance of Association is called a link[3] [23]. It has the same semantics as stereotype Decomposition. The stereotype Decomposition possesses some of its own new attributes and properties.

[3] *Link:* it is instance of UML metaclass Association. All previous occurrences of the term link were interpreted as English language words, not as Associations.

Attributes

Stereotype attributes:

- id: String Defines the identifier of the <<Decomposition>> Association.

Constraints

- <<Decomposition>> Association has an id tag, which must be unique within URN specification.
- Each <<Decomposition>> Association must have a unique name.
- A GRL<<Actor>> can never be the source or the destination of a <<Decomposition>> Association.
- A *belief* is a GRL<<IntentionalElement>> that can never be the source or destination of a <<Decomposition>> Association.

Semantics

<<Decomposition>> Association has different types that are specified by *decompositionType* attribute of GRL<<IntentionalElement>>. Those types are *AND, XOR, IOR.*

By using several types of <<Decomposition>> Association, it is possible to decompose a target GRL<<IntentionalElement>> into many sources GRL<<IntentionalElement>>s as desired. For the decomposition type with value *AND*, it is mandatory that the entire set of source GRL<<IntentionalElement>> for the target GRL<<IntentionalElement>> be satisfied.

<<Decomposition>>Association also enables the description of alternative means for satisfying a target GRL<<IntentionalElement>> (*XOR* for mutually exclusive alternatives, or *IOR* for alternatives that are not mutually exclusive)[17]. One of the source GRL<<IntentionalElement>>s is sufficient for the target GRL<<IntentionalElement>> to be satisfied

Modellers may apply additional constraints according to particular modelling requirements with <<Decomposition>> Association, like restricting only a *Task* as a GRL<<IntentionalElement>> target for a link.

Attribute *name*: String of <<Decomposition>> Association is mapped with name of Class, which is inherited from NamedElement.

Notation

The notation for <<Decomposition>> Association is as shown in Figure 43:

Figure 43 Decomposition

References

- UML SS: 7.3.33 NamedElement (from Kernel, Dependencies)
- UML SS: 7.3.3 Association (from Kernel)

3.3.13 DecompositionType

The stereotype DecompositionType extends the metaclass Enumeration with a multiplicity of [0..1].

Note: The purpose of stereotype DecompositionType is to decompose the GRL<<IntentionalElement>> into one of three types according to the value of its attribute *decompositionType*. These types are *AND*, *XOR* or *IOR.*

Attributes

No attributes.

Constraints

No constraints.

Semantics

<<DecompositionType>> Enumeration is a user defined data type that is mapped with EnumerationLiteral. The literals of <<DecompositionType>> Enumeration are used as value by the attribute decompositionType. This attribute belongs to GRL <<IntentionalElement>> metaclass.

<<DecompositionType>> Enumeration has three user defined values: *AND* means that each of the sub GRL<<IntentionalElement>>s is necessary. *XOR* means one of the sub GRL<<IntentionalElement>>s is enough and only one is selected. *IOR* means one of the sub GRL<<IntentionalElement>>s is enough, but many may be selected.

These values are used by attribute *decompositionType* of <<IntentionalElement>> Class.

Notation

No notation.

References

- UML SS: 7.3.16 Enumeration (from Kernel).
- UML SS: 7.3.17 EnumerationLiteral (from Kernel).

3.4. Global Overview of Profile

The next five figures summarize the stereotypes and extensions to UML present in our profile for GRL and discussed in the previous section.

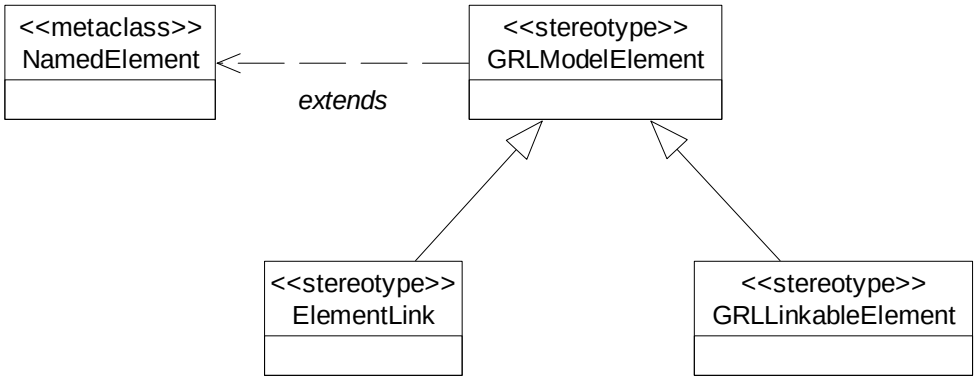

Figure 44 GRL Model Element

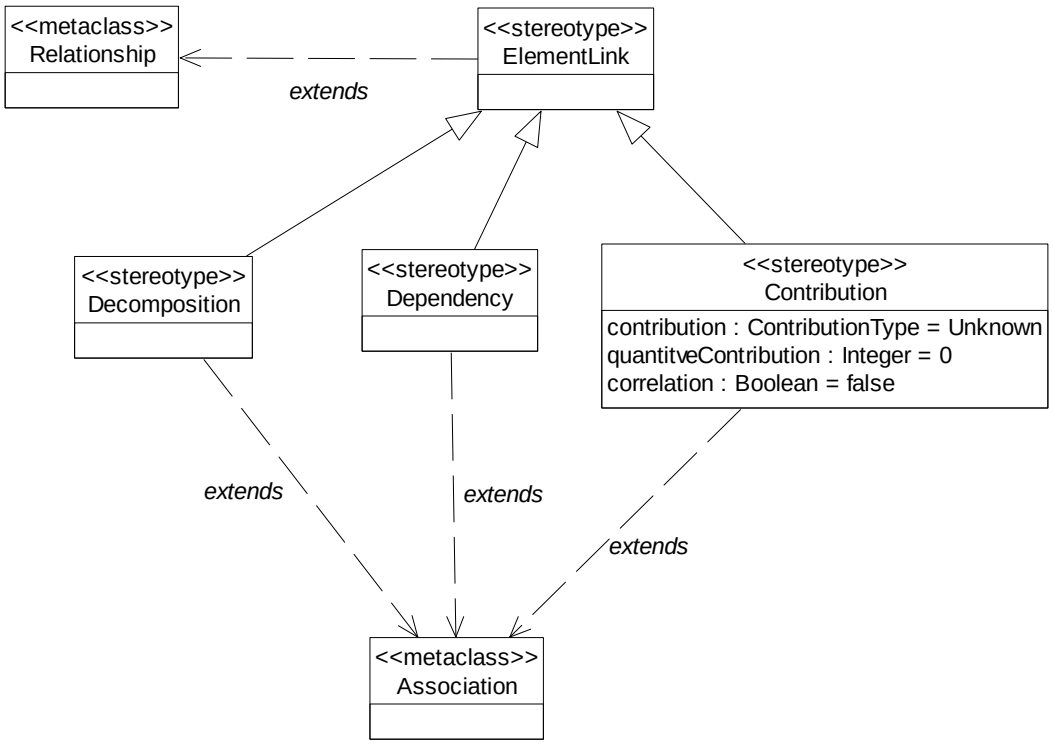

Figure 45 Element Link

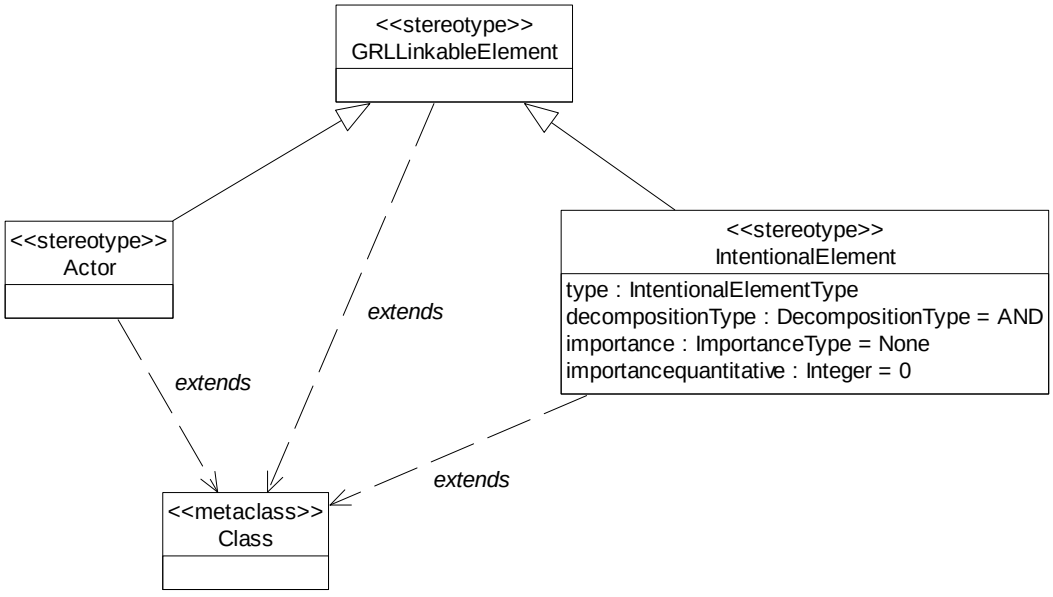

Figure 46 GRL Linkable Element

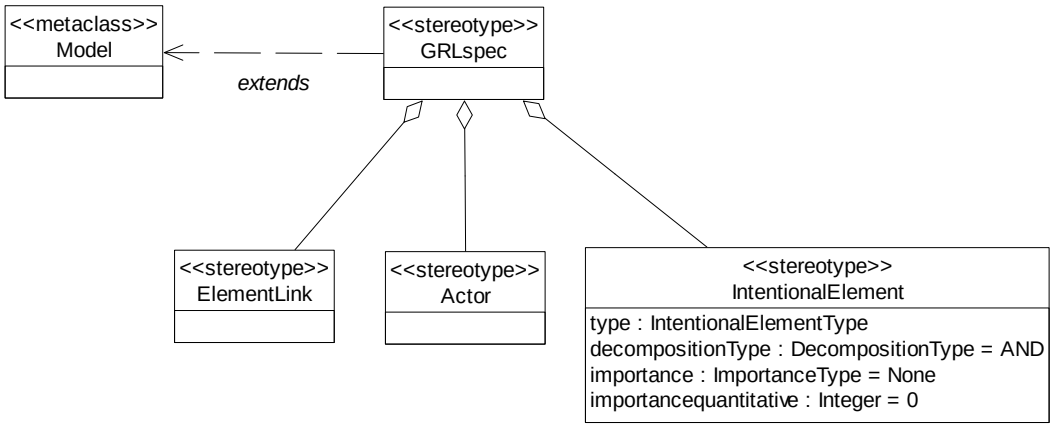

Figure 47 GRL Spec

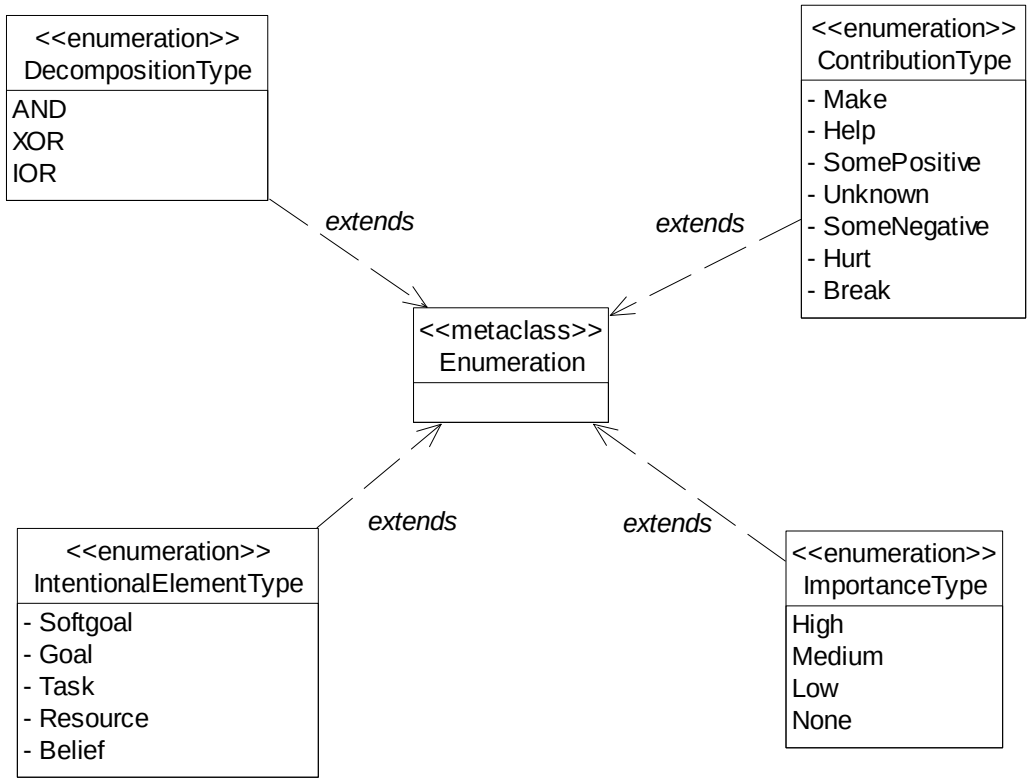

Figure 48 Enumerations

3.5. Chapter Summary

In this chapter, we presented the correspondence of the appropriate UML metaclasses with our GRL metaclasses. All GRL intentional elements are associated with the UML Class metaclass and all ElementLinks are associated with UML Association metaclass. Additionally, GRL Actor is also associated with UML Class metaclass and behaves as a container and a stakeholder. GRL Actor is semantically different from UseCase Actor. Some additional constraints required for GRL elements and inherited constraints from their extended UML metaclasses were also discussed. The next chapter will discuss the implementation of this profile in an industrial-strength UML tool.

Chapter 4. Profile Implementation

This chapter provides a step-by-step implementation of our GRL profile with the tool named Tau G2 4.0 from Telelogic [26] . Section 4.1 gives an introduction to Tau and its features. Section 4.2 discusses Tau's support for profile creation using the Stereotype Mechanism (SM) and the Metamodel Extension Mechanism (MEM). The steps we followed to create the GRL profile are also included. The final section discusses tool limitations and the visual appearance of the profile.

4.1. Introduction to Telelogic Tau G2 4.0

Telelogic (now IBM) Tau G2 version 4.0 [26] was released on February 24, 2008. Tau supports Model Driven Development (MDD) in a UML-based environment. The product is available for all well-known operating systems including Microsoft Windows, Sun Solaris, Redhat Enterprise Linux, and Citrix XPe. The tool has the ability to integrate with Eclipse and Microsoft Visual Studio .NET, as well as to fulfill requirements of domains such as aerospace, defence, enterprise IT, financial services and transportation. Tau fully supports OMG UML 2.1 for systems modelling. It also supports related standards such as the System Modeling Language (SysML 1.0), the Extensible Markup Language (XML) and the UML Testing Profile (U2TP). Figure 49 shows the Tau editor interface.

Tau G2 4.0 supports round-trip engineering of Java, C#, C++, Web Service Definition Language (WSDL), XML Schema Definition (XSD) and Common Object Request Broker Architecture (CORBA). Tau supports automatic bi-directional communication with the Eclipse environment, so the impact of any change in the code can be automatically reflected in the model and vice-versa.

