



UNIVERSIDADE FEDERAL FLUMINENSE

Karen da Silva Figueiredo

**MODELING AND VALIDATING NORMS IN
MULTI-AGENT SYSTEMS**

Niterói, Rio de Janeiro, Brazil

2011

Karen da Silva Figueiredo

MODELING AND VALIDATING NORMS IN MULTI-AGENT SYSTEMS

Dissertation presented to the Computer
Science Department of the
Universidade Federal Fluminense
(UFF) in partial fulfillment of the
requirements for the Master Degree.

Advisor:
Viviane Torres da Silva

Niterói, Rio de Janeiro, Brazil

2011

MODELING AND VALIDATING NORMS IN MULTI-AGENT SYSTEMS

Dissertation presented to the Computer
Science Department of the
Universidade Federal Fluminense
(UFF) in partial fulfillment of the
requirements for the Master Degree.

Approved by:

Prof. DSc. Viviane Torres da Silva (Advisor) – IC/UFF

Prof. DSc. Christiano de Oliveira Braga – IC/UFF

Prof. DSc. Ricardo Choren Noya – IME

Prof. DSc. Leonardo Gresta Paulino Murta – IC/UFF

Niterói, Rio de Janeiro, Brazil

2011

ACKNOWLEDGEMENTS

To Kṛṣṇa, the Supreme Creator and Director of the Universe, I thank You for allowing me to achieve this one more stage of my life. To Him all my humble obeisance!

To Prabhupāda, my Original Guru, I thank You for inspiring me through Your example. Every time I felt like "giving up" I remembered of Your story and how all of this is just a tiny little grain of sand compared to it.

To my mother and grandmother, the sunshine of my life, I thank you for being so kind, careful and adorable even when I couldn't correspond to you.

To all my friends, especially to Isabela, Leonardo, Lilia, Larissa and Caitanya, for being there for me when "*the rain started to pour*".

To my boyfriend for being so understanding and lovely all the time. Thanks for being part of my life so intensely.

To the teachers Viviane Silva (master advisor), Aline Vasconcelos (graduation advisor), Leonardo Murta and Ana Cristina Bicharra for all the support and incentive during this journey, especially to Viviane, for all the patience that you had with me.

Finally, to all the colleagues of course and friends that I met in Niterói for making living here more easy and funny.

*tasmāc chāstram pramāṇam te
kāryākārya-vyavasthitau
jñātvā śāstra-vidhānoktam
karma kartum ihārhasi*

“One should therefore understand what is duty and what is not duty by the norms of the scriptures. Knowing such rules and regulations, one should act so that he may gradually be elevated.” – Kṛṣṇa, Bhagavadgītā As It Is (Prabhupāda, 1968)

ABSTRACT

The designers of open multiagent systems have to deal with the possibility that agents may not behave as they are supposed to. Norms provide a means for regulating agents' behavior by describing their permissions, prohibitions and obligations. In this dissertation we propose a normative modeling language called NormML that makes possible the modeling of the main elements that compose the norms. The dissertation presents not only the abstract syntax but also the concrete syntax of NormML used by designers to model the norms of multi-agent systems.

It is also the aim of this work to present a mechanism used to validate the norms modeled at design time. The validation process has two steps. First, the mechanism checks if the models respect the constraints defined by the normative language and, then, checks for conflicts among the modeled norms. Two norms are in conflict if, for instance, one states a permission and another a prohibition to a given agent to execute the same action at the same time period.

The modeling and the validation of the norms are supported by a tool called NormML Tool Kit that offers a set of plugins to the Eclipse IDE platform with the aim to make possible the creation and validation of the norms modeled with NormML.

Keywords: Norm, Multi-agent System, Modeling Language, Modeling, Metamodel, Validation, Conflict.

LIST OF FIGURES

Figure 2.1 Example of metamodel and model representations adopted in this work

Figure 2.2 SecureUML metamodel

Figure 3.1 Conference management process of the local conference

Figure 4.1 The NormML metaclasses used to define the deontic concepts of a norm

Figure 4.2 The NormML metaclasses used to define the involved entities of a norm

Figure 4.3 The NormML metaclasses used to define the action of a norm

Figure 4.4 The NormML metaclasses used to define the resource of a norm

Figure 4.5 The NormML metaclasses used to define the activation constraints of a norm

Figure 4.6 The NormML metaclasses used to define the operands of an If constraint of a norm

Figure 4.7 The NormML metaclasses used to define the sanctions of a norm

Figure 4.8 The NormML metaclasses used to define the context of a norm

Figure 4.9 Norm N1

Figure 4.10 Norm N2

Figure 4.11 Norms N3 and N4

Figure 4.12 Web store norms graphical model

Figure 6.1 NormML graphical model of N1 and N2

Figure 6.2 NormML graphical model of N3, N7 and N8

Figure 6.3 NormML graphical model of N4, N5 and N6

Figure 6.4 NormML graphical model of N9, N10 and N11

Figure 7.1 The NormML Tool Kit process

Figure 7.2 The NormML Editor wizard

Figure 7.3 Eclipse perspective of the NormML Editor plugin

Figure 7.4 Creation and edition of N1 in the NormML Editor plugin

Figure 7.5 OCL invariant example in the Ecore editor

Figure 7.6 N1 violation of "NormContextCanNotBeNull" invariant

Figure 7.7 N1 violation of "NormContextCanNotBeNull" invariant result

Figure 7.8 NormML Conflict Checker menu

Figure 7.9 Check for conflicts result of the NormML Conflict Checker plugin

Figure 7.10 NormML metamodel described as a class diagram with EOS

Figure 7.11 N1 abstract model described as an object diagram with EOS

Figure 7.12 XSL template that implements Rule1

Figure 7.13 Main OCL operation of the check for conflicts described with EOS

Figure A.1 Conserved elements of the SecureUML metamodel

Figure G.1 N1 and N2

Figure G.2 N3

Figure G.3 N4 and N5

Figure G.4 N4 and N6

Figure G.5 N7

Figure G.6 N8

Figure G.7 N9 and N10

Figure G.8 N9 and N11

LIST OF TABLES

Table 2.1 Resources and their actions

Table 4.1 Resources and their actions

Table 5.1 Main elements of a norm

Table 5.2 Checking for conflicts analysis

Table 6.1 Main elements of the norms of the local conference management system

Table B.1 NormML dialect action hierarchy

Table D.1 NormML semantically opposite actions

Table E.1 NormML graphical model stereotypes

LIST OF ABBREVIATIONS

- AML – *Agent Modeling Language*
- AORML – *Agent-Object Relationship Modeling Language*
- API – *Application Programming Interface*
- AUML – *Agent Unified Modeling Language*
- DSL – *Domain-specific Language*
- EMF – *Eclipse Modeling Framework*
- EOS – *Eye OCL Software*
- IDE – *Integrated Development Environment*
- MAS – *Multi-agent System*
- MAS-ML – *Multi-agent System Modeling Language*
- MASQ – *Multi-Agent System based on Quadrants*
- MOF – *MetaObject Facility*
- NormML – *Normative Modeling Language*
- OCL – *Object Constraint Language*
- O-MASE – *Organization-based Multiagent System Engineering*
- OMG – *Object Management Group*
- PASSI – *Process for Agent Societies Specification and Implementation*
- RBAC – *Role-based Access Control*
- ST – *Secure Tropos*
- UML – *Unified Modeling Language*
- XML – *Extensible Markup Language*
- XSLT – *Extensible Stylesheet Language Transformations*