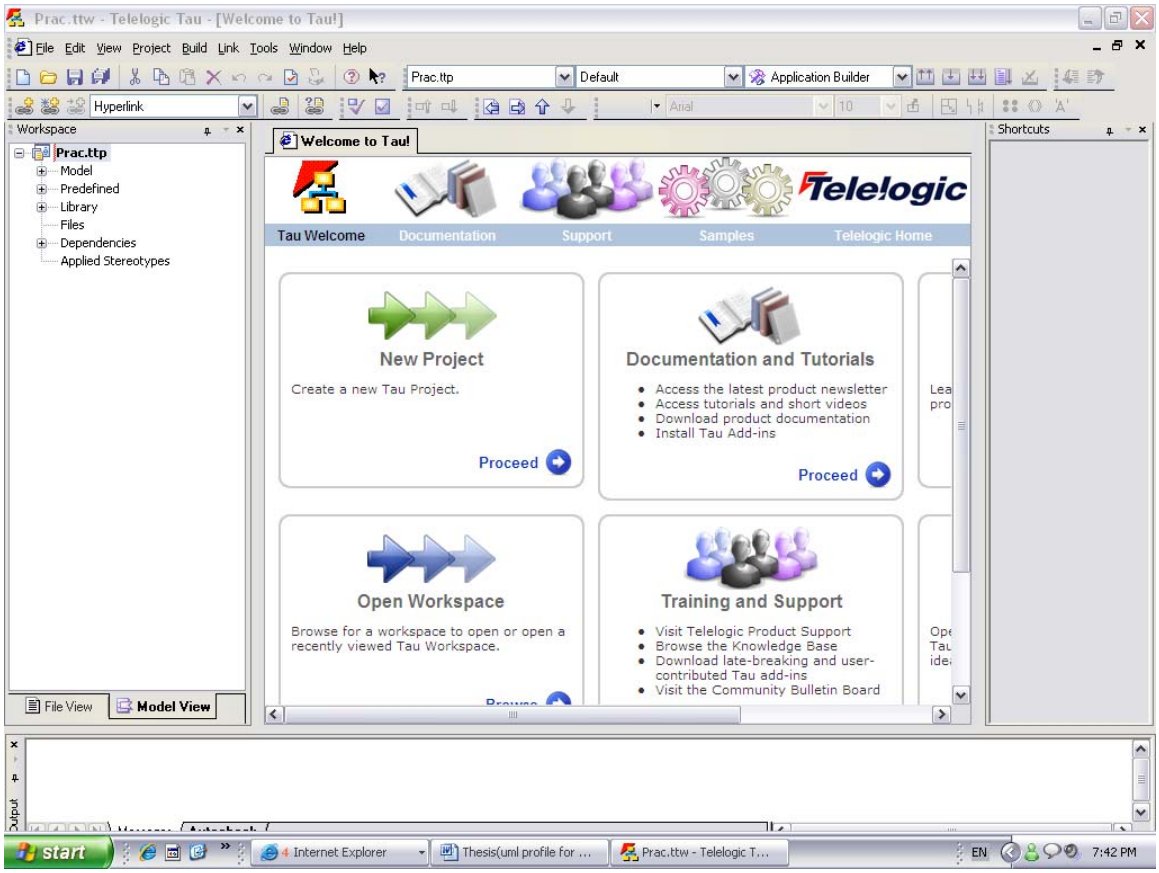

Figure 49 Telelogic Tau G2 4.0 Editor

4.2. Profile Support in Tau G2 4.0

Tau G2 4.0 supports UML profiling. As previously discussed, Tau allows to extend the UML metamodel and, therefore, enables modellers to customize the UML metamodel according to specific domains. Tau supports both the stereotype mechanism and the metamodel extension mechanism discussed in Section 2.5.3. We briefly describe how these two extension mechanisms can be used to support our UML profile for GRL in the next two subsections. The tool has some limitations that are outlined in Section 4.2.4.

4.2.1 Stereotype Mechanism (SM)

As discussed in Chapter 2, the SM is a straightforward way of creating a profile. Some specific steps are required in order to use Tau to design a UML profile with the SM. As

no formal documentation exists that describe these steps, they are included here so others can reproduce our results or create their own profile:

1. Create a directory structure in <Tau Installation>\addins folder.
2. In this folder, create a directory with a profile-appropriate name, like GRLProfile.
3. In this directory, create two sub directories, named “etc” and “script”.
4. Create a script in GRLProfile directory with the same name as the parent directory and the extension .mod. The code of GRLProfile.mod will be:

```
[simplegrlprofile]
"scope" = "PROJECT"
"version" = "1.0"
"longname" = "GRL Profile by Stereotype Mechanism"
"description" = "It enables one to design a model, based on
GRL elements"
"product" = "elvis"
[simplegrlprofile/Bin]
"listBin"= ""
[simplegrlprofile/Script]
"listScript" = ""
[simplegrlprofile/Etc]
"listEtc" = "urn:u2:addins/simplegrlprofile/etc/
simplegrlprofile/simplegrlprofile.u2".
```

5. Launch Tau G2 4.0 and create a “UML for Modelling” project in “etc” directory. Provide the same name to the project as simplegrlprofile. By right clicking on the project, select the option “stereotype” and check TTDPredefinedStereotypes::profile.
6. Create a class diagram in the model.
7. Collapse the library TTDMetamodel. There is a metaclass *Class*. Drag and drop *Class* into the Class diagram.
8. Create a stereotype in the Class diagram by Tau tool palette. Assign Softgoal as name to the stereotype.
9. Provide an extension link from the stereotype to TTDMetamodel::Class. A multiplicity is mandatory in the extension link. This method allows for the creation of all stereotypes and their TTDMetamodel metaclasses that they extend.
10. Create the remaining stereotypes for the profile. These stereotypes represent the GRL elements that are added to the profile.

At this point, the profile is created. For the activation and usability of this profile, we have to follow these additional steps:

11. Open a new instance of Tau. Create a project Prac (or any other name).

12. Create a Class diagram and add classes to it. Right click on *Class*, select stereotype, and apply simplegrlprofile::Softgoal, which will inherit all of the Softgoal properties.

13. Repeat this action for the other stereotypes.

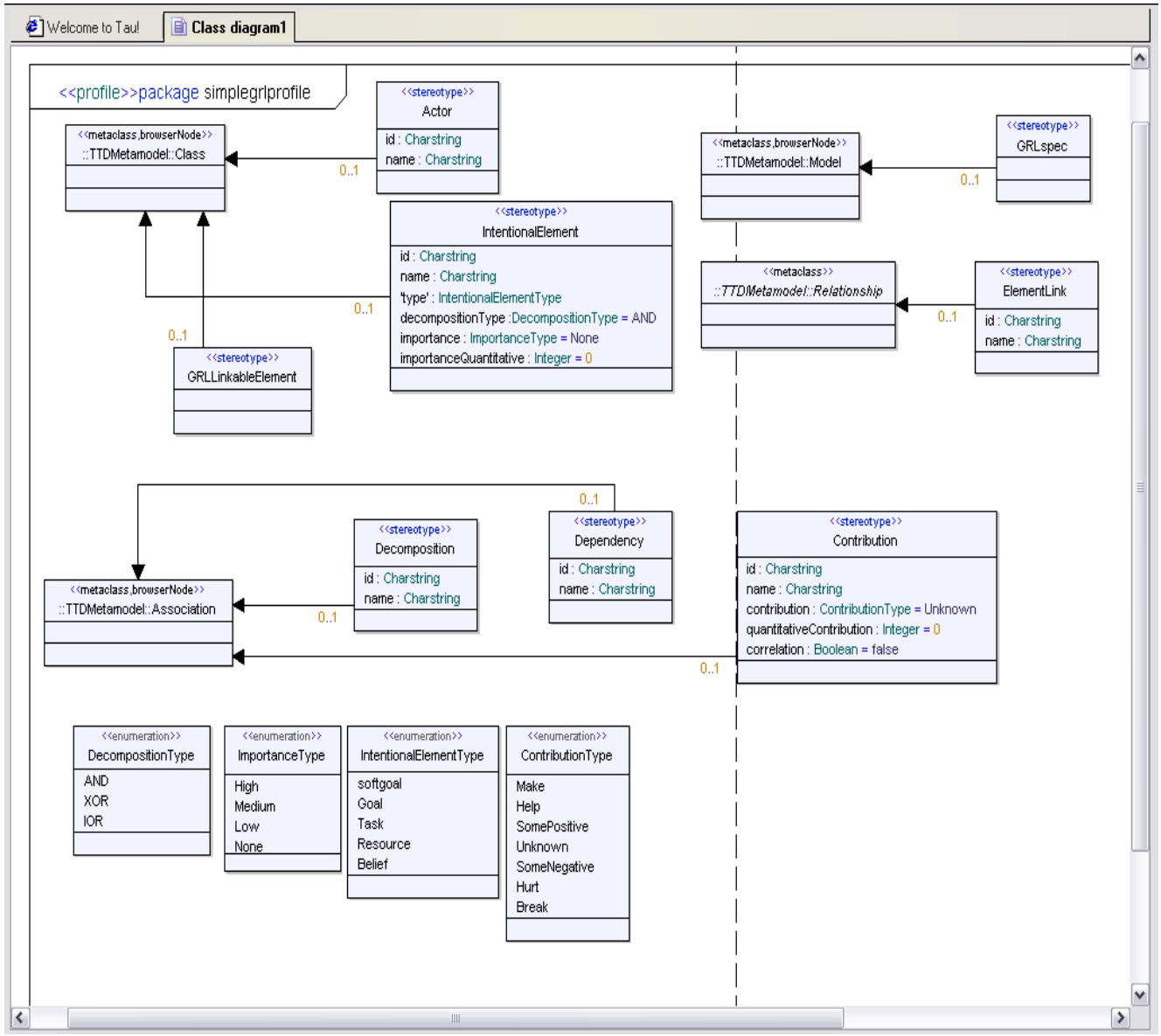

Figure 50 GRL Profile by Stereotype Mechanism (1)

Figure 50 shows the stereotypes that represent the GRL metamodel. The modeller can use these stereotypes to design a system with a goal-oriented modelling view.

4.2.2 Metamodel Extension Mechanism (MEM)

Metamodel Extension Mechanism [29] is a more complicated but more complete way of profiling UML. The support for MEM profiling was one of the reasons for the selection of Tau as a target tool in this thesis. However, we realized after some experiments with the tool that Tau does *not* fully support MEM profiling. Additionally, only very basic documentation and limited support is provided by Tau for this type of profiling.

The following are the steps for creating a profile using this mechanism:

1. Install the FIDebugger, which is part of TAU G2 SDK. A UML entity is by default provided with a unique, randomly generated identifier called a Globally Unique Identifier (GUID). A GUID remains unchanged for the entire lifetime of an entity. FIDebugger is used to read that GUID for any GRL element residing in the GRL profile.
2. Create a directory structure in <Tau Installation>\addins folder.
3. In this folder, create a directory with a profile-appropriate name, like GRLProfile.
4. In this directory, create two sub directories, named “etc” and “script”.
5. Create a script in GRLProfile directory with the same name as the parent directory and the extension .mod. Some changes have occurred in the mod file, which will cause the script to looks like the following:

```
[GRLProfile]
"scope" = "PROJECT"
"version" = "1.0"
"longname" = "GRL Profile by Metamodel Extension Mechanism"
"description" = "Enables one to design a model based on GRL
elements and creates a GRL editor with its own tool palette."
"product" = "elvis"
[GRLProfile/Bin]
"listBin"= ""
[GRLProfile/Script]
"listScript" = "load.tcl"
[GRLProfile/Etc]
"listEtc" = "urn:u2:addins/GRLProfile/etc/GRLProfile/
GRLProfile.u2"
```

6. Create a project with the same name as the parent directory name and the mod file name (GRLProfile). Select TTDPredefinedStereotypes::profile.
7. In the main GRLProfile Package, create four additional sub packages:

   a. GRL Model

   This package contains all metaclasses used for GRL model creation. Pass the GRL information to the package. See Figure 51- Figure 54 for a description.

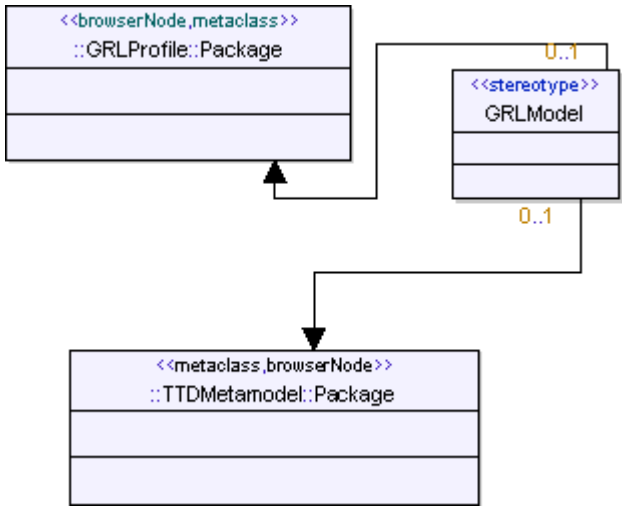

Figure 51 GRL Model Package (1)

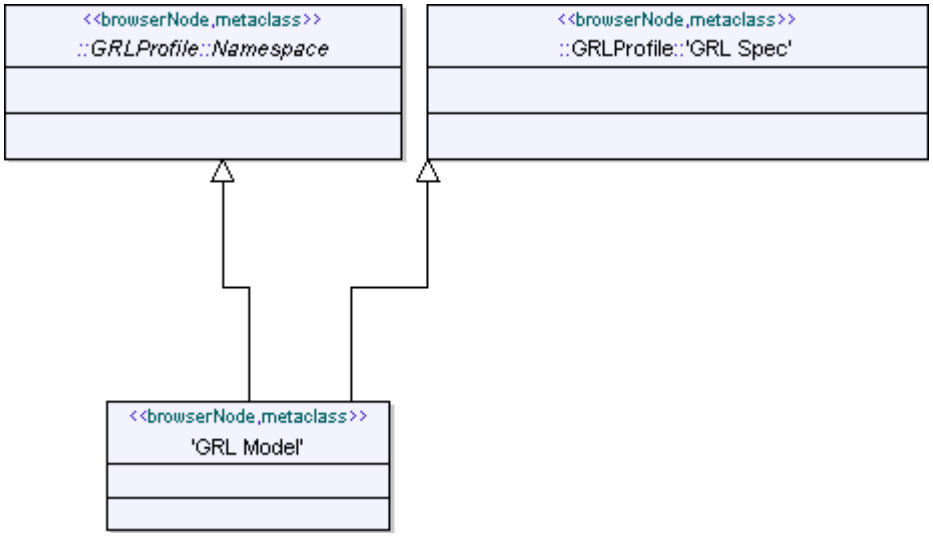

Figure 52 GRL Model Package (2)

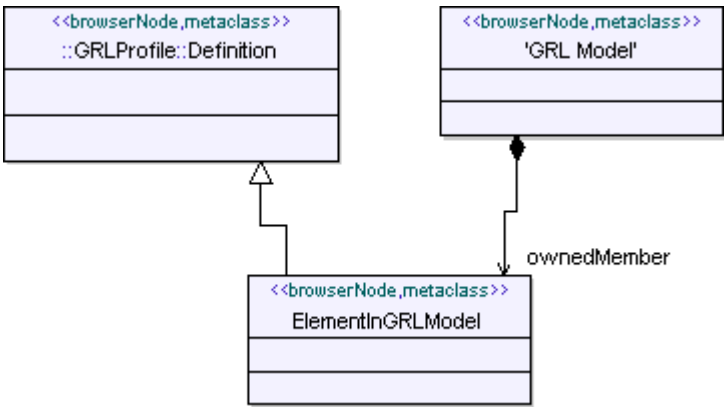

Figure 53 GRL Model Package (3)

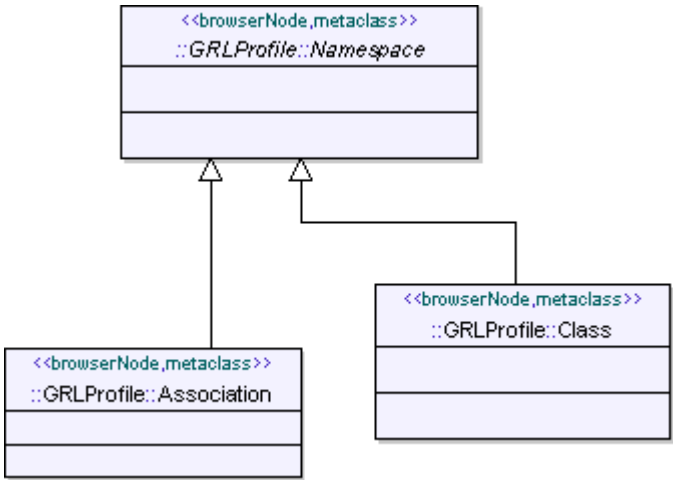

Figure 54 GRL Model Package (4)

b. GRL Editor

This package contains all of the information necessary to create a GRL editor, to specify which information can be kept by the editor, as well as to whom this information can be passed to. The package diagrams are depicted in Figure 55 - Figure 58.

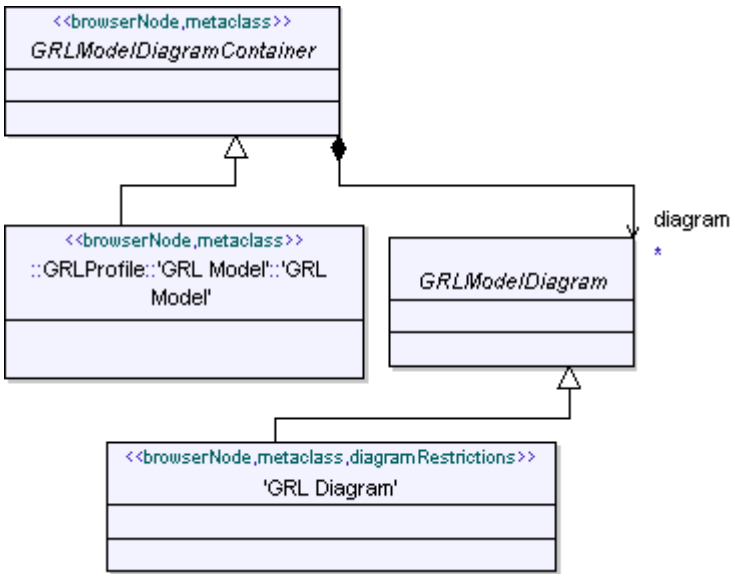

Figure 55 GRL Editor Package (1)

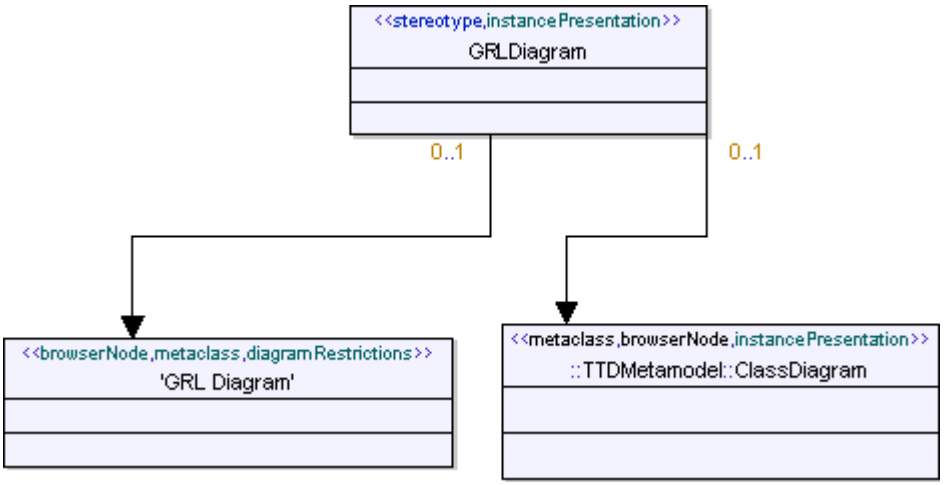

Figure 56 GRL Editor Package (2)

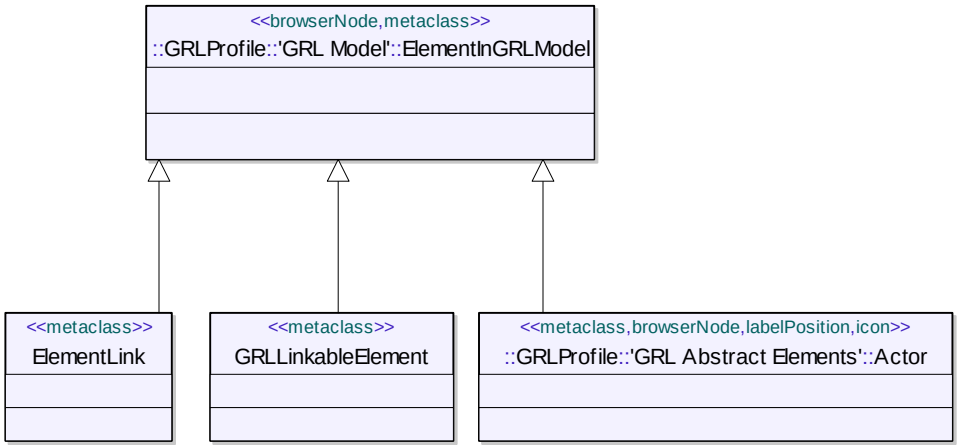

Figure 57 GRL Editor Package (3)

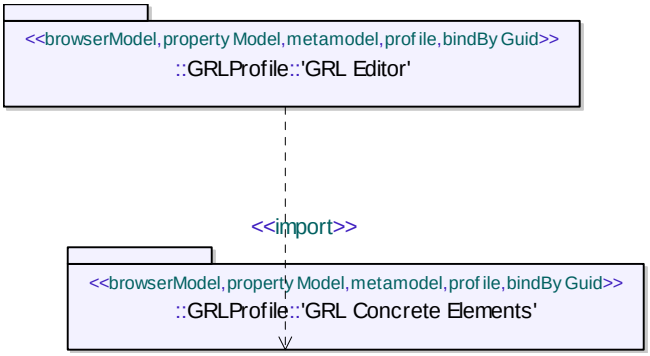

Figure 58 GRL Editor Package (4)

c. GRL Concrete Elements

This package shows the metaclasses created for GRL elements. These meta-classes describe the actual GRL constructs. Figure 59 and Figure 60 show the content of this package.

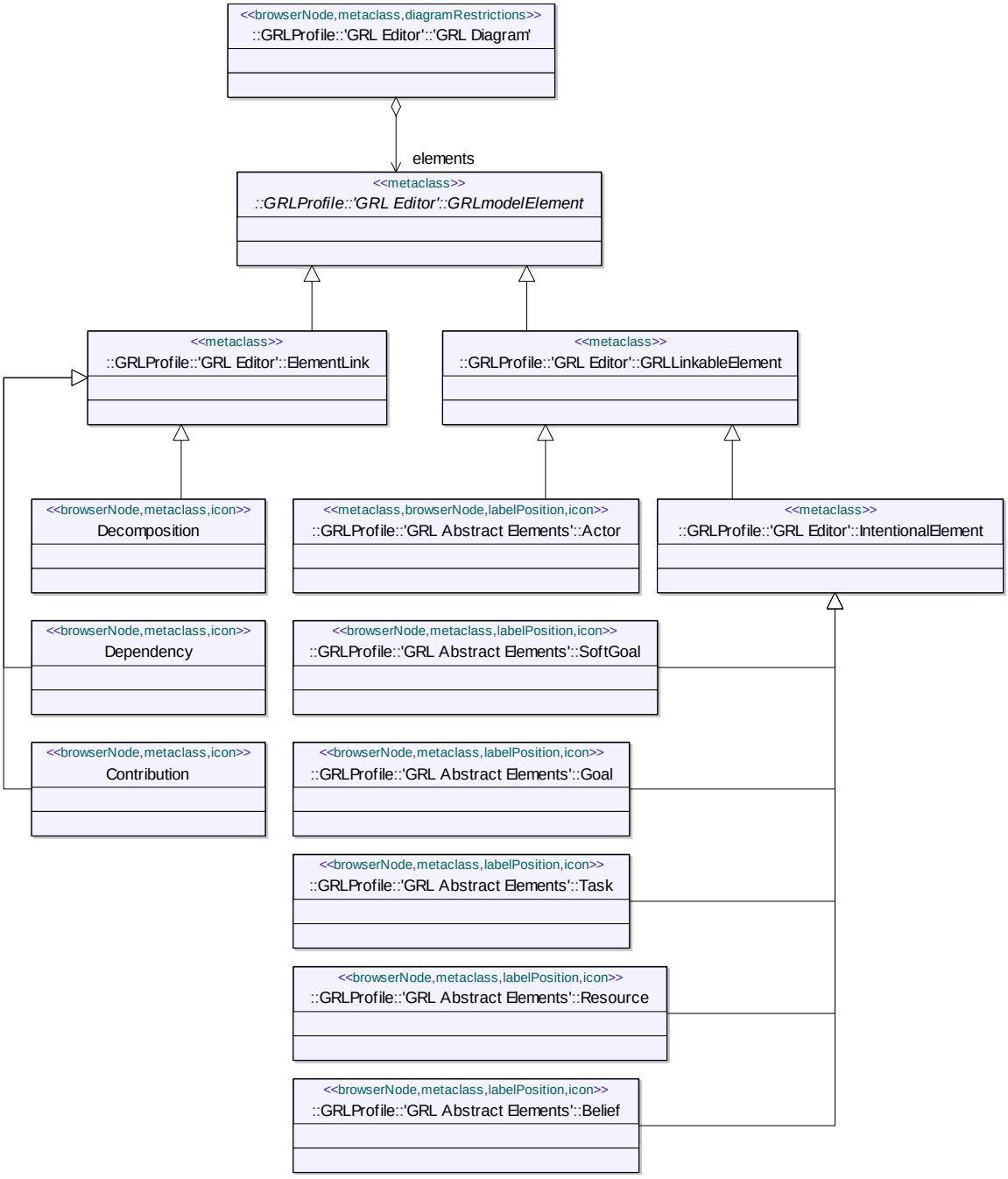

Figure 59 GRL Concrete Elements Package (1)

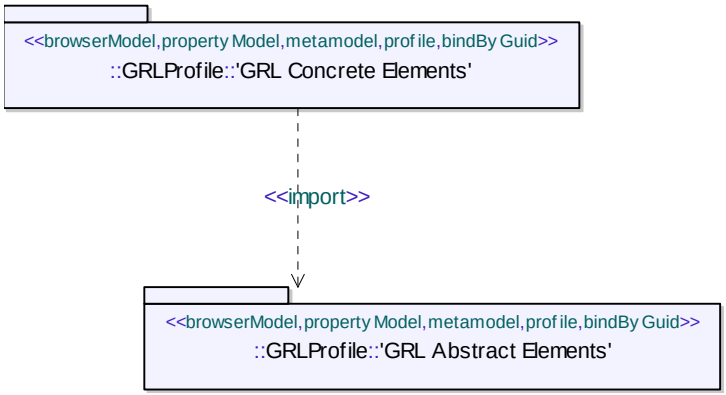

Figure 60 GRL Concrete Elements Package (2)

d. GRL Abstract elements

This package has all of the stereotypes that represent the GRL profile elements. The package diagram is depicted in Figure 61 and Figure 62.

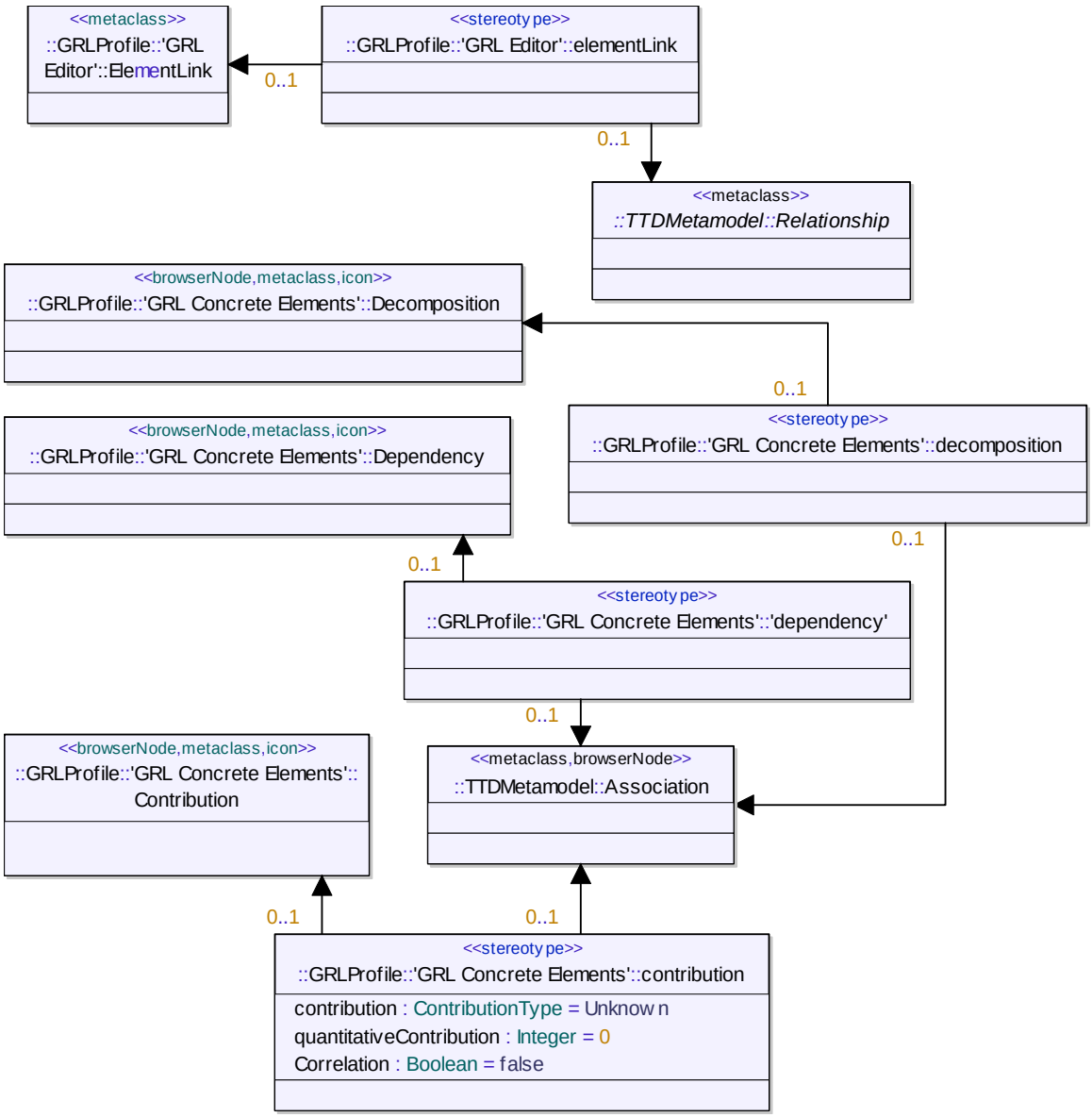

Figure 61 GRL Abstract Elements Package (1)

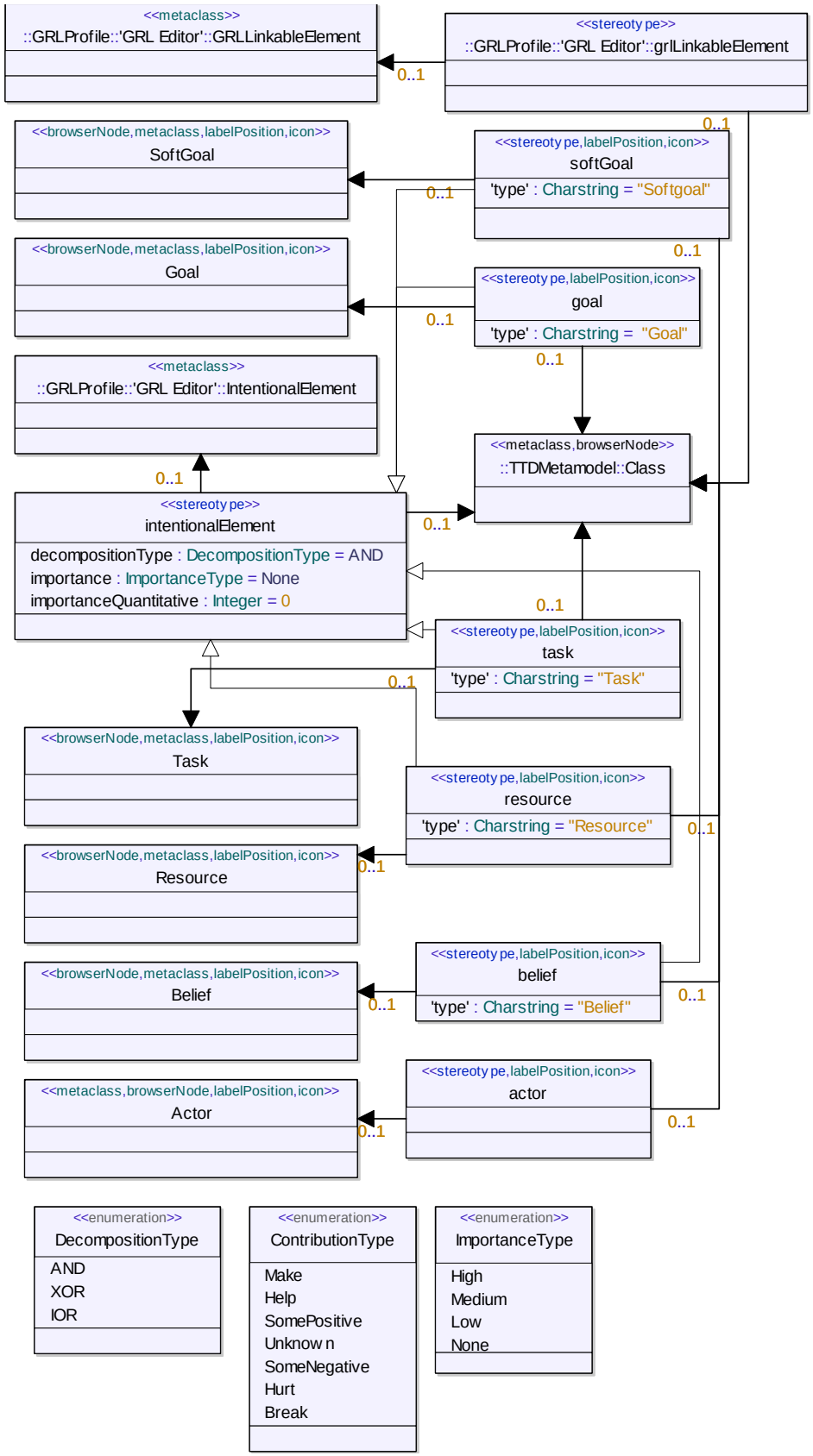

Figure 62 GRL Abstract Elements Package (2)

All four packages extend the following predefined stereotypes, which will be explained in the next section:

- TTDExtensionManagement::browserModel
- TTDExtensionManagement::propertyModel
- TTDPredefinedStereotypes::metamodel
- TTDPredefinedStereotypes::profile
- TTDPredefinedStereotypes::bindByGuid

Classes and stereotypes use these predefined stereotypes with some addition from the above mentioned:

- TTDPredefinedStereotypes::metaclass
- TTDStereotypesDetails::icon
- TTDStereotypesDetails::labelPosition
- TTDExtensionManagement::diagramRestrictions
- TTDExtensionManagement::instancePresentation

GRLProfile is the main package and represents the GRL core. It has all metaclasses that are not included in the above mentioned packages. Additionally, a number of properties are required and can be made visible by simply right clicking on any element and selecting the properties option. The package GRL Model and the GRL Editor have constructs which require extension of the GRL elements with the metamodel of TAU.