SUMMARY

CHAPTER 1: INTRODUCTION	12
1.1 OBJECTIVES AND MAIN CONTRIBUTIONS	13
1.2 STRUCTURE	14
CHAPTER 2: BACKGROUND.....	16
2.1 AGENTS AND MULTI-AGENT SYSTEMS.....	16
2.2 MODELS, METAMODELS AND SYNTAXES	18
2.3 SECURE UML	20
CHAPTER 3: NORMML: A NORMATIVE MODELING LANGUAGE.....	23
3.1 MAIN ELEMENTS OF A NORM	24
3.2 THE NORMML METAMODEL	27
3.2.1 <i>Deontic concept</i>	30
3.2.2 <i>Involved entities</i>	30
3.2.3 <i>Actions</i>	32
3.2.4 <i>Activation constraints</i>	35
3.2.5 <i>Sanctions</i>	37
3.2.6 <i>Context</i>	37
3.2.7 <i>Modeling norms with NormML</i>	38
3.3 THE WELL-FORMEDNESS RULES OF NORMML METAMODEL	40
3.4 CHECKING FOR CONFLICTS	44
3.4.1 <i>Context analysis</i>	45
3.4.2 <i>Involved entities analysis</i>	46
3.4.3 <i>Deontic concept analysis</i>	47
3.4.4 <i>Action analysis</i>	47
3.4.5 <i>Activation constraints analysis</i>	48
3.4.6 <i>Sanctions analysis</i>	49
3.5 THE NORMML CONCRETE SYNTAX.....	50
3.5.1 <i>Creating the graphical models</i>	52
3.5.2 <i>Mapping from concrete to abstract syntax</i>	54
CHAPTER 4: RELATED WORK.....	58
4.1 MAIN ELEMENTS OF A NORM	58
4.2 CHECKING FOR CONFLICTS	63
4.2.1 <i>Modeling languages, methodologies and organizational models</i>	63
4.2.2 <i>Other approaches that deal with norm conflicts</i>	65
CHAPTER 5: EXAMPLE APPLICATION	67
5.1 CONFERENCE MANAGEMENT SYSTEM	67

CHAPTER 6: EVALUATION	70
6.1 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH NORMML	70
6.2 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH AORML.....	78
6.3 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH GAIA.....	79
6.4 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH OPERA.....	80
6.5 FINAL REMARKS	81
CHAPTER 7: THE NORMML TOOL KIT	82
7.1 MODELING NORMS	83
7.2 CHECKING (CONCRETE) MODELS.....	85
7.3 CHECKING FOR CONFLICTS	88
7.3.1 <i>Transforming concrete to abstract models</i>	89
7.3.2 <i>Running operations to check the abstract models</i>	92
CHAPTER 8: CONCLUSION AND FUTURE WORK	94
REFERENCES	96
APPENDIX A: THE NORMML EXTENSION OF SECUREUML	104
APPENDIX B: NORMML DIALECT ACTION HIERARCHY	107
APPENDIX C: THE WELL-FORMEDNESS RULES OF NORMML	109
APPENDIX D: SEMANTICALLY OPPOSITE ACTIONS	114
APPENDIX E: LIST OF THE GRAPHICAL MODEL STEREOTYPES OF THE NORMML CONCRETE SYNTAX	115
APPENDIX F: FROM NORMML CONCRETE MODELS TO ABSTRACT MODELS	116
APPENDIX G: LOCAL CONFERENCE MANAGEMENT SYSTEM ABSTRACT MODELS	120
APPENDIX H: CONFLICT CASES	128

CHAPTER 1: INTRODUCTION

Open multi-agent systems (MAS) are societies in which autonomous, heterogeneous and independently designed agents can work towards similar or different ends (López y López, 2003). In order to cope with the heterogeneity, autonomy and diversity of interests among the different members, governance (or law enforcement) systems have been defined. A governance systems define a set of norms (or laws) that must be followed by the system entities.

According to Psychology and Sociology, norms are the rules that a society or a group uses to define appropriate and inappropriate values, beliefs, attitudes and behaviors (Deutch and Gerard, 1955). In MAS, Norms are used to regulate the behavior of the agents by describing the actions that can be performed or states that can be achieved (*permissions*), actions that must be performed or states that must be achieved (*obligations*), and actions that cannot be performed or states that cannot be achieved (*prohibitions*). They represent a way for agents to understand their responsibilities and the responsibilities of the others. Norms are used to cope with the autonomy, different interests and desires of the agents that cohabit the system.

Norms can be defined at *design time* together with the modeling of the system, or created at *runtime* by agents that have the power to do so (López y López, 2003). In this dissertation we focus on the description of norms at *design time*, thus; the creation of norms at *runtime* is out of the scope of this work.

The modeling of norms is an important part of the specification of a system and should be treated as an essential task of MAS design for two reasons:

- I. *Norms refer to actions, entities and resources that compose a system.* So, the refinement of the system may influence the norms and the definition of a new norm will only be possible if the actions, entities and resources being mentioned in the norm are being considered in the system design.
- II. *Norms' conflicts can cause the redesign of a system.* During the specification of the system norms conflicts may arise. Two norms are in conflict if, for instance, one gives a *permission* and another a *prohibition* to an agent to execute the same action at the same time period.

When norms are defined at *design time* some of those conflicts can be detected and solved by, for instance, amending the conflicting norms, which might cause the system's redesign (by the inclusion of new actions and agents, for example). By solving at least part of the conflicts at *design time*, it is possible to reduce the time the agents will spend executing such task at *runtime*.

Due to the interdependency between the modeling of norms and the modeling of the elements of the system and the importance of finding out conflicts and solving them at *design time*, it is important that the modeling languages and the notations used by methodologies and organizational models to model MAS make possible the modeling of the norms together with the modeling of the whole system and also provide mechanism for solving the conflicts at design time.

Although there are many modeling languages and notations, proposed by methodologies and organizational models, that provide support to the modeling of norms, none of the analyzed approaches provide support to the modeling of all elements that compose a norm identified during this work. Moreover, those approaches also fail on providing support for the verification of conflicts among the norms modeled at design time.

1.1 OBJECTIVES AND MAIN CONTRIBUTIONS

Given the foregoing, the goal of this work is to develop a modeling language called NormML that is able to model the norms of a MAS and to check the conflicts between these norms at *design time*. In order to do so, we first identify the main elements that compose a norm, by analyzing the literature on specification and implementation of norms, to guarantee that the proposed modeling language widely supports the description of norms.

The novelty of our approach is threefold: (i) the modeling language itself, to model norms and its main elements; (ii) a mechanism for checking for conflicts between norms that considers the main elements that compose the norms; and (iii) a tool to support the modeling of norms using NormML and can automatically validate

the models according to the metamodel of the language and check conflicts between norms at *design time*.

This work also aims at comparing the proposed modeling language NormML with related work to: (i) investigate if the elements that compose norms can also be modeled by using the MAS modeling languages and notations provided by methodologies and organizational models; and (ii) to explore the MAS languages, methodologies and organizational models in order to find out if and how they give support to the checking of conflicts at *design time*.

1.2 STRUCTURE

The dissertation is organized as follows: Chapter 2 provides some background material for the rest of this work.

Chapter 3 presents the normative modeling language NormML and identifies the main elements that compose a norm. The metamodel of the language is described and the mechanism used to validate the models and check for conflicts between the modeled norms is detailed. Also, a graphical notation for NormML models is proposed together with a mapping between the graphical notation and the abstract notation of the language.

Chapter 4 discusses the support given by the modeling languages and the notations provided by the methodologies and organizational models analyzed to model norms and to check norm conflicts. Also, other related approaches in norms conflicts are addressed.

In Chapter 5 an example of application of norms in a MAS is presented. The application is situated in the area of Conference Management and is used in Chapter 6 to evaluate the proposed modeling language.

In Chapter 6 we use NormML to model the application Conference Management presented in Chapter 5 and to check for norms conflicts. A comparison between NormML and some related MAS modeling approaches is traced to evaluate our approach.

Chapter 7 presents the NormML Editor and the NormML Conflict Checker plug-ins. They were developed to the Eclipse IDE (The Eclipse Foundation, 2011)

platform with the objective to support the modeling and validation of norms using NormML and the checking for conflicts in NormML models. Chapter 8 concludes and describes some future works.

CHAPTER 2: BACKGROUND

In this chapter we briefly provide background material for the rest of this work. First, in Section 2.1 we introduce the terms agents and multi-agent systems since they concern the scope of this work.

Section 2.2 introduces the notions of models, metamodels and syntaxes that are necessary to understand the design of NormML.

In this dissertation we want to explore the modeling of norms using RBAC concepts. So, in Section 2.3 we introduce SecureUML (Basin *et al.*, 2006), a Domain-specific Language (DSL) for modeling RBAC policies. The reasons why SecureUML was chosen are: it has been applied successfully both in academic projects (Basin *et al.*, 2006) and industrial ones (Clavel *et al.*, 2008); it has a well-defined syntax, given by its metamodel; it has a formal semantics (Basin *et al.*, 2009); and it is designed specifically for RBAC modeling.

2.1 AGENTS AND MULTI-AGENT SYSTEMS

A popular definition for software agent is “a software program that does something, often on behalf of a person or other agent” possibly by: (i) involving automated tasks; (ii) using some intelligence; (iii) communicating with the user or other agents in a cooperative and coordinated manner; (iv) learning and changing its behavior over time; and (v) operating on its own initiative (OMG Agent Platform Special Interest Group, 2011). Bradshaw (1997) lists a set of properties that agents have, proposed at first by Etzioni and Weld (1995) and Franklin and Graesser (1996), which are commonly accepted by the researches of the area. Those properties are:

- **Autonomy:** the ability to execute its own tasks and to achieve its own goals without necessarily requiring user influence;

- **Collaborative behavior:** the capability of working together with other agents (by cooperating, negotiating, coordinating and delegating tasks) to achieve a common goal, i.e., the goal of the system where they are interacting;
- **Reactivity:** the ability to sense real-time domain events and act triggered by them;
- **Communicability:** the ability to communicate with humans, other agents, legacy systems, and information sources;
- **Personality:** the capability of manifesting the believable attributes such as emotions;
- **Adaptability:** the ability to learn and evolve based on their experiences, other agents experiences and changes of the host place; and
- **Mobility:** the capability of moving from one host place to another.

A system may contain one or more agents, in the latter case, we call it a **multi-agent system** (MAS). A MAS consists of agents, objects and organizations. **Organizations** can be understood from two perspectives: first as the process of grouping a set of agents and other organizations (i.e. sub-organizations), and second, as an entity with its own goals (Dignum, 2009). Agents, objects and organizations are immersed in **environments**, i.e. a local host that provide resources and offer services (Silva *et al.*, 2003). Each agent stores information about the states of the environment it inhabits and about other entities of the system. Those nested information are called the agent's **beliefs** (Wooldridge, 1997).

Agents are goal-oriented entities, i.e., they execute in order to achieve a set of goals that represent the agent's desires (Rao and George, 1995). Agents can execute a set of communicative actions (as the sending and receiving of messages mentioned above) and non-communicative **actions** to achieve their goals. When a set of actions are executed with the specific objective of achieving a certain goal, it is called a **plan**.

Due to its mobility, agents can move from an organization to another and from an environment to another. When an agent enters an organization, it must commit to (at least) one role described in the organization. A **role** restricts the behavior of an agent in the organization, defining its social behavior (Silva *et al.*, 2003). The interactions between agents of an organization occur through the roles played by them. Each role defines a set of **protocols** to regulate its interactions. Protocols are composed of dialogue structures, i.e. **messages** that the agent playing the role can send or receive.

2.2 MODELS, METAMODELS AND SYNTAXES

A **model** is an abstraction of a phenomenon of the real world. A modeling language provides a vocabulary (concepts and relations) for creating models. Such vocabulary is described by the **metamodel** of the modeling language which elements formalize the language concepts and their relationships.

A metamodel may include constraints that associate semantic restrictions to the elements of the metamodel. Those constraints specify additional properties that the models must fulfill as instances of the metamodel, i.e. specify the **well-formedness** conditions (or **well-formedness rules**) of the models with respect to its metamodel and the *consistency* conditions between metamodel concepts. A model always conform to a single metamodel.

Meta-Object Facility (MOF) is a standard from the Object Management Group (Object Management Group, 2011a) that states an abstract language for describing structures of objects that can be represented in a given language, i.e. MOF specifies a language for metamodels description. MOF corresponds to the top level (M3) of the four layer architecture of metamodeling illustrated in Figure 2.1.

Each model in a layer M_x is an instance of a model of M_{x+1} . Thus, in the layer M2 languages metamodels can be described using MOF as metalanguage (i.e. the language used to describe the metamodel vocabulary), e.g. the UML metamodel (Object Management Group, 2011b) is an instance of the MOF meta-metamodel.

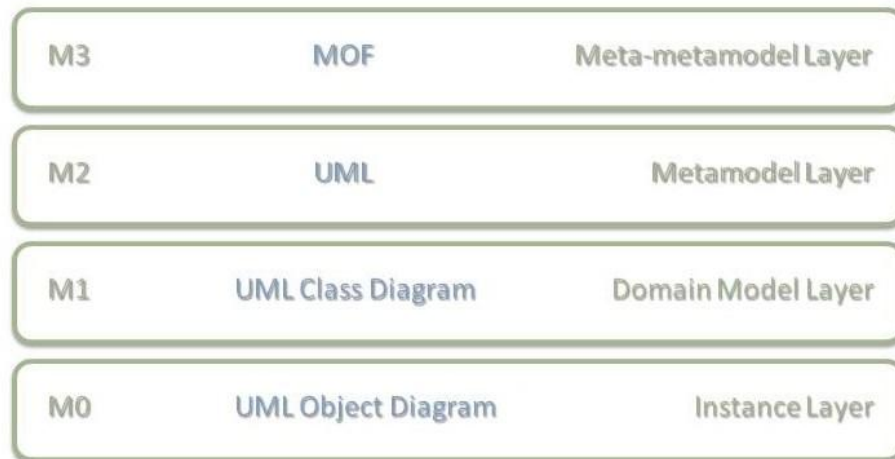


Figure 2.1 Example of the MOF architecture

In the M1 layer, domain models can be defined according to the metamodel of M2, e.g. an UML class diagram is an instance of the UML metamodel. And in the M0 layer instances of the elements of M1 can be described, e.g. an UML Object Diagram is an instance of the UML Class Diagram.

Another approach is to use UML as metalanguage in the top of the metamodeling architecture. By choosing UML as metalanguage, a metamodel is represented by a class diagram, its constraints are written in OCL (Object Constraint Language) (Object Management Group, 2011c), and the models are represented by object diagrams. This is the choice followed in this work as illustrated in Figure 2.2. The elements described in the class diagram are metaclasses and metarelationships and the elements described in the object diagram are classes and relationships instances of the former elements. OCL is then used to describe restrictions over the elements of the class diagram that are checked by queering the elements of the object diagram in order to guarantee that the model represented in the diagram complies with the metamodel represented in the class diagram.

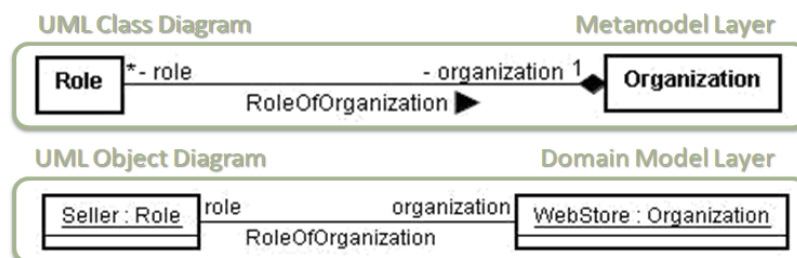


Figure 2.2 Example of metamodel and model representations adopted in this work

A modeling language specification may distinguish between **abstract syntax** and notation (also called **concrete syntax**). The abstract syntax defines the language primitives used to construct models as the vocabulary described by the metamodel, whereas the concrete syntax defines the graphical representation of these primitives as icons, labels, or figures. In this work we propose both a concrete and an abstract syntax to the normative modeling language, and also a set of rules that guides the translation from one to another.

2.3 SECURE UML

SecureUML (Basin *et al.*, 2006) provides a language for modeling *Roles*, *Permissions*, *Actions*, *Resources*, and *Authorization Constraints*, along with the relationships between permissions and roles, actions and permissions, resources and actions, and constraints and permissions. SecureUML leaves open what the protected *resources* are and which actions they offer to clients.