8. By launching the debugger, retrieve the GUID of the GRL profile and create a Tool Command Language (TCL) script to launch the GRL profile. This TCL file (load.tcl) should be located in the script directory which was produced when the directory architecture was first created. The text file will have the following information:

```
package require commands
# Returns the session of the active project. Should be used whenever
# the session is needed.
proc GetActiveSession {} {
            set activeProject [std::GetActiveProject]
            return [lindex [std::GetModels -kind U2 -project
                            $activeProject] 0]
}
set ProfilePath [file join [std::GetInstallationDirectory]
                 addins/GRLProfile/etc/GRLProfile/GRLProfile.u2]
output "Loading GRLProfile ..."
u2::LoadLibrary $ProfilePath
```

```
set GRLMetaModel [u2::FindByGuid [GetActiveSession]
                          "QlhHIIuL3KVLJ89hjVuJ76RI"]
if {$GRLMetaModel != 0} {
                RegisterMetaModel $GRLMetaModel
                u2::SelectMetaModel $GRLMetaModel
}
output "Done.\n"
```

4.2.3 Predefined Stereotypes Description

Regarding the metamodel extension mechanism, there was a need to use predefined tool stereotypes to obtain advanced profile functionalities. These predefined stereotypes are listed below:

TTDExtensionManagement::browserModel
This stereotype is used for the application of metaclasses in metamodels and determines the availability of nodes in the view.

TTDExtensionManagement::propertyModel
This stereotype is intended for the application of metamodels and determines the presence of a property view for the particular metamodel.

TTDPredefinedStereotypes::metamodel
This stereotype specifies that a package contains a representation of a metamodel; the package typically contains classes that represent metaclasses in that metamodel.

TTDPredefinedStereotypes::profile
A stereotype profile is a package that is used for model extensibility and typically contains a set of stereotypes that may be applied to elements in a model.

TTDPredefinedStereotypes::bindByGuid
This stereotype specifies that contained references shall exclusively be bound by GUID.

TTDPredefinedStereotypes::metaclass
This stereotype specifies that a class represents a metaclass.

TTDStereotypesDetails::icon
This stereotype is used to associate an icon with the graphical appearance of a symbol.

TTDStereotypesDetails::labelPosition

This stereotype determines the position of the label (vertical or horizontal).

TTDExtensionManagement::diagramRestrictions

This stereotype is applied to a metaclass in a diagram, in order to disable the use of a text or comment symbol in that diagram.

TTDExtensionManagement::instancePresentation

This stereotype describes how instances of a stereotype should be presented by the properties editor; applying this stereotype as the default presentation enables it to be customized.

4.2.4 Limitations of the Tool

Although Telelogic Tau G2 4.0 supports UML profile creation, there exist several limitations. The tool does not possess any construct or mechanism by which an Actor boundary can be created. Furthermore, none of the profile creation mechanisms (i.e. Stereotype Mechanism and Metamodel Extension Mechanism) supports the custom appearance of GRL links. The tool, also, does not support all of the UML metamodel classes. At the time of the GRL profile modelling, we found that the tool did not support the Enumeration metaclass and the NamedElement metaclass. It is not apparent whether there are other non-supported metaclasses, as our scope was limited. The tool forces the use of its own custom enumeration construct. Thus, we did not find a way to extend our user defined data types with the UML Enumeration metaclass. Moreover, we were forced to deviate from the original GRL metamodel, since the tool does not provide the NamedElement UML metaclass. Consequently, we used the Definition[4] metaclass as a replacement. The GRL metamodel does not have a Package to encapsulate all GRL construct, but it is limited by the tool that uses Package for accumulating and containing the GRL profile constructs.

The tool does not support the association of customized multiple icons with Enumeration literals. As a result, we were forced to deviate from the original GRL metamodel and we instead generalized the IntentionalElement class into five subclasses

[4] *Definition* is a metaclass defined in Tau Metamodel.

called: Softgoal, Goal, Task, Resource and Belief. This generalization allowed us to assign customized icons to the classes.

Another limitation is caused by the stereotypes that are associated with metaclasses other than the Class metaclass. These stereotypes are neither applied nor selected automatically by the tool at runtime. Consequently, this made us unable to automatically implement the functionality of the Associations and led us to manually assign the appropriate stereotype. However, we did not face this problem when dealing with the other stereotypes associated with the Class metaclass.

The tool has a number of views to show the different aspects of the model. These views include a standard view and a diagram view. When a UML diagram and GRL diagram are created together, the diagram view reveals all of the diagrams but not the customized tool palette for the GRL diagram. Only the Class diagram tool palette is shown, as the GRL diagram is based on the Class diagram. Conversely, the standard view only shows the GRL diagram and its tool palette, but no other types of UML diagrams.

After discussion with the Telelogic support staff, it was concluded that some limitations were due to a bug in the tool.

4.3. Profile-Based GRL Editor

The two approaches discussed above were used to create GRL profiles and models based on these profiles. The following is a discussion on GRL model edition based on the two mechanisms.

4.3.1 Stereotype Mechanism

This approach does not provide a separate GRL editor. We must activate the GRL profile and then create a Class diagram. In the Class diagram editor, we create classes and manually associate an appropriate stereotype to each of these classes. A property view enables the setting of attribute values for the elements. The resulting model appears as in Figure 63. This approach significantly increases the probability of mixing UML elements with GRL elements in undesirable ways.

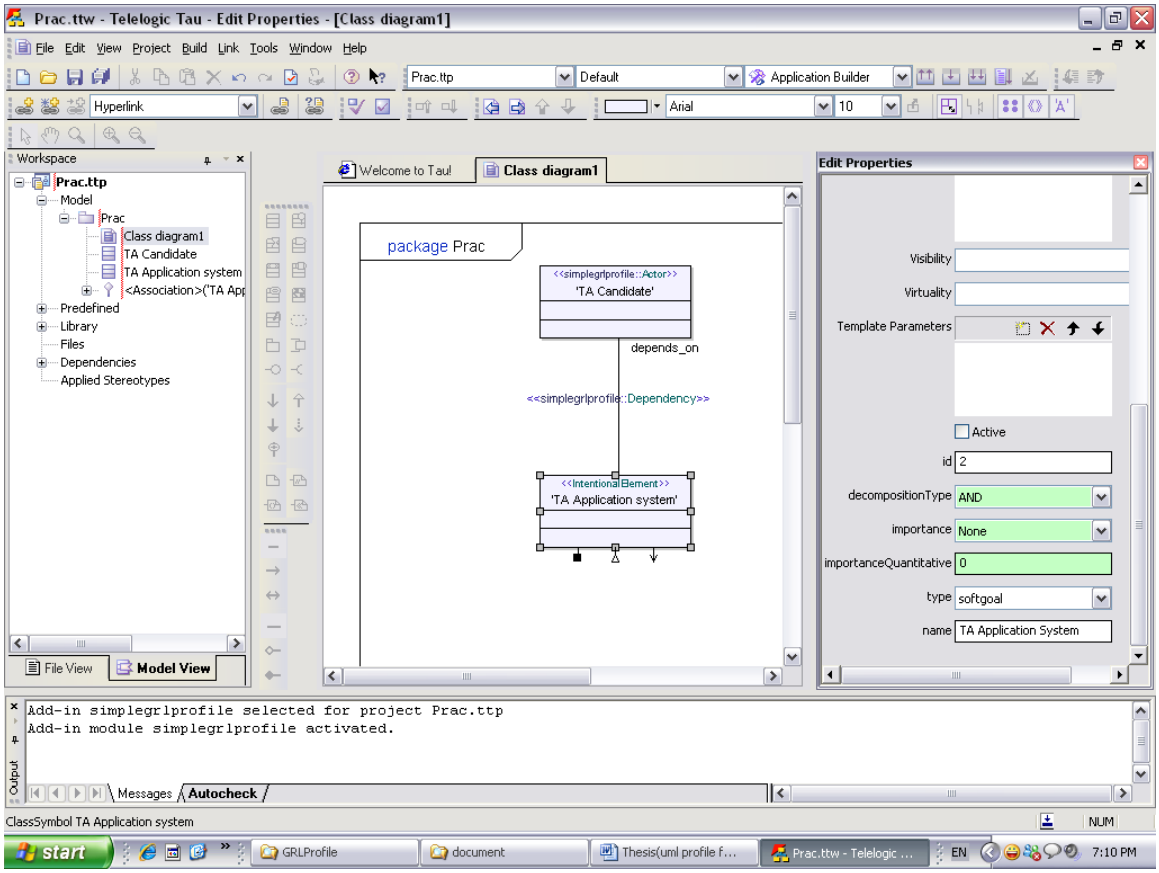

Figure 63 Stereotype Profile View

4.3.2 Metamodel Extension Mechanism

This approach allows obtaining a specific GRL editor with a customized tool palette for GRL constructs. It is possible to “drag and drop” elements in the GRL editor to create a GRL model. The user is able to create a model based solely on GRL elements with little chance of diagram pollution. Again, a property view can be used to provide values to attributes without visual representation. The resulting diagram appears as in Figure 64.

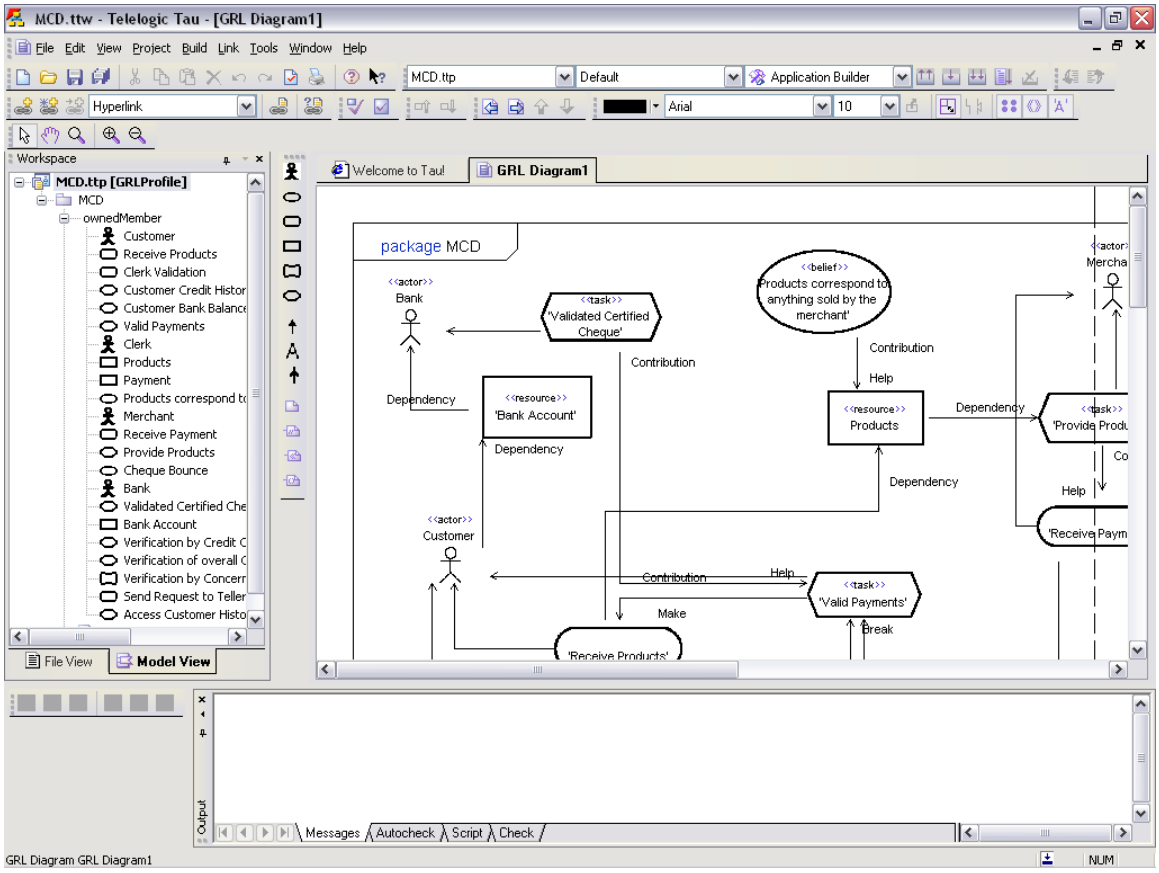

Figure 64 GRL Editor View

4.4. Chapter Summary

In this chapter we have used Telelogic Tau G2 4.0 for implementing our GRL profile. We used both the Stereotype Mechanism and the Metamodel Extension Mechanism, and the steps required to create such profiles were defined and illustrated. We faced some tool limitations that forced alterations to the implemented GRL metamodel. The major limitation was the lack of support of the Association metaclass in the UML profile.

Chapter 5. Experiments and Evaluation

In this chapter we use goal models created with a specialized GRL tool (jUCMNav [19]) and model them again with the help of our GRL profile created using the Metamodel Extension Mechanism in TAU. “jUCMNav [24] is a graphical editor and an analysis and transformation tool for the User Requirements Notation (URN). URN is intended for the elicitation, analysis, specification and validation of requirements. URN combines two complementary views: one for goals provided by the Goal-oriented Requirement Language (GRL) and one for scenarios provided by the Use Case Map (UCM) notation” [19]. A snapshot of this editor is shown in Figure 65.

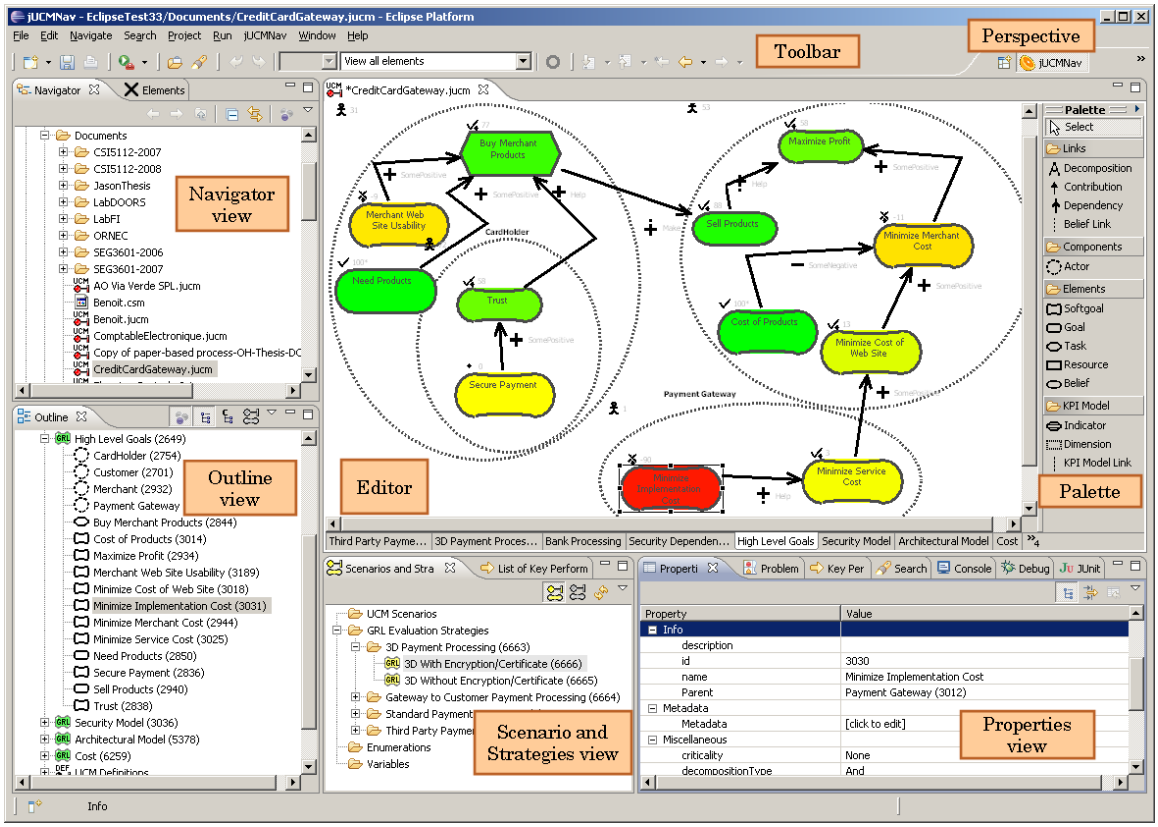

Figure 65 jUCMNav Editor View ([19])

We attempt to validate our profile through the creation of a sample model. We will compare an implementation of the model using both jUCMNav and Tau with the GRL profile. This example is taken from software engineering course assignment given by D. Amyot (with his permission) in 2007. The application model is a university Teaching Assistants (TAs) allocation system. This system models the concerns of various stakeholders for an upcoming system used to allocate TAs to courses in a University.

There are two alternatives considered for TA candidates to apply for courses: Paper-based forms and Web-based forms. There are also two alternatives for allocating candidates to courses: 1) fast and cheap allocation based completely on the candidate-provided information, and 2) candidates are also interviewed to ensure minimal competencies. This system is limited to four actors with several goals, tasks, softgoals and beliefs. These actors are:

TA Candidate: Their expectation from the system is to be secure and flexible. According to them, the system should allow one to apply to many courses at a time in a secure way. They do not want to enter the same information repeatedly and also do not want to submit forms by physical means at a particular office. The following figure shows that TA candidates can apply using a paper-based application or web-based application. With the former, TA Candidate cannot easily apply to many courses at once and there are security concerns. The latter enables the TA Candidate to apply to many courses at once by a secure mean. Candidate may or may not be interviewed with either means of application.

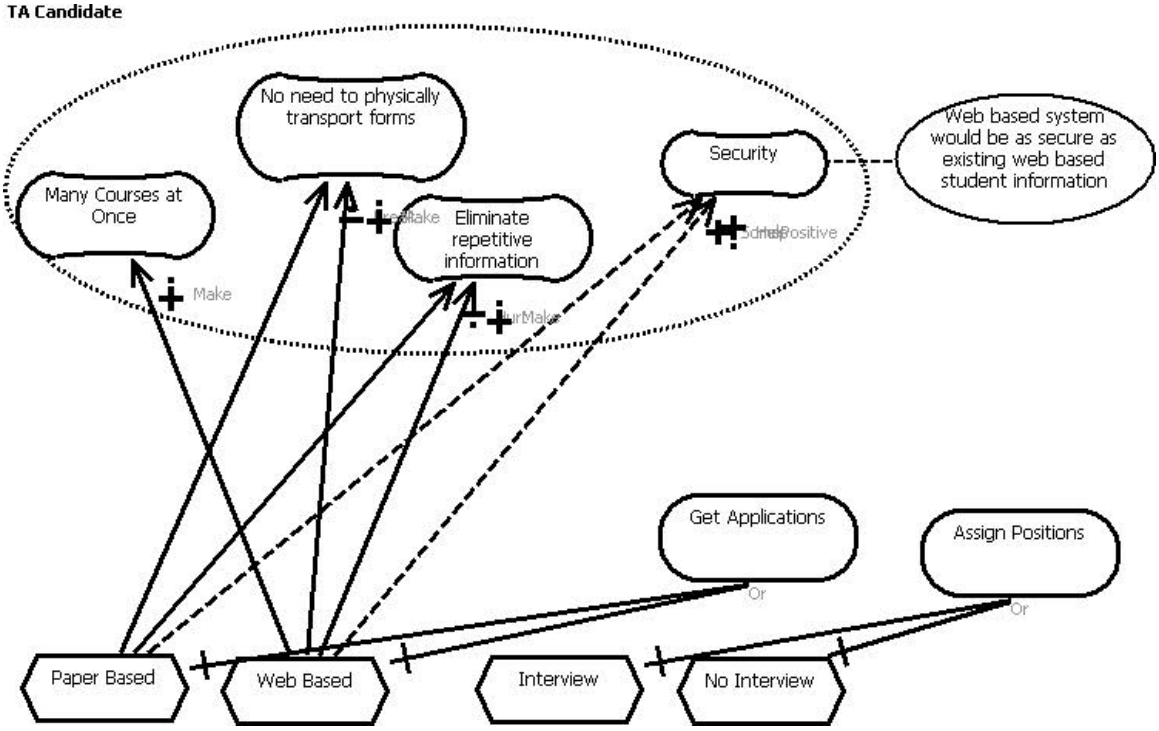

Figure 66 TA Candidate, Modelled with jUCMNav

Administration: The University administration wants the system to be produced and operated at a very low cost and enormously wants to avoid objections and grievances from candidates. The following figure shows that the administration may or may not interview candidates during the selection process. If the administration follows the process of interviewing each candidate, then there is a possibility of discrimination. This also affects the effort of maximizing the number of selected TAs and increases the cost of the selection process. On the other hand, if the administration does not do any interview, this decreases the selection process cost, helps to select a maximum number of TAs, and decreases the possibility of discrimination.

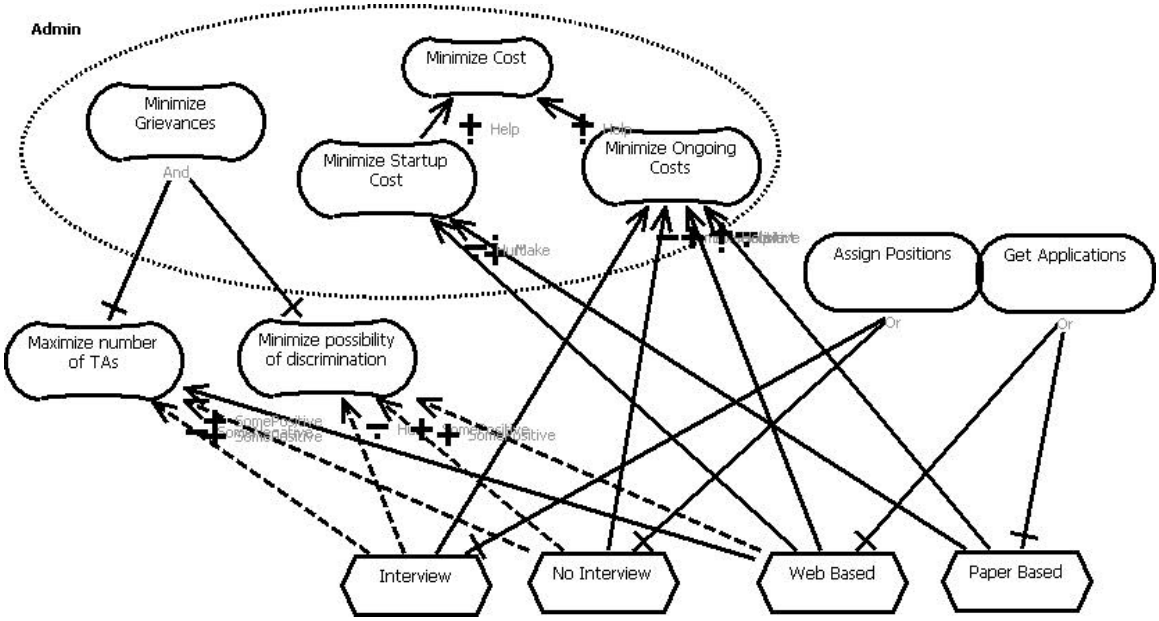

Figure 67 Admin, Modelled with jUCMNav

Students: Students want a competent TA for their labs and tutorials. They also desire to get TAs as early as possible at the beginning of the semester. The following figure shows that by using interviews, students get a competent TA but the process may take more time. Without interviews, students may get TAs earlier but the competency will not be guaranteed.

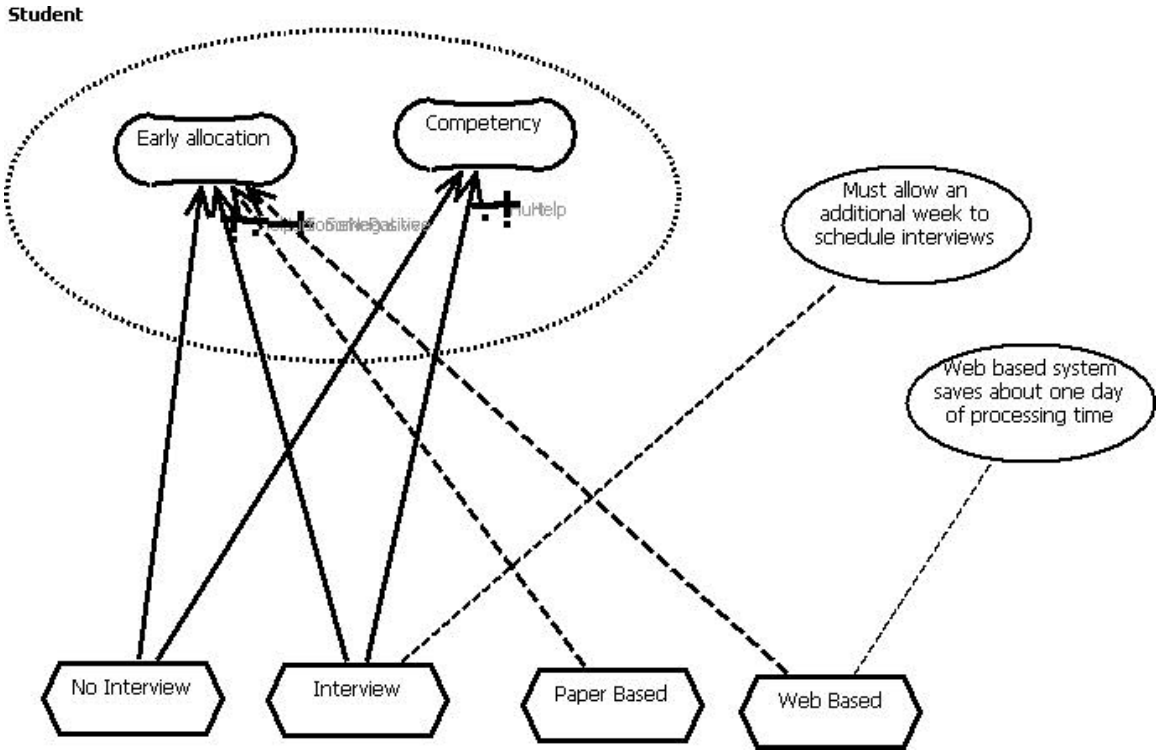

Figure 68 Student as Designed in jUCMNav

TA Union: The TA Union wants a maximum number of candidates to be able to access the system and fill the forms, and that a maximum number of appointments of TA position become possible.

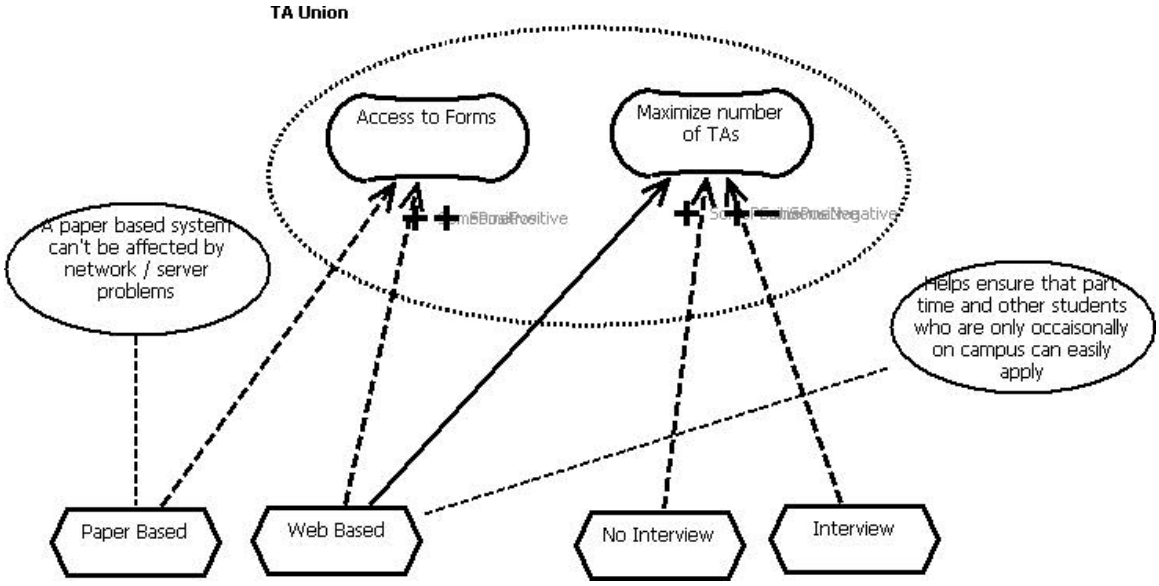

Figure 69 TA Union, Modelled with jUCMNav

This example will reveal several limitations of Tau, as previously discussed in Section 4.2.4. The nature and correctness of the modelling of the above actors and intentions is not a concern here; this is only an example that uses many elements of the notation.

Another Example considers is a Merchants and Customer Dependencies (with the jUCMNav Editor): The merchant-customer dependency system has almost all the possible usage scenarios of GRL constructs. The model includes four actors: Customer, Clerk, Bank and Merchant. Actor Customer receives products from actor Merchant. In response, actor Customer sends a Payment to actor Merchant. Payment should be valid. Its validity can be checked by actors Clerk and Bank. An effort was made in Figure 70 to cover a maximum number of GRL constructs. Both considered example's diagrams are aimed at showing the completeness of our work. They include the GRL constructs of goal, softgoal, task, belief, resource, dependency, contribution, correlation, decomposition and actor.

Actor Customer contains intentional elements as well as another actor with its own intentional elements. It has dependency links with intentional elements residing outside of its boundary. Intentional elements inside its boundary have contribution links, dependency links, and "OR" decomposition links. The actor also has dependencies with other actors and intentional elements.

Actor Clerk is placed inside the boundary of actor Customer (the customer here is likely a large organization). It owns intentional elements and has contribution links with other intentional elements owned by actor Customer. Actor Clerk's intentional elements have "And" and "Or" decomposition links as well as contribution links with standalone intentional elements.

Actor Bank is dependent on actor Customer for specifying a Bank Account during transactions. It resides outside of Customer's boundary.

Actor Merchant is also a standalone actor. It does not reside in any other actor boundary and it also has its own intentional elements, which depend on other intentional elements outside the boundary of the actor.