ComponentUML (Basin *et al.*, 2006) is a simple language for modeling component-based systems that provides a subset of UML class models: entities can be related by associations and may have attributes and methods. In Basin *et al.* (2006), the elements of the ComponentUML class models are used as resources in SecureUML. Therefore, *Entity*, *Attribute*, *Method*, *Association* and *AssociationEnd* are the possible protected resources. The actions that can be used to restrict the access to these *resources* can be either *Atomic* or *Composite*. The atomic actions are intended to map directly onto actual operations of the modeled system (delete, update, read, create and execute). The composite actions are used to hierarchically group atomic ones. In Table 2.1 we describe the actions used to restrict the access to the resources, where underlined actions are composite actions.

Resource	Actions
Entity	create, <u>read</u> , <u>update</u> , delete, <u>full access</u>
Attribute	read, update, <u>full access</u>
Method	Execute
AssociationEnd	read, update, <u>full access</u>

Table 2.1 Resources and their actions

The metamodel of SecureUML+ComponentUML is shown in Figure 2.2. By using such SecureUML+ComponentUML metamodel (from now on referred as SecureUML metamodel) it is possible, for instance, to specify the permissions a user playing a given role must have to execute a method (or to update an attribute) of a resource. In order to do so, it is necessary to instantiate the metaclasses *User*, *Role*, *Permission*, *Method* (or *Attribute*) and *AtomicExecute* (or *AtomicUpdate*) from the SecureUML metamodel.

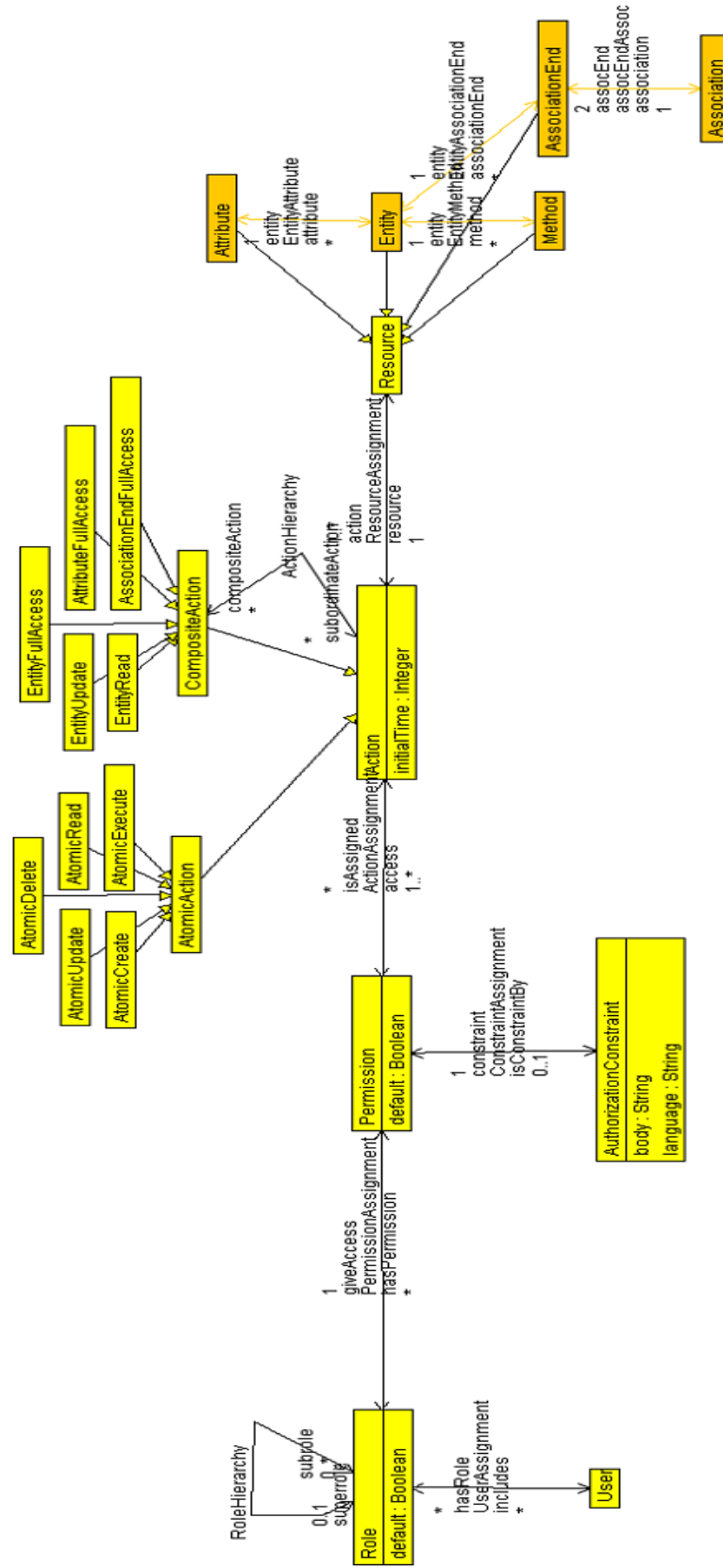


Figure 2.2 SecureUML metamodel

CHAPTER 3: NORMML: A NORMATIVE MODELING LANGUAGE

This chapter presents the normative modeling language called NormML, which is the core of this work. The main goal of NormML is to support the modeling of the norms of a MAS and ensure that there are no conflicts between the norms described. The current version of NormML that is being presented in this work is an extension of its preliminary versions (Figueiredo *et al.* 2011, Figueiredo and Silva, 2010a and Silva *et al.*, 2010).

NormML is a UML-based modeling language for the specification of norms. The use of UML as metalanguage allows for an easy integration of NormML with UML-based MAS modeling languages such as AUML (Odell *et al.*, 2000), AML (Danc, 2008) and MAS-ML (Silva *et al.*, 2008). Moreover, we can use metamodel-based validation techniques in the scope of norms specified in NormML.

Our modeling language was designed with the perception that norm specification in MAS design and security policy specification in RBAC (Role Based Access Control) (Ferraiolo and Kuhn, 1992) design are closely coupled issues. RBAC security policies specify the *permissions* that a *user* has under a given *role*, while trying to access system *resources*. In MAS we specify the *norms* that regulate the behavior (or *actions*) of a *role*, an *agent* or an *agent playing a given role*, for instance. Although we consider security policies and norms coupled issues, norms can be violated since they only define how agents *should* behave.

This chapter is organized as follows. In Section 3.1 we identify the main elements that compose the norms and its characteristics in order to include these elements in the normative modeling language. In Section 3.2 we present the NormML metamodel and how it represents the main elements of the norms. Section 3.3 describes the well-formedness rules of the NormML metamodel and Section 3.4 details the mechanism used to check for conflicts between the modeled norms. Finally, in Section 3.5 a concrete syntax for NormML is proposed.

3.1 MAIN ELEMENTS OF A NORM

In this section we discuss the key static aspects of a norm, i.e., the main elements that compose a norm: *deontic concept*, *involved entities*, *actions*, *activation constraints*, *sanctions* and *context*. Such elements were found out after investigating fourteen specification and implementation languages used to describe and implement norms (Aldewereld *et al.*, 2006, Cholvy, 1999, Cranefield, 2007, Fornara and Colombetti, 2008, García-Camino *et al.*, 2005, García-Camino *et al.*, 2006, Governatori and Rotolo, 2004, Lomuscio and Sergot, 2004, Lopes-Cardoso and Oliveira, 2010, López y López *et al.*, 2002, López y López, 2003, Silva, 2008, Vasconcelos *et al.*, 2007 and Vigano and Colombetti, 2008).

Our objective while investigating those implementation and specification works was not to do a critical analysis of each element mentioned in each work, but it was to try to consider all elements mentioned in them in order to do a deep investigation of the norms composition to develop a normative modeling language that could contemplate all such concepts.

In order to exemplify the elements presented below, consider some norms that govern a simplified version of a *web store*. The *web store* is being modeled as an organization that inhabits the *market place* environment and defines three roles to be played by the agents: *manager*, *seller* or *buyer*. All norms are in the context of the organization *web store* that inhabits the environment *market place*.