Standalone intentional elements have correlation links, dependency links and contribution links. There is a belief “Cheque Bounces” that does not belong to any actor and has a correlation link with an intentional element owned by actor Customer

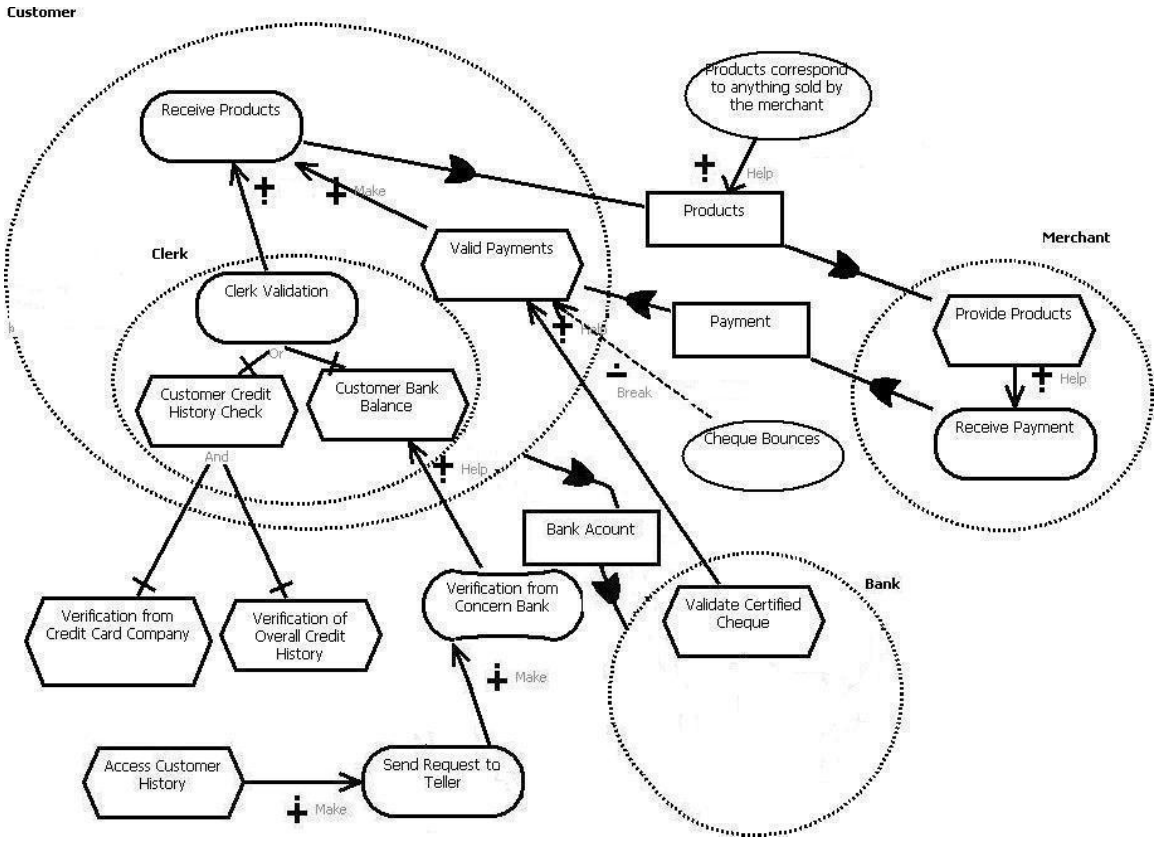

Figure 70 Merchant and Customer Dependencies, Modelled with jUCMNav

The goal of this exercise is to demonstrate the profile’s level of effectiveness and preciseness in comparison to jUCMNav. Both considered examples will mutually cover all constructs, links and scenarios of GRL. In Section 5.1, we describe the Ta system model created using Telelogic TAU G2 4.0 with the GRL profile extension. Section 5.2 discusses the Merchant and Customer Dependencies model created using Telelogic TAU G2 4.0 with the GRL profile extension and Section 5.3 evaluates the two designs from our perspective.

5.1. The TA System Designed in Tau-GRL Profile

Our GRL model is composed of four diagrams, one per actor. Each diagram created using the profile's implementation will be compared to the equivalent jUCMNav diagram.

TA Candidate

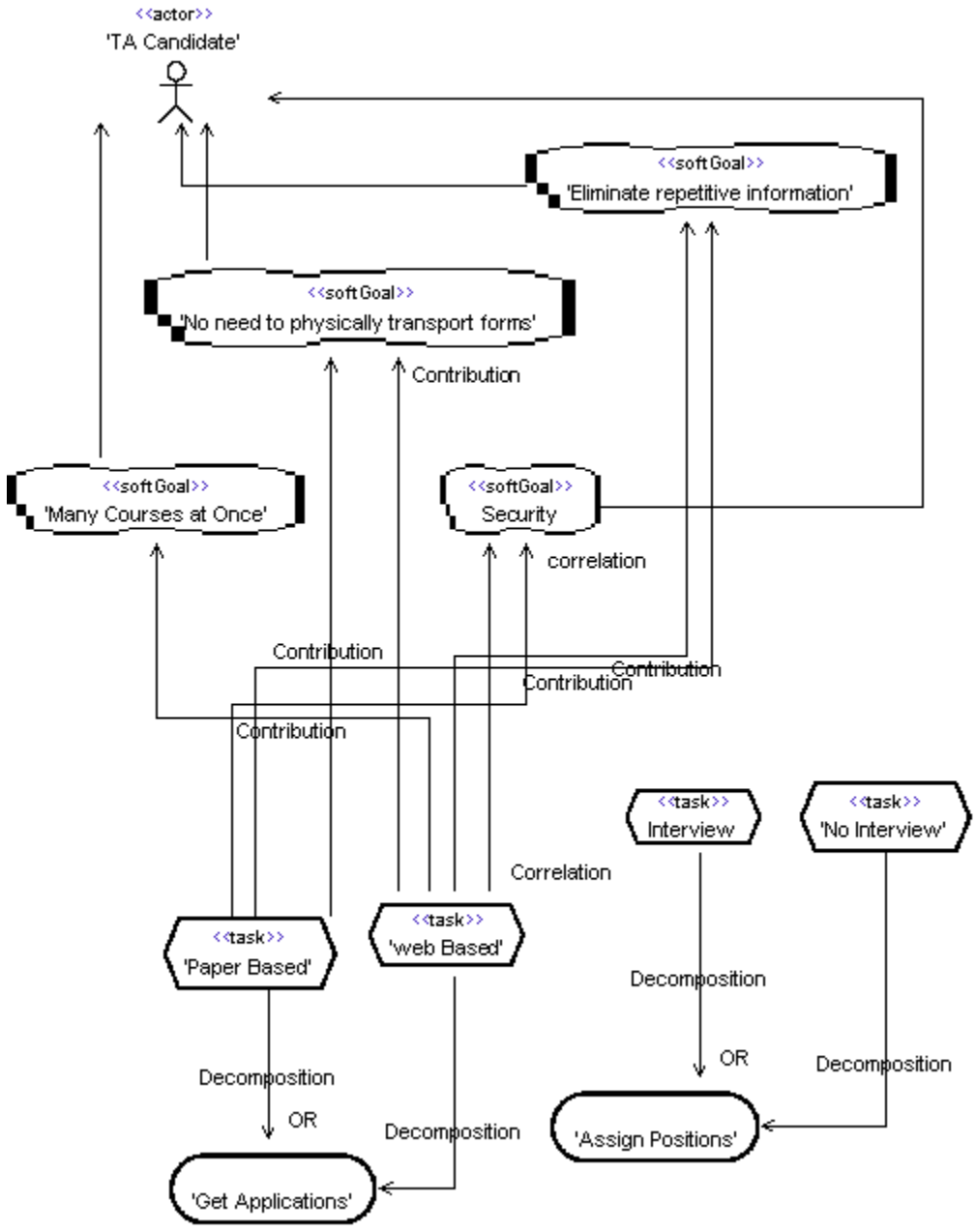

Figure 71 TA Candidate, Modelled with Tau GRL Profile

Figure 66 and Figure 71 represent the first diagram of our GRL model in jUCMNav and Tau-GRL profile, respectively. This is a representation of the TA Candidate actor. The actors, intentional elements and links are captured correctly in our profile. However, by comparing both diagrams, we can see that the actor boundary is not supported by our GRL profile, because of a limitation of Tau, and containment relations have to be shown using associations. Textual and graphical representations of the impact and decomposition type of the different links are also missing in Figure 71 Again; this is because of a lack of support by the tool. We can show a textual representation of these elements only as comments, but they are captured formally in the properties of their respective model elements (so the information is there for analysis).

Admin

Figure 67 and Figure 72 show the Admin actor goal diagram in both jUCMNav and Tau-GRL profile. In this comparison, our focus is on correlations, decomposition, contribution, goal, softgoal and task. They are captured, but with limitations similar to the ones discussed before.

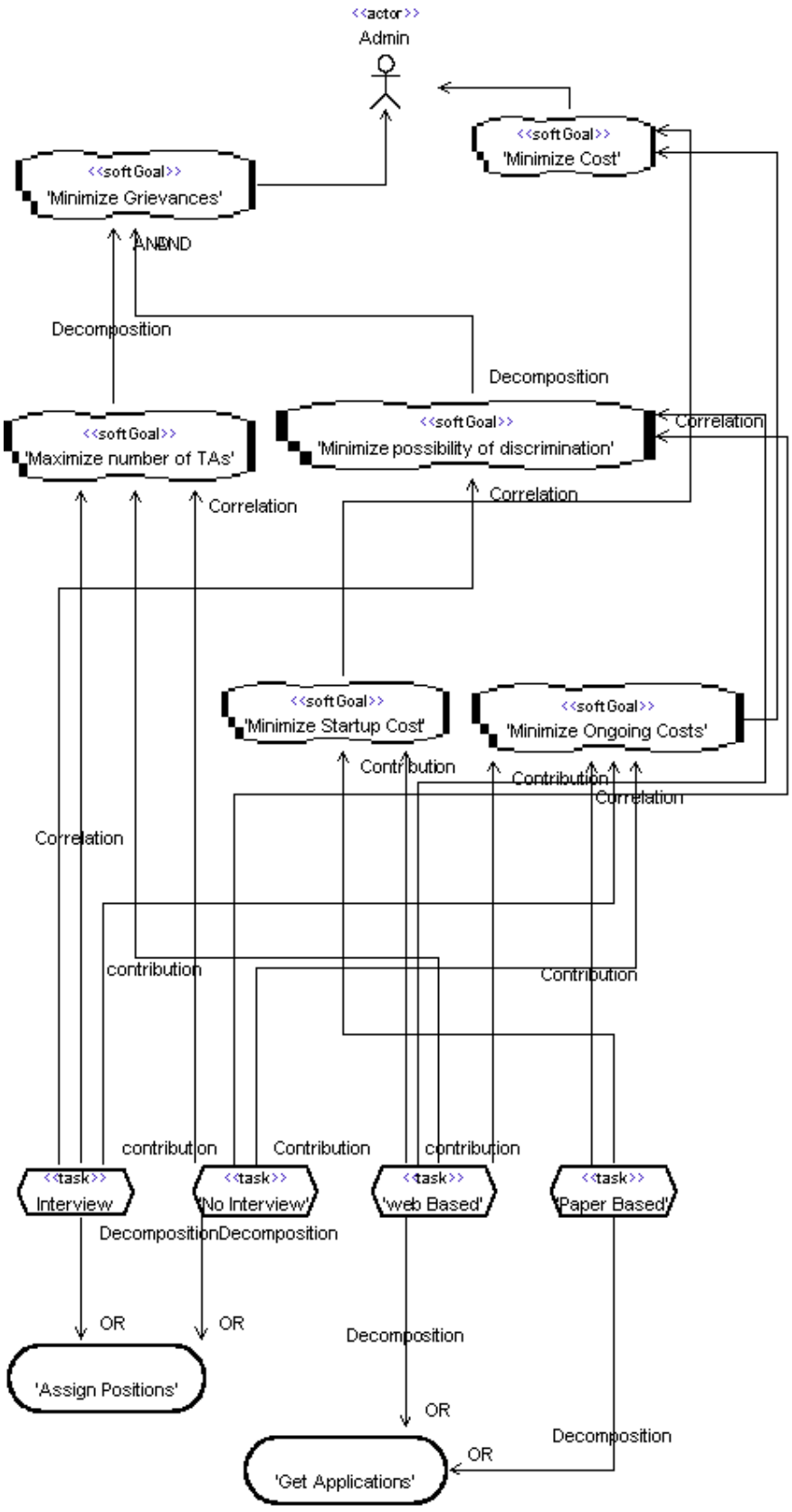

Figure 72 Admin, Modelled with Tau GRL Profile

Student

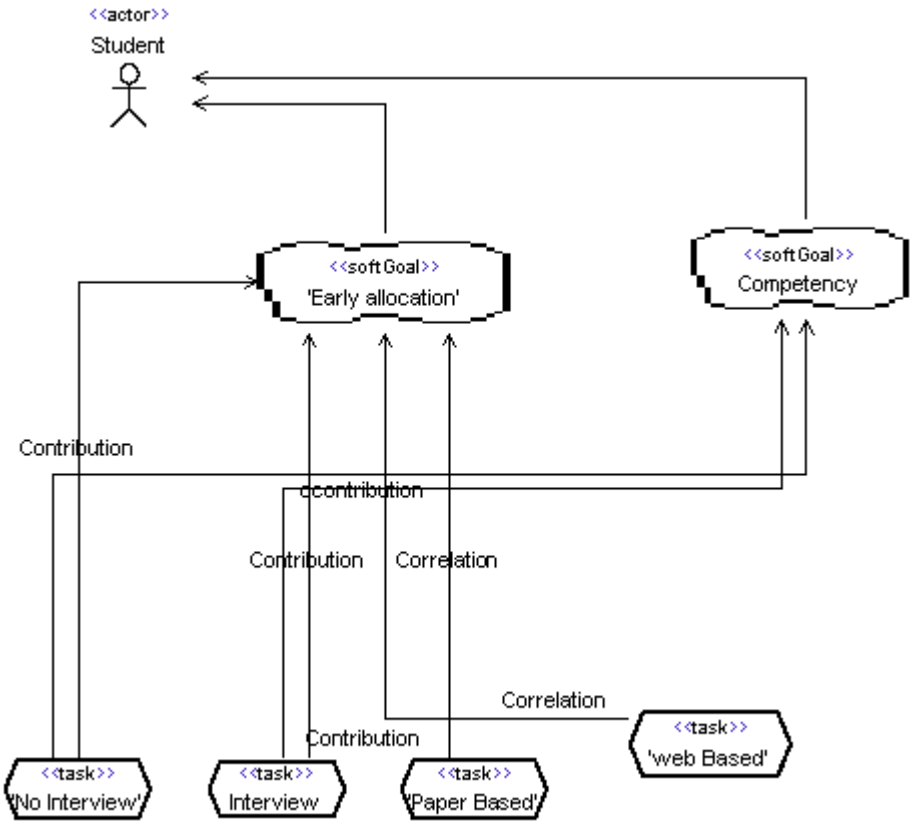

Figure 73 Student, Modelled with Tau GRL Profile

Actor Student's goals are to get a competent TA on time. Figure 68 and Figure 73 show these goals modeled with both tools. The above diagrams have intentional elements owned by the actor and some intentional elements that are outside of their boundaries.

TA Union

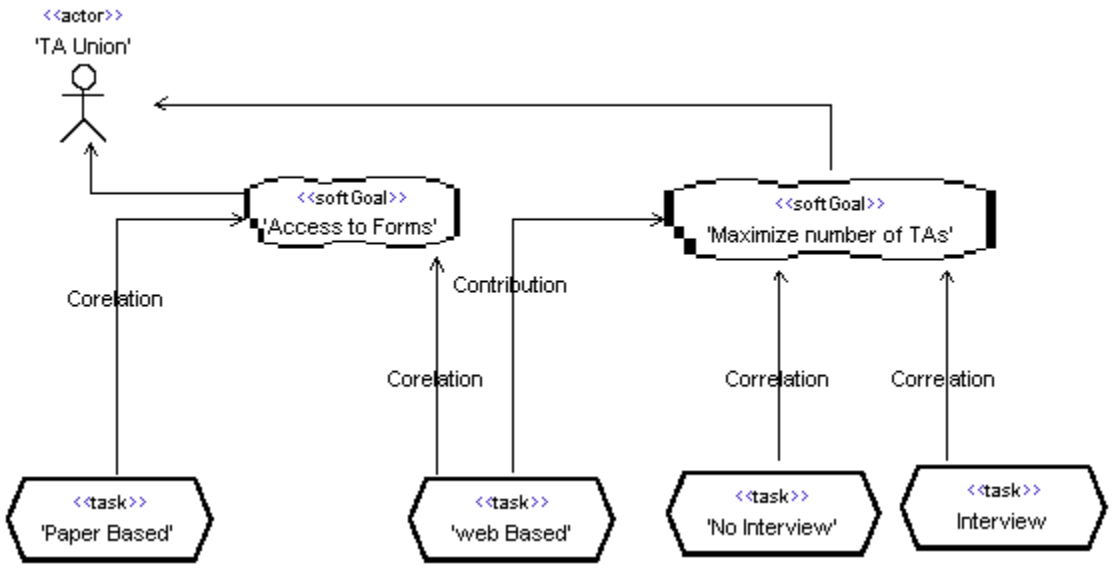

Figure 74 TA Union, Modelled with Tau GRL Profile

Actor TA Union's goals are modeled in Figure 69 and Figure 74. This actor wants a maximum of students to be able to fill forms for a TA application and that the number of hired TAs be maximized. Both diagrams show that a Web-based application is a good approach for maximizing the number of TAs by the contribution link of Web-based task with the softgoal "Maximize number of TAs".

5.2. The Merchant and Customer Dependencies Designed in Tau-GRL Profile

In this example, we cover all constructs and scenarios which are not discussed in the previous example.