- **N1:** Sellers are permitted to update the price of the goods before receive the open sales alert from the manager.
- **N2:** Sellers are obliged to delete the good's advertisement if the stock of the good is empty.
- **N3:** Buyers are obliged to pay for the good that they have bought.
- **N4 (Punishment for the violation of N3):** Buyers are prohibited to buy goods.

The elements that compose a norm are based on the premise that *norms restrict the behavior of system entities during a period of time and define the sanctions applied when they are violated or fulfilled.*

- **Deontic Concept:** Deontic logic refers to the logic of requests, commands, rules, laws, moral principles and judgments (Meyer and Wieringa, 1993). In MAS, such concepts have been used to describe behavior restrictions for the agents in the form of *obligations* (what the agent must execute), *permissions* (what the agent can execute) and *prohibitions* (what the agent cannot execute). Thus, one of the main elements of a norm is the identification of the type of restriction being defined, i.e., the identification of the deontic concept associated with the norm.

E.g.: The deontic concept of the norm N1 is “permission”.

- **Involved Entities:** Since norms are always defined to restrict the behavior of entities, the identification of such entities whose behavior is being restricted is essential. A norm may regulate the behavior of *individuals* (e.g., a given agent, or an agent while playing a given role) or the behavior of a *group of individuals* (e.g., all agents playing a given role, groups of agents, groups of agents playing roles or all agents in the system).

E.g.: The entities involved in norm n1 are all the agents playing the role “Seller”.

- **Actions:** Since a norm defines restriction over the execution of entities, it is important to clearly represent the actions being regulated. Such actions can be *communicative* ones, typically represented by the sending and receiving of a message, or *non-communicative* actions (such as to access and modify a resource, to enter in an organization, to move to another environment, etc.).

E.g.: The action of N1 is a non-communicative action that represents the updating of the price of the good.

- **Activation Constraints:** Norms have a period during while they are active, i.e., during while their restrictions must be fulfilled. Norms can be activated by

one constraint or a set of constraints that can be: *the execution of actions, the specification of time intervals* (before, after, between), *the achievement of systems states or temporal aspects* (such as dates), and also the fulfillment / violation of another norm.

E.g.: N1 is activated before the seller receives the open for sales alert from the manager.

Figure 3.1 illustrates the life cycle of a norm. First, the norm is created in the system. Eventually, the set of activation constraints of the norm becomes true (i.e. the actions restricting the norm are executed, or the time intervals and states restricting the norm are achieved, or another norm is violated or fulfilled) and the norm is activated and must be fulfilled by the involved entities of the norm. If the involved entities execute the action or achieve the states regulated by the norm while the norm is activated, then the norm is fulfilled. Else, the norm is violated, i.e. the set of activation constraints of the norm becomes false and the involved entities of the norm did not execute the action or achieve the states regulated by the norm.

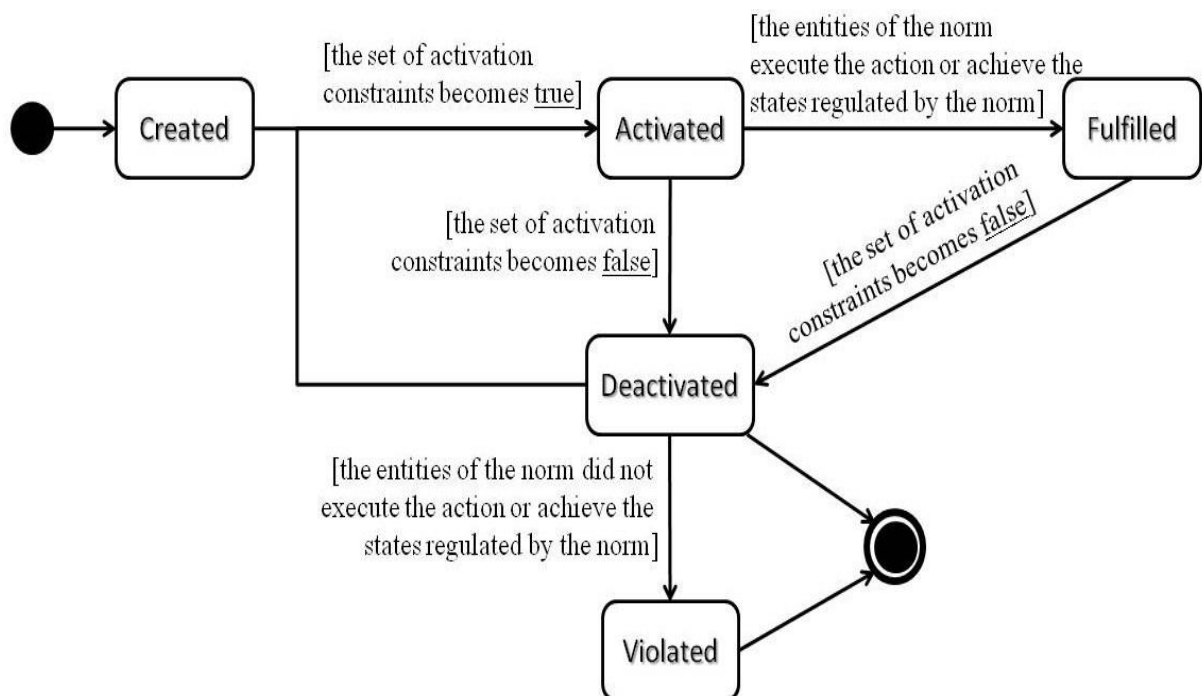


Figure 3.1 The life cycle of a norm

- **Sanctions:** When a norm is violated the entity that has violated this norm may suffer a *punishment*. The punishments are used by the system to regulate the behavior of the agents since they intend to discourage the agents to violate

the norms. In the same way, when a norm is fulfilled the entity who has followed the norm may receive a *reward*. The rewards are so used to motivate the agents to fulfill the norms. Such rewards and punishments are called sanctions and should be described together with the norm specification. A sanction is both part of a norm and is a norm itself.

E.g.: Norm N4 is a sanction that states a punishment if the norm N3 is violated.

- **Context:** Norms are usually defined in a given context that determines the area of their application. A norm can, for instance, be described in the context of a given *environment* and should be fulfilled only by the agents executing in the environment. A norm can also be defined in the context of an *organization* and must be fulfilled only by the agents playing roles in the organization.

E.g.: All the norms presented are defined in the context of the organization WebStore.

NormML gives support to the modeling of all elements presented in this section as shown in the next section.

3.2 THE NORMML METAMODEL

As stated before, norms can be viewed as security policies. While in SecureUML it is possible to define the *permissions* a *user* has, i.e., the constraints that a *user*, in a given *role*, must fulfill to perform *actions* over the system resources, in NormML we extend the SecureUML language to be possible to define the *norms* an *entity* must obey, i.e., to be possible to describe the set of *actions* that the *agents*, *roles*, *agents playing roles* or *groups of agents* in a given context (*organization* or *environment*) are *obliged*, *permitted* or *prohibited* to execute conditioned by the execution of other *actions* and the achievement of *dates* and *states*. The language also makes possible the definition of *sanctions*, i.e., *rewards* and *punishments*, to be applied in case of fulfillment or violation of the norms.

The metamodel of the current version of NormML extends the SecureUML metamodel by including the following new elements: (i) *Norm* (to model norms); (ii) *Agent* (to represent agents whose behavior is being restricted by the norm); (iii) *Organization* and *Environment* (to model contexts and groups of agents); (v) *NormConstraint*, *Before*, *After*, *Between*, *If*, *Date*, *Operand*, and *Value* (to describe activation constraints); (vi) *AgentAction*, *Message*, *Protocol*, *Belief*, *Goal* and *Plan* (to represent new system's resources whose access are controlled by the norms); (vii) *AtomicSend*, *AtomicReceive*, *AtomicAchieve*, *AtomicEnter*, *AtomicLeave*, *AtomicCancel*, *AtomicCommit* and a set of new *composite actions* (to model norms that restrict the execution of actions that access the new resources); and (viii) *Sanction*, *Punishment* and *Reward* (to model rewards and punishments).

Note that the NormML metamodel is a non-conservative extension of the SecureUML metamodel since (i) some metaclasses were redefined and consequently some relationships were modified, (ii) some attributes of metaclasses were eliminated because they were not being used and (iii) some invariants were discarded since they were not applied to the metaclasses of the new metamodel. For instance, the *User* metaclass and all the attributes of the metaclasses of SecureUML were removed, and the *Permission* and the *AutorizationConstraint* metaclasses were replaced by the *Norm* and *NormConstraint* metaclasses. Because of that, some relationships were also changed, for example, the *ConstraintAssignment* relationship between the *Permission* and the *AutorizationConstraint* metaclasses defined in SecureUML was replaced by the *NormConstraintAssignment* relationship between the *Norm* and *NormConstraint* metaclasses in NormML. Appendix A points out the main differences between the metamodels.

A norm corresponds to an instance of the NormML metamodel, i.e., it is defined by instantiating several metaclasses and their relationships. Figure 3.2 presents an overview of the NormML metamodel and its main metaclasses. In the next sections, we present the complete NormML metamodel by focusing on the definition of the main elements that compose a norm.

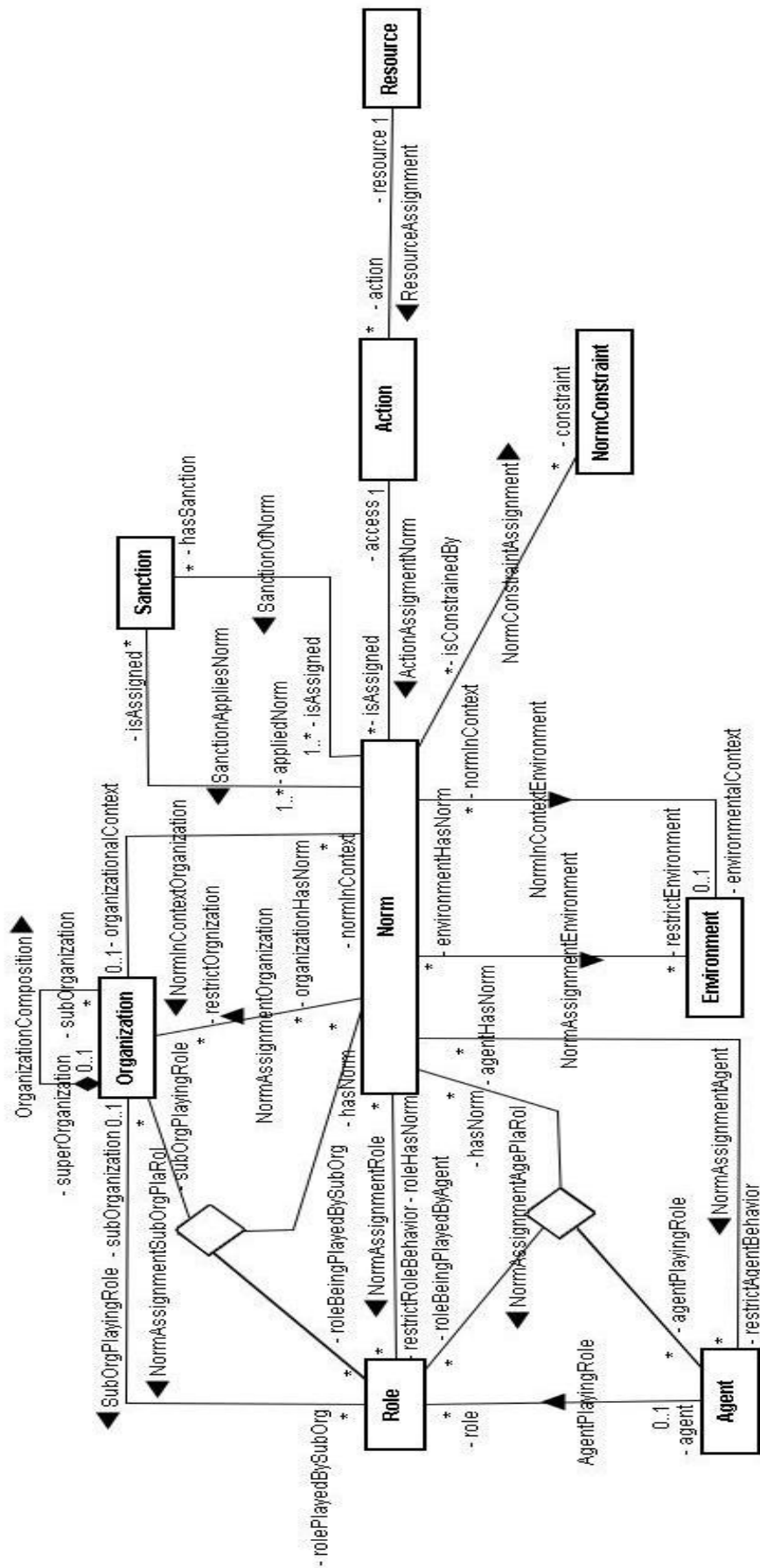


Figure 3.2 The NormML main metaclasses of the NormML metamodel

3.2.1 Deontic concept

A norm can be either an obligation (represented by the metaclass *NormObligation*), a permission (represented by the metaclass *NormPermission*) or a prohibition (represented by the metaclass *NormProhibition*), as illustrated in Figure 3.3. They correspond to the deontic operators that define the deontic concept of the norm. Thus, to describe a prohibition to an entity, for instance, the *NormProhibition* metaclass must be instantiated.

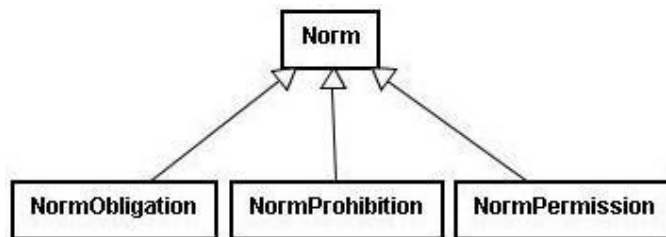


Figure 3.3 The NormML metaclasses used to define the deontic concepts of a norm

3.2.2 Involved entities

A norm can be described to regulate the behavior of: (i) *agents*; (ii) all agents that play a given *role*; (iii) a specific agent when it is playing a given role; or even (iv) a group of agents that are part of an *organization*¹ or an *environment*. Figure 3.4 depicts the part of the NormML metamodel to be used to define the entities whose behavior is being regulated. Such part should be used as follows:

- To regulate the behavior of an agent it is necessary to instantiate the *Agent* metaclass;
- To regulate the behavior of all agents that play a given role it is necessary to instantiate the respective *Role* metaclass;

¹ In our work, we do not make any distinction among the definition of group, team and organization.

- To regulate the behavior of a specific agent when it is playing a given role it is necessary to instantiate the *Agent* and the *Role* metaclasses;
- To regulate the behavior of all agents that play roles in an organization it is necessary to instantiate the *Organization* metaclass;
- To regulate the behavior of all agents that play roles in an sub-organization while such sub-organization is playing a role in its super-organization it is necessary to instantiate the *Organization* metaclass of the respective sub-organization and the *Role* metaclass; and
- To regulate the behavior of all agents that inhabit an environment it is necessary to instantiate the *Environment* metaclass.