Merchants and Customer Dependencies (with the Tau GRL Profile)

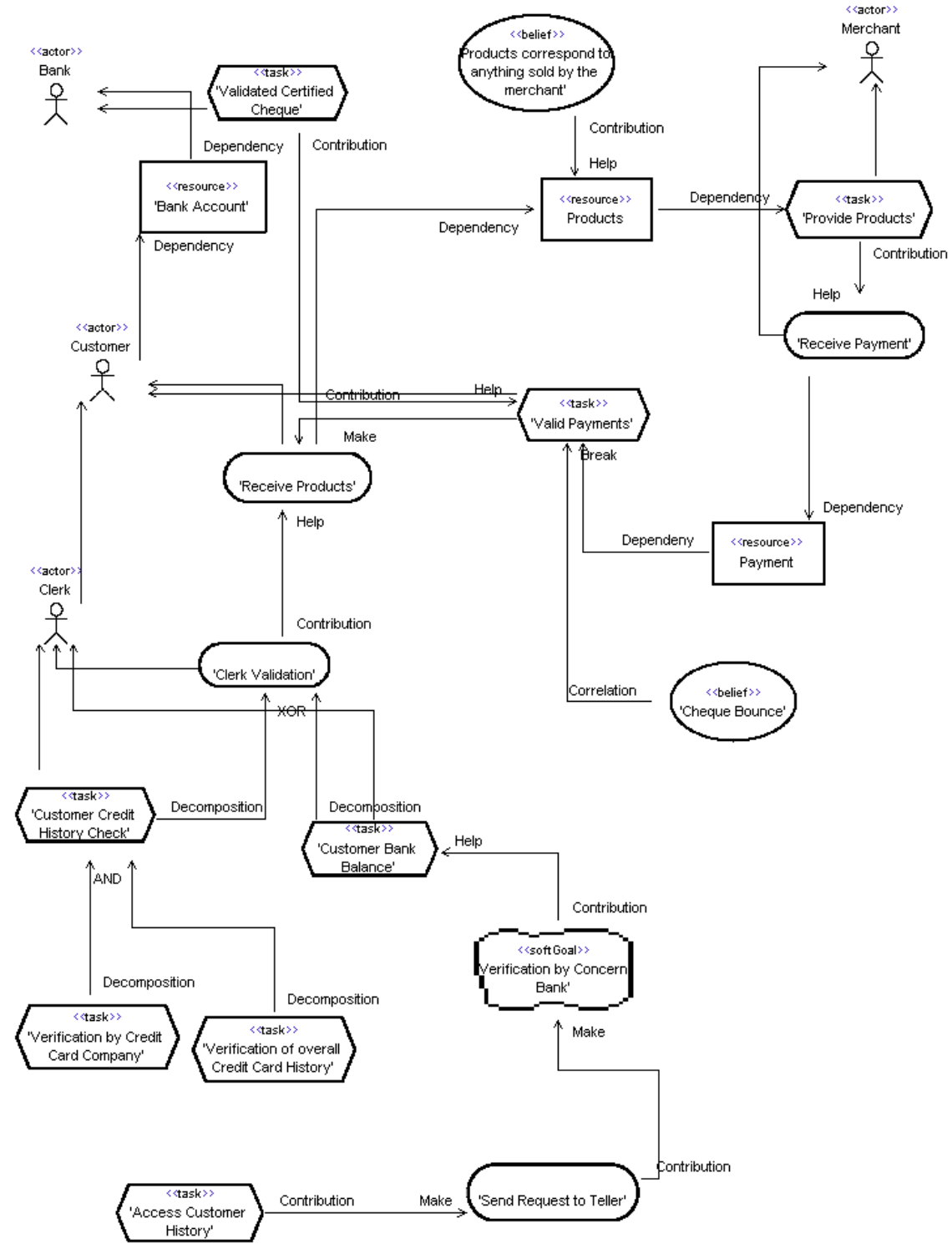

Figure 75 Merchant and Customer Dependencies, Modelled with Tau GRL Profile

This sample model has an actor (Bank), a goal (Receive Payment), a task (Provide Products), a belief (Cheque Bounce), resources (Products) and a softgoal (Verification by Concern Bank). All links (Dependency, Contribution, Correlation, and Decomposition) are also covered as mentioned in Table 4. The example also covers different scenarios like actor containing other actors, actor containing intentional elements, belief linked with a resource, intentional elements owned by an actor which depend on stand alone intentional elements and vice versa, actor depending on other actor. Figure 75 has Contribution types (Make, Help, and Break) but these are just names and they are actually set in the properties of the constructs. All valid links from actor to actor, actor to intentional elements, intentional elements to intentional elements and intentional elements to actor are also examined in the sample models. Table 4 summarizes all GRL constructs from sample models

Table 4 Summary of GRL Constructs Used in Sample Models

Category	Element Name	Figure#	Construct Instance
Intentional Elements	Goal	Figure 71	Get Applications
	Softgoal	Figure 71	Security
	Task	Figure 71	Interview
	Resource	Figure 75	Products
	Belief	Figure 75	Cheque Bounce
Actor	Actor	Figure 71	TA Candidate
Element Links	Decomposition	Figure 75	XOR (Clerk Validation: Goal)
		Figure 75	AND (Customer Credit History Check: Task)
		Figure 72	IOR (Get Applications: Goal)
	Contribution	Figure 75	Send Request to Teller: Goal
	Correlation	Figure 75	Valid Payments: Task
	Dependency	Figure 75	Valid Payments: Task

5.3. Evaluation

The comparison of the two designs provides a visual evaluation of the limitations of Tau profiling. We attempted in this example to include all of the GRL constructs. Actors, as well as all types of intentional elements, element links, and contributions are supported. However, we were not able to support *visually* some of these constructs using the profiling feature in Tau. This resulted in the need for compromises and forced us to deviate from our original GRL metamodel. The differences can be summarized as follows:

- Decompositions differ visually in the jUCMNav example from the Tau GRL profile. This is because there is no customized appearance for links in Tau GRL profile.
- The new GRL metamodel no longer supports Belief as a comment; it is now an IntentionalElement. This metamodel change has not yet been reflected in jUCMNav but was reflected in Tau.
- Due to Tau limitations, we were unable to implement, in a visual way, the actor boundary construct in Tau GRL profile. However, the actor boundary construct is well supported in the GRL profile as discussed in Chapter 3. We have linked IntentionalElements association to Actor, to preserve the sense of ownership.
- The qualitative values of Contribution links are not shown in Tau GRL profile but appear in the jUCMNav example. They exist as properties in the Tau GRL profile. It is therefore, possible for the modeller to set qualitative values as attributes.
- One thing that our Tau GRL profile supports well and that the current release of jUCMNav does not yet support is direct dependencies between actors.
- Tau does not support OCL implementation for profiling. There is a notion of informal constraints in Tau that are limited to text. Consequently, the GRL profile in Tau does not implement the constraints mentioned in Chapter 3, for each GRL element.

The extents to which GRL profile meet our requirements are examined in the following four subsections.

5.3.1 Integration with UML

To fulfill this requirement, we have created a model where the UML diagrams and the GRL diagrams are connected to each other. This makes the exchange of information between the two different types of diagram possible and allows achieving the integration requirement. This integration helps to create required UML diagrams based solely on GRL artefacts.

To make these connections, we have used a start link and end link, already available in Tau, between GRL constructs and UML diagram constructs. These are similar in spirit and form to the URN links discussed in Figure 8. This also allows to navigate from one diagram construct to the other. As shown in Figure 76 and Figure 77, all (blue coloured) underlined constructs of the GRL diagram are linked to the Use Case diagram constructs with (blue coloured) triangles. For instance, the use case actor Customer is linked to the GRL actor Customer. The user can navigate back and forth between the two diagrams by clicking on these elements.

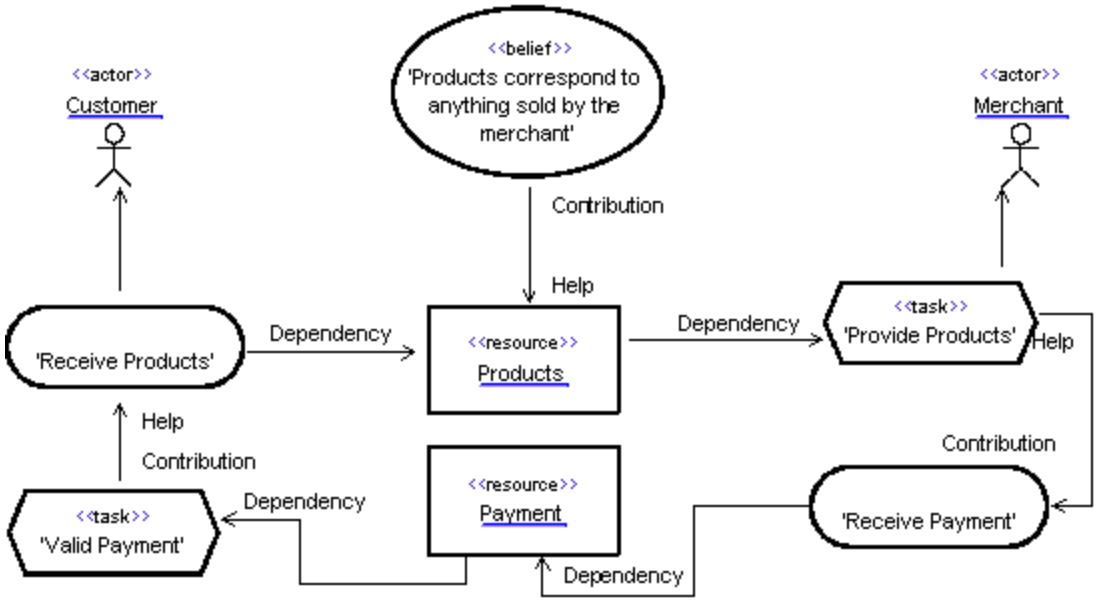

Figure 76 GRL Diagram to Show Links

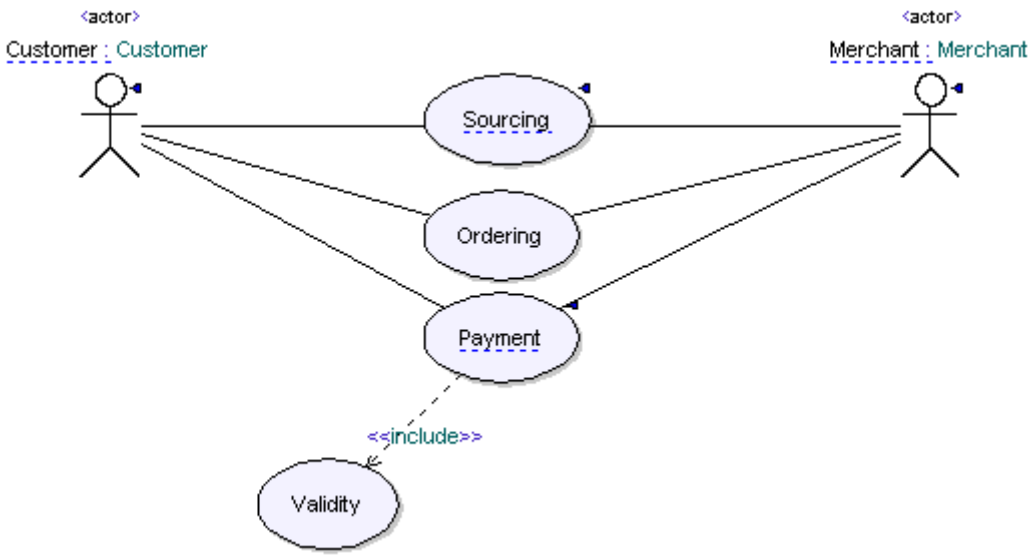

Figure 77 Use Case Diagram to Show Links

An interesting feature that we have provided is the re-usability of the GRL diagram constructs. This feature allows a UML diagram to re-use a reference to a GRL construct from another diagram. This means that any change in the re-used construct (e.g., the name or some other attribute) appears automatically on all its other occurrences in all the diagrams of the model.

An example demonstrating how our work fulfills this requirement is given below in Figure 78 - Figure 80. This example shows a system that is modelled first by a GRL diagram and, re-using some of the information available in the GRL diagram, we created two other UML diagrams representing the same model. A GRL actor can for example be dragged from a list of UML constructs and dropped onto a use case diagram or a sequence diagram to create new references to the original element.

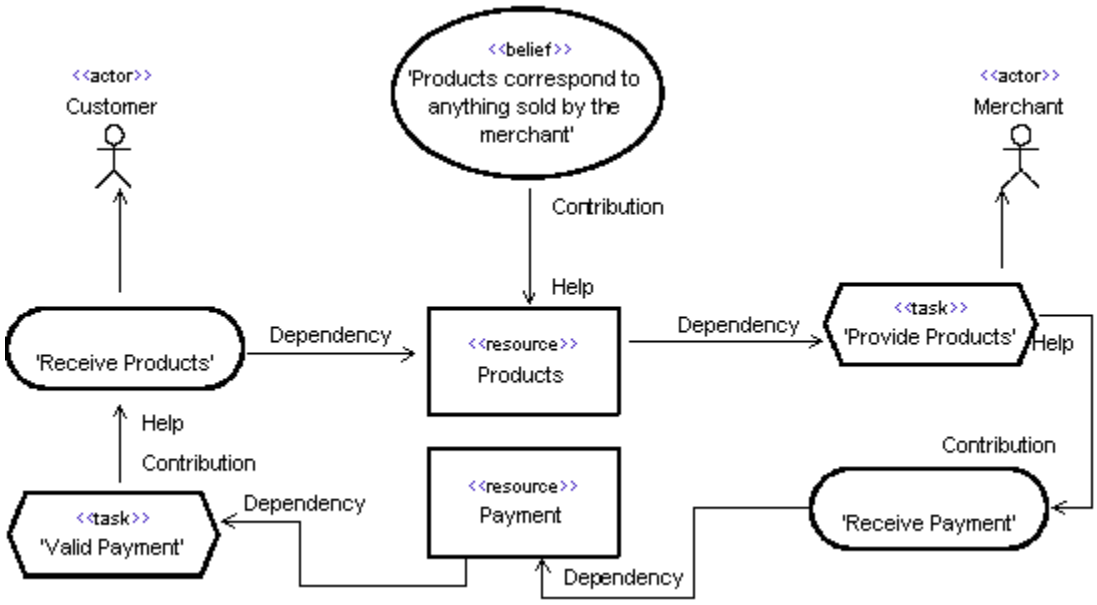

Figure 78 Customer Merchant Dependency (GRL Diagram)

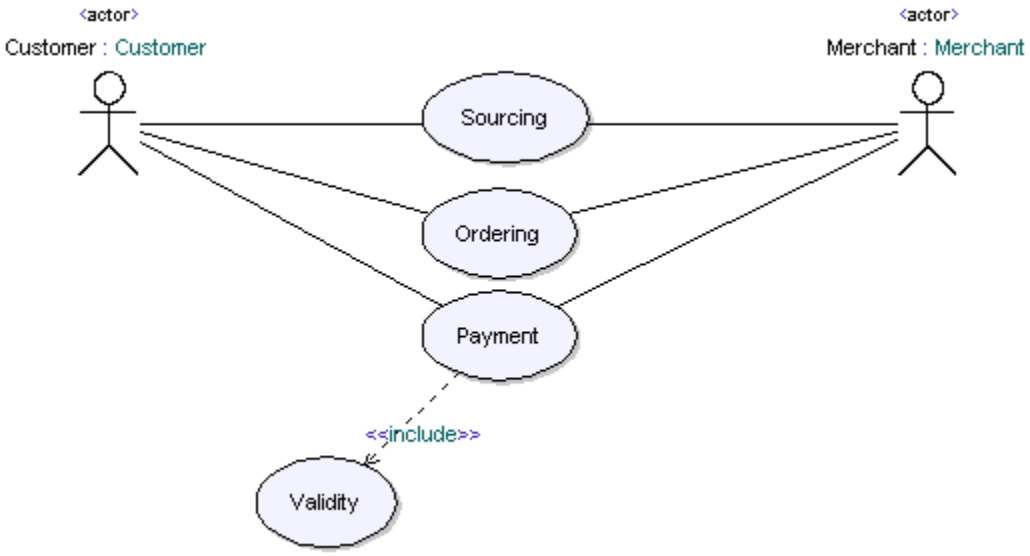

Figure 79 Customer Merchant Dependency (Use Case Diagram)

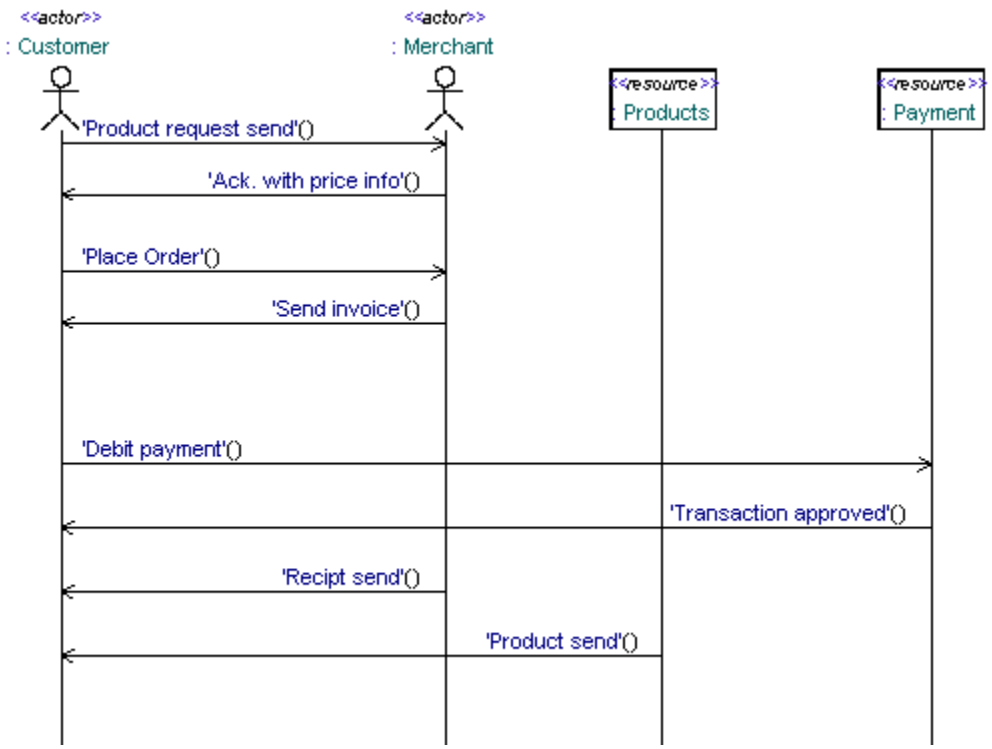

Figure 80 Customer Merchant Dependency (Sequence Diagram)

5.3.2 Diagram Pollution Avoidance

Because of the availability of separate GRL editors and a separate customized tool bar, diagram pollution is avoided in our work. This can be seen clearly in Figure 81, where the palette available for a GRL diagram in a model only contains GRL constructs.