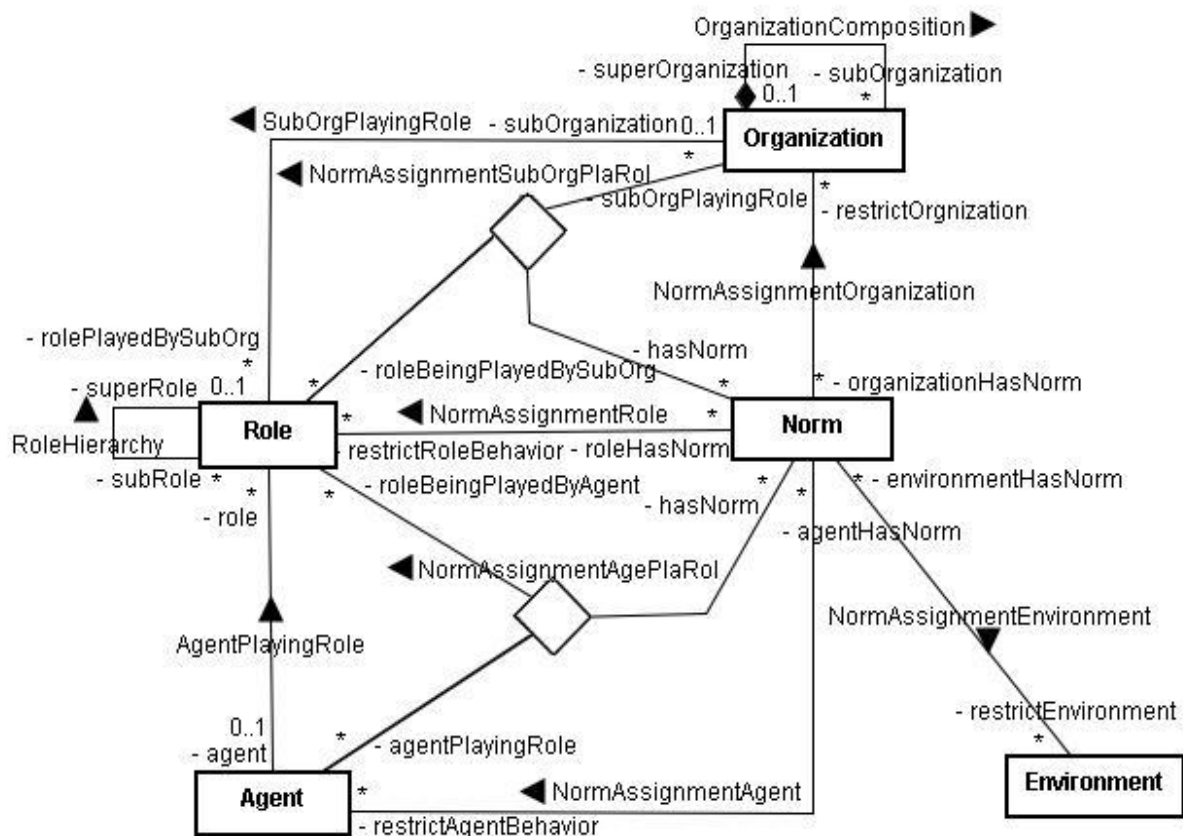


Figure 3.4 The NormML metaclasses used to define the involved entities of a norm

Thus, to model a norm that, for instance, states a prohibition to all agents that play a given role, the *NormProhibition* metaclass, the *Role* metaclass and the *NormAssignmentRole* relationship must be instantiated.

3.2.3 Actions

NormML inherits four resource kinds from SecureUML: *Attribute*, *Method*, *Entity* and *AssociationEnd*. It extends the set of resources with: *Agent*, *Role*, *Organization*, *Environment*, *AgentAction*, *Message*, *Protocol*, *Belief*, *Goal* and *Plan*. Figure 3.6 shows the NormML resources and its relations.

As in SecureUML, each resource kind has a set of actions that can be used to control the access to the resource as illustrated in Figure 3.5. In Table 3.1 we describe which actions are used to restrict the access to each resource, where underlined actions are composite actions.

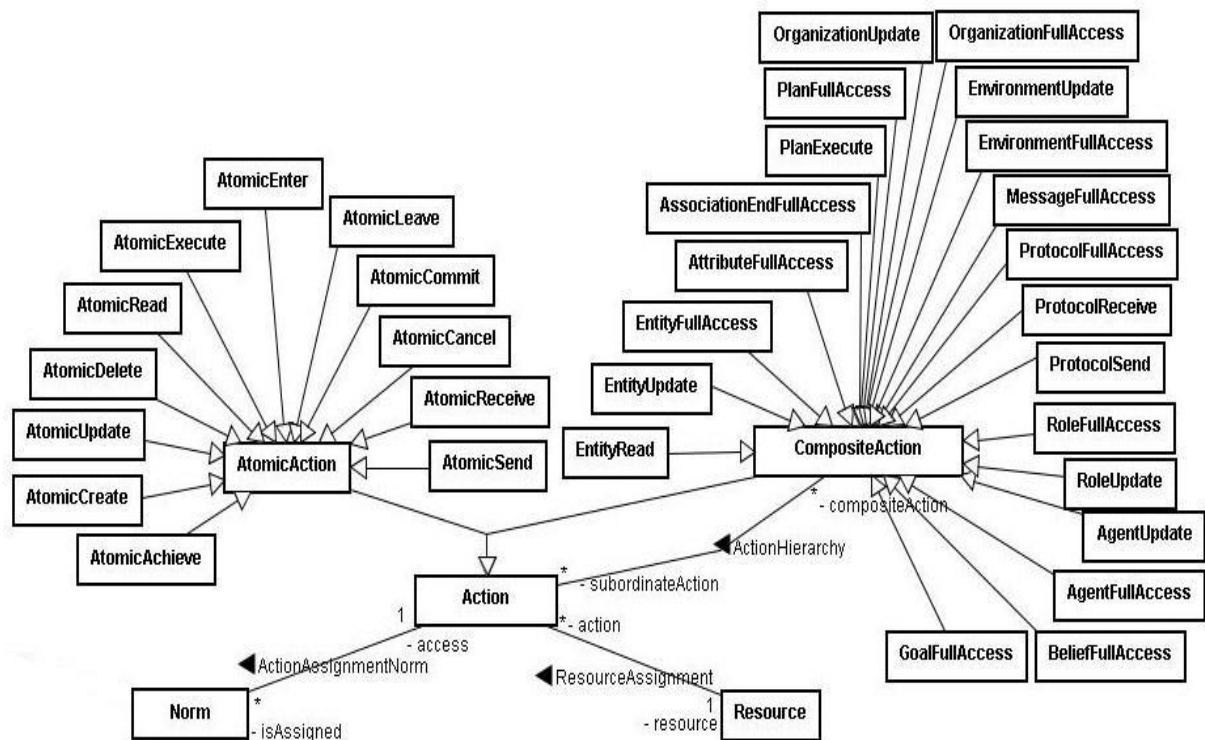


Figure 3.5 The NormML metaclasses used to define the action of a norm

Resource	Actions
Entity	create, <u>read</u> , <u>update</u> , delete, <u>full access</u>
Attribute	read, update, achieve, <u>full access</u>
Method	execute
AssociationEnd	read, update, <u>full access</u>
Agent	create, delete, <u>update</u> , <u>full access</u>
Role	create, delete, commit, cancel, <u>update</u> , <u>full access</u>
Organization	create, delete, enter, leave, <u>update</u> , <u>full access</u>
Environment	create, delete, enter, leave, <u>update</u> , <u>full access</u>
AgentAction	execute
Message	receive, send, <u>full access</u>
Protocol	create, delete, <u>receive</u> , <u>send</u> , <u>full access</u>
Belief	create, delete, update, <u>full access</u>
Goal	achieve, commit, cancel, <u>full access</u>
Plan	create, delete, update, <u>execute</u> , <u>full access</u>

Table 3.1 Resources and their actions

The composite actions are composed of other atomic or composite actions, according to the relations between the resources. In Appendix B a mapping between the composite actions and its subordinate atomic actions is described.

By instantiating the *Norm*, *Action* and *Resource* metaclasses and the *ActionAssignmentNorm* and *ResourceAssignment* relationships, it is possible to model norms that define different ways of restricting the access to different resources. For instance, in the case of restrictions applied to the resource that defines the *actions of agents* (*AgentAction* metaclass), the behavior that must be used to restrict the access to such resource is *the execution of the action* (*AtomicExecute*). Note that *AgentAction* is the resource and *AtomicExecute* is the action being used to control or restrict the access to the resource. In other to provide another example, consider the need for restricting the access to a given messages (*Message* metaclass). In such case, three different access control can be defined: control the access to (i) the *sending* of the message (*AtomicSend*), (ii) the *receiving* of the message (*AtomicReceive*) or (iii) the *full access* (*send+receive*) of the message (*MessageFullAccess*).