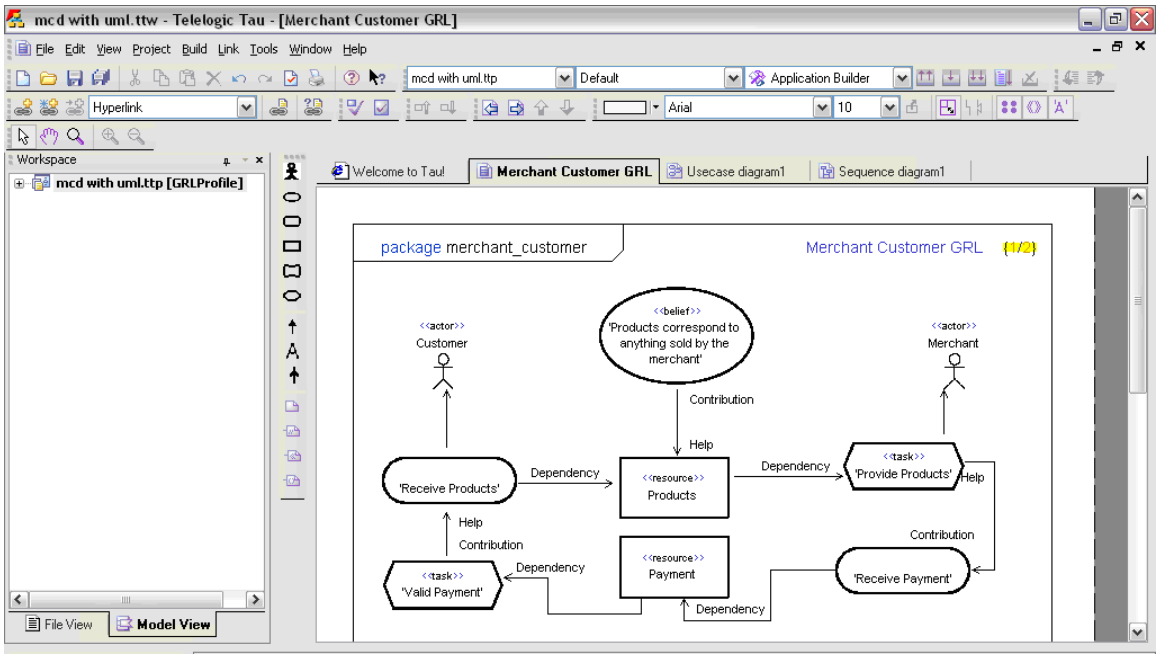

Figure 81 GRL Diagram in GRL Editor

5.3.3 Metamodel Stability

Our GRL profile is based on a metamodel that has been created two years ago and that is also at the core of the goal modelling tool jUCMNav. This metamodel is ongoing a standardization process by the ITU-T [17]. Consequently, our work meets the metamodel stability requirement, although we expect minor modifications in the final form of the standard.

5.3.4 Implementability of the Profiling Mechanism

We implemented our profile using both the Stereotype Mechanism and Metamodel Extension Mechanism. The latter approach led to better results in terms of what the editor enables modellers to do. Although the visualisation of some of the GRL elements is still an issue, all of the attributes can be set correctly through property panels.

5.4. Chapter Summary

This chapter discussed the implementation of the GRL profile. From the above examples and evaluations, it is possible to conclude that profiling practices are sufficiently mature

to support a goal modelling profile in a way that satisfies our four requirements. However, the tool we used for profiling GRL (Tau) is still incomplete in several aspects related to visualization. More effort is required to improve the tool functions, which will improve its ability to create good editors for UML profiles.

Chapter 6. Conclusions

6.1. Summary

In this thesis, we defined a UML profile for the Goal-oriented Requirement Language based on a GRL metamodel. This metamodel is currently used by an editor named jUCMNav. It is also undergoing standardization [17] in ITU-T, with an expected completion date in the fall of 2008. The profile was implemented for validation in an industrial-strength UML tool, namely Telelogic Tau G2 4.0. Two examples were designed to analyze how well our profile can be supported in practice (with a comparison to jUCMNav) and to evaluate Tau's features and limitations for supporting such a profile. Previous research work suggested metamodels for non-functional requirements and functional requirements. However, a standard UML profile addressing a goal-oriented modelling domain does not exist. All of the reviewed previous work was based on presumed and non-validated metamodels.

During our research, we extended the UML metamodel for a goal-oriented modelling domain. UML does not allow a direct customization of its metamodel. A profiling mechanism is rather provided through which a modeller can extend and then customize the UML metamodel. We used this feature to map the GRL metamodel classes with appropriate UML metaclasses as described in Chapter 3. The mapping was completed by strictly following the ITU-T guidelines for UML profile creation. We also ensured that the inherited UML metaclass constraints were not violated.

This thesis work entailed the following research process. Section 2.1 briefly discussed the UML architecture with a detailed focus on its infrastructure and superstructure. Additionally, the UML metamodel and its layers were also reviewed. Section 2.2 gave an overview of URN and of its meta-metamodel (Z.111). In Section 2.3, GRL was studied in detail with its elements and notations along with an overview of an evaluation mechanism. Section 2.4 presented an explanation of the GRL metamodel based on the draft ITU-T standard document Z.151. Section 2.5 and 2.6 examined the standard rec-

ommended methods for a UML profile creation. Tools supporting profile creation and ITU-T's standard for profiles were studied in detail. Section 2.7 discussed previous research work for goal modelling and compared their results with our stipulated requirements.

Chapter 3 explained the correspondence of the GRL metamodel with the UML metamodel. This chapter proposed appropriate UML metamodel classes that should be extended by GRL metamodel classes. This chapter included the description of all attributes, semantics, constraints, notations, and individual class references for stereotypes.

Chapter 4 included the implementation of our suggested UML profile for GRL using Telelogic Tau G2 4.0. Section 4.1 included a discussion of Tau. Section 4.2 entailed the detailed discussion of Tau's support for profile creation by the stereotype mechanism and the metamodel extension mechanism. Because of weaknesses in Tau's documentation, our implementation was based on the exploration of Tau's metamodel and its add-ins. We were forced to deviate from the original metamodel because of some limitations of Tau.

Chapter 5 consisted of experimentations and evaluation of our implemented profile. The evaluation was assessed based on a comparison between jUCMNav and our profile for the creation of two goal models.

6.2. Concluding Remarks

The work in this thesis is the first to design, implement, and evaluate a UML profile for standard-based goal-oriented modelling. Although some goal-modelling profiles already existed in the literature, such as those in [4][5][6][9][25], none of them is fully implemented. In our work, we addressed four requirements, which are 1) the integration with UML, 2) the avoidance of diagram pollution, 3) the stability of the metamodel and 4) the implementability of the profiling mechanism. Our work was compared with four related approaches [5][6][9][25]. The results of the comparison were detailed in Chapter 2 and they are repeated in Table 5 together with a new row (at the bottom) that addresses our UML profile for GRL. It is worth mentioning that our profile is the only one that meets all the stipulated requirements, mainly because it is the only one implemented. Interest-

ingly, the implementation of the profile showed us some major limitations in the state-of-the-art profile creation tool we used.

Table 5 Comparison of GRL Profile in Tau with Previous Work

	Integration With UML	**Diagram Pollution Avoidance**	**Metamodel Stability**	**Tool Support**
A Template based analysis of GRL [6]	Not Satisfied	Not Satisfied	Not Satisfied	Not Satisfied
UML profile for Enterprise Goal Modelling [9]	Satisfied	Not Satisfied	Not Satisfied	Partially Satisfied
UML profile for Softgoal by use case driven approach [25]	Satisfied	Not Satisfied	Partially Satisfied	N/A
Using UML to reflect non-functional requirements [5]	Not Satisfied	N/A	N/A	N/A
UML Profile for Goal-oriented Modelling	Satisfied	Satisfied	Satisfied	Satisfied

Our proposed solution acknowledges that the selected tool Tau is not yet in a position to completely support profiling. Due to Tau's limitations (which were detailed in our implementation and experiment work), we were forced to make alterations to the GRL metamodel. However, these alterations did not change the semantics of the original metamodel. There are visualization limitations for some of the elements but all the required attributes are accessible via a property panel.

Last but not least, the profile implementation was used to model two sample applications: a university Teaching Assistant allocation system, and a Merchant and Customer system. All the GRL constructs were covered by these examples. In addition, it was shown how the goal view integrates with the other types of diagrams in a UML model.

6.3. Future work

The extension of the UML metamodel for GRL is a significant step in establishing our metamodel's compatibility with the UML metamodel. Its successful implementation adds to our overall vision of effectively mapping the UML metamodel elements to GRL metamodel elements.

Future works include adding GRL strategies to the profile. This will enable modellers to capture in UML strategies meant to evaluate their goal models. The creation of a UML profile for Use Case Map (UCM) will also be a considerable contribution to modelling and would help covering the entire User Requirements Notation. However, the mission will be comparatively more technical than creating a UML profile for GRL due to the existence of some dynamic elements (e.g. dynamic stub) in UCM, and the fact that there are many more metaclasses, associations and attributes to support.

Finally, the integration of UML profiles for GRL with jUCMNav should also be considered for future work. UML tools could export goal models created with the profile to the jUCMNav XML-based format. Such integration would improve the way applications are validated.

References

[1] Abdulhadi, S., Grau, G., Horkoff, J., and Yu, E.: *i* Guide*. V. 3.0, August 2007. http://istar.rwth-aachen.de/tiki-ndex.php?page_ref_id=67.

[2] Amyot, D., Farah, H., and Roy, J.-F.: Evaluation of Development Tools for Domain-Specific Modeling Languages. R. Gotzhein, R. Reed (Eds.) *SAM 2006: Language Profiles - Fifth Workshop on System Analysis and Modelling*, Kaiserslautern, Germany, May 2006. LNCS 4320, 183-197, Springer.

[3] Amyot, D.: Introduction to the User Requirements Notation: Learning by Example. In: *Computer Networks*, 42(3), 285-301, 21 June 2003.

[4] Chung, L., Nixon, B.A., Yu, E., and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Dordrecht, USA, 2000

[5] Cysneiros, L.M. and Leite, J.C.S.P: Using UML to Reflect Non-Functional Requirements. *CASCON 2001*, Toronto, Canada, November 2001.

[6] Dallons, G., Heymans, P., and Pollet, I.: A template-based analysis of GRL. In: *Workshop on Evaluating Modeling Methods for System Analysis and Design (EMMSAD'05)*, Porto, Portugal, 493-504, June 2005.

[7] Eclipse: *Eclipse Modeling Framework (EMF)*, http://www.eclipse.org/emf/.

[8] Favre J.M.: Towards a Basic Theory to Model Model Driven Engineering In: *WISME, Lisboa, Portugal*, October 11, 2004.

[9] Grangel, R., Chalmeta, R., Campos, C., Sommar, R., and Bourey, J.-P.: A Proposal for Goal Modelling Using a UML Profile. *Enterprise Interoperability III*, 679-690, Springer, 2008.

[10] IBM: *Rational Software Architect 7.0.0.3*, July 2007. http://www-306.ibm.com/software/awdtools/architect/swarchitect/

[11] Institute for Software Integration System: *The Generic Modeling Environment (GME)*, 2004. http://www.isis.vanderbilt.edu/projects/gme.

[12] ITU-T: *Specification and Description Language (SDL) – ITU-T Recommendation Z.100*. Geneva Switzerland, 2007.

[13] ITU-T: *SDL-2000 combined with UML – ITU-T Recommendation Z.109*. Geneva, Switzerland, June 2007.

[14] ITU-T: *Notations to Define ITU-T Languages – ITU-T Draft Recommendation Z.111*. Geneva, Switzerland, April 2008.

[15] ITU-T: *Guidelines for UML profile design – ITU-T Recommendation Z.119*. Geneva, Switzerland, December 2006.

[16] ITU-T: *Language Requirement and Framework – User Requirements Notation – ITU-T Recommendation Z.150*. Geneva, Switzerland, February 2003.

[17] ITU-T: *User Requirements Notation (URN) – ITU-T Draft Recommendation Z.151*. Geneva, Switzerland, April 2008.

[18] Janmohamed, N.: *Expressing Goal-oriented Requirement Language in UML 2.0: Examining the functionality of UML Profiles*. CSI 4900 report, SITE, University of Ottawa, April 2005.

[19] *jUCMNav*, version 3.1.0, http://jucmnav.softwareengineering.ca/jucmnav/

[20] OMG: *Catalog of UML Profile specifications*. http://www.omg.org/technology/documents/profile_catalog.htm.

[21] OMG: *Meta Object Facility (OMG MOF): core specification version 2.0.* formal /2006-01-01, http://www.omg.org/spec/MOF/2.0/PDF.

[22] OMG: *Unified Modeling Language (OMG UML): infrastructure version 2.1.2* formal /2007-11-04, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF.

[23] OMG: *Unified Modeling Language (OMG UML): superstructure version 2.1.2* formal /20007-11-02, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

[24] Roy, J.-F., Kealey, J., and Amyot, D.: Towards Integrated Tool Support for the User Requirements Notation. R. Gotzhein, R. Reed (Eds.) *SAM 2006: Language Profiles - Fifth Workshop on System Analysis and Modelling*, Kaiserslautern, Germany, LNCS 4320, 198–215, Springer, May 2006.

[25] Supakkul, S. and Chung, L.: A UML profile for goal-oriented and use case-driven representation of NFRs and FRs. *SERA'05 Revised Selected Papers*, LNCS 3647, 29-41, Springer, 2006.

[26] Telelogic, An IBM Company: *Tau Documentation 4.0*. February 2008. http://www.telelogic.com/products/tau/tau/index.cfm

[27] Weiss M. and Amyot D.: Business Process Modeling with URN. In: *International Journal of E-Business Research*, 1(3), 63-90, July 2005.

[28] Xactium: *XMF-Mosaic Getting Started Guide*, Version 1.0, July 2005. http://www.xactium.com.

[29] Jiang, Y., Shao, W., Zhang, L., Ma, Z., Meng, X., and Ma, H.: On the Classification of UML's Model Extension Mechanism. *UML 2004*, LNCS 3273, 54-68, Springer, 2004.

[30] Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. *3rd IEEE Int. Symp. on Requirements Engineering*, Washington, USA. IEEE CS, 226-235, 1997.