3.2.4 Activation constraints

NormML allows for the specification of the time period that a norm is active based on the execution of actions and based on the definition of dates and predicates (i.e., values associated with attributes, beliefs and goals), as shown in Figure 3.7. The activation period of a norm corresponds to the period when the agent must fulfill the norm.

The activation constraints are represented by the metaclass *NormConstraint*. If a norm is conditioned by a *Before* clause, it means that the norm is active before the execution of the action and/or the achievement of the date described in the *Before* clause. If a norm is conditioned by an *After* clause, it means that the norm is active only after the execution of the action and/or the achievement of the date described in the *After* clause. In the case of a *Between* clause, the norm is only active during the period delimited by two actions or dates.

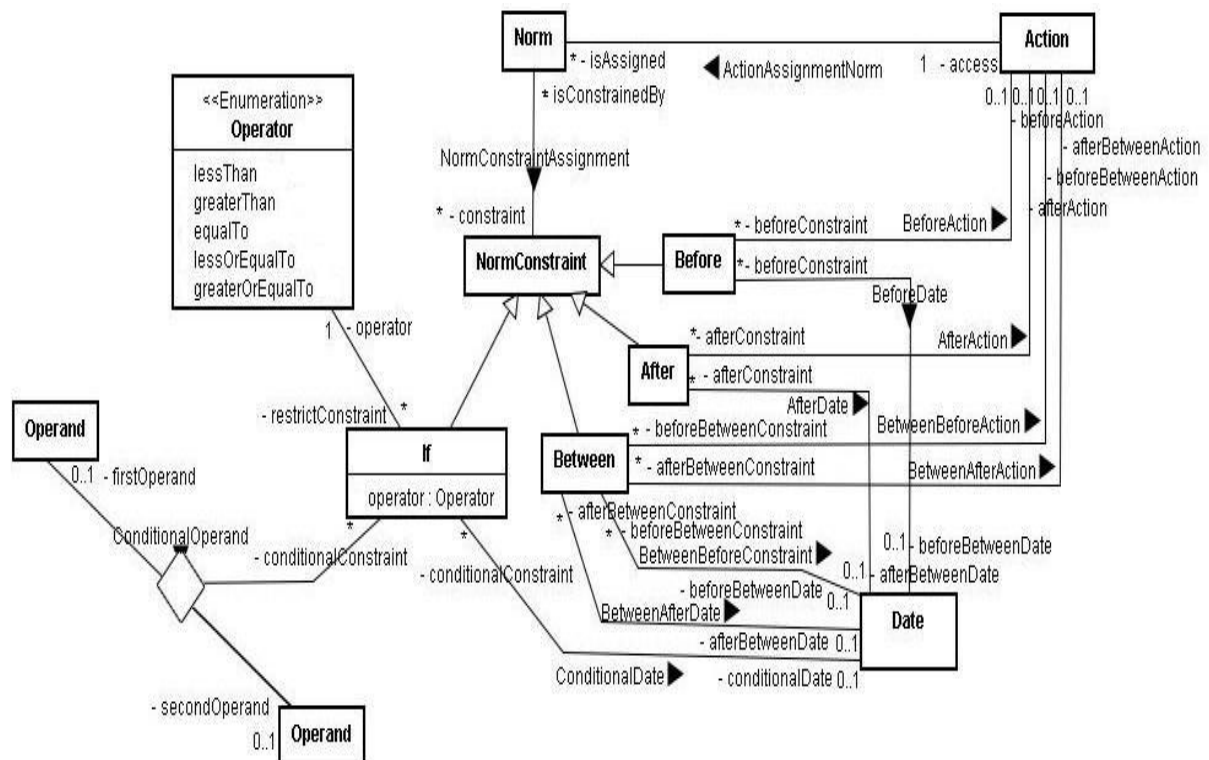


Figure 3.7 The NormML metaclasses used to define the activation constraints of a norm

The *If* constraint has an *operator* attribute that defines the range of values that will activate the norm. The operator of the *If* constraint together with the date or

operands associated with the constraint compose the period when the norm is active. Operands can be an *Attribute*, a *Belief*, a *Goal* or a *Value* (see Figure 3.8). The *ConditionalOperand* relationship must be read from the *firstOperand* to the *secondOperand*. Note that the operand *Value* can only be used as a *secondOperand*.

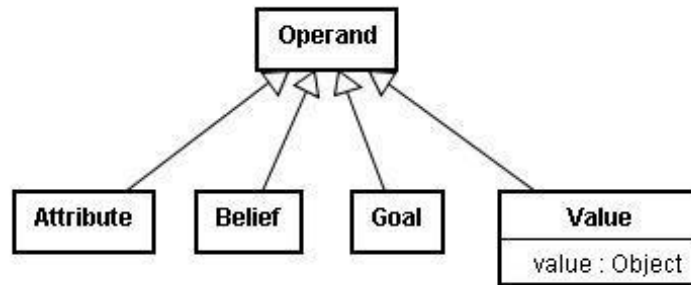


Figure 3.8 The NormML metaclasses used to define the operands of an *If* constraint of a norm

In the case of a norm conditioned by an *If* clause, the norm is activated when: (i) the date described in the *If* clause is achieved; or (ii) the relation between the operands described in the *If* clause becomes true, i.e. the attribute, belief or goal described in the *firstOperand* compared with the attribute, belief, goal or value in the *secondOperand* respects the operator described in the *If* clause.

When an *If* constraint is associated with a date the content of the attribute operator must be “*equalTo*”. The same is valid when the *firstOperand* is a *Goal*. When an *If* constraint is associated with an *Attribute* or a *Belief* as its *firstOperand*, any value of the *Operator* enumeration class can be assumed by the operator attribute of the *If* constraint.

As a result, to model a norm that, for instance, states a prohibition that is activated when an attribute achieves a particular value, the *NormProhibition*, *If*, *Attribute* and *Value* metaclasses, and the *ConditionalOperand* and the *NormConstraintAssignment* relationships must be instantiated.

3.2.5 Sanctions

NormML supports the description of sanctions (*Sanction* metaclass) for the norms, as shown in Figure 3.9. A sanction may be a reward applied when the norm is fulfilled (by instantiating the metaclass *Reward*) or a punishment applied when the norm is violated (by instantiating the metaclass *Punishment*). A sanction activates other norms (represented by the *SanctionAppliesNorm* relationship) to restrict the behavior of an entity that can be the one that has fulfilled/violated the norm or another entity that is the one responsible to apply the reward or punishment.

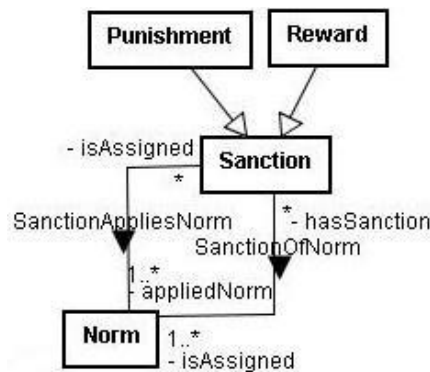


Figure 3.9 The NormML metaclasses used to define the sanctions of a norm

For instance, in case an agent violates an obligation, another norm can be activated to prohibit the agent from executing a particular action. In order to model that, the *Punishment*, *NormObligation* and *NormProhibition* metaclasses must be instantiated. A *SanctionOfNorm* relationship must exist between the *NormObligation* and the *Punishment* instances and a *SanctionAppliesNorm* relationship must be represented between the *Punishment* and the *NormProhibition* instances.

3.2.6 Context

NormML makes possible the definition of norms in two different contexts, as illustrated in Figure 3.10: *Organization* and *Environment*. Organizations (and sub-organizations) define roles played by agents or sub-organizations, and both

organizations and agents inhabit environments. Thus, to describe a prohibition in the context of an organization (or an environment), the *NormProhibition* metaclass, the *Organization* (or *Environment*) metaclass and the *NormInContextOrganization* (or *NormInContextEnvironment*) relationship must be instantiated.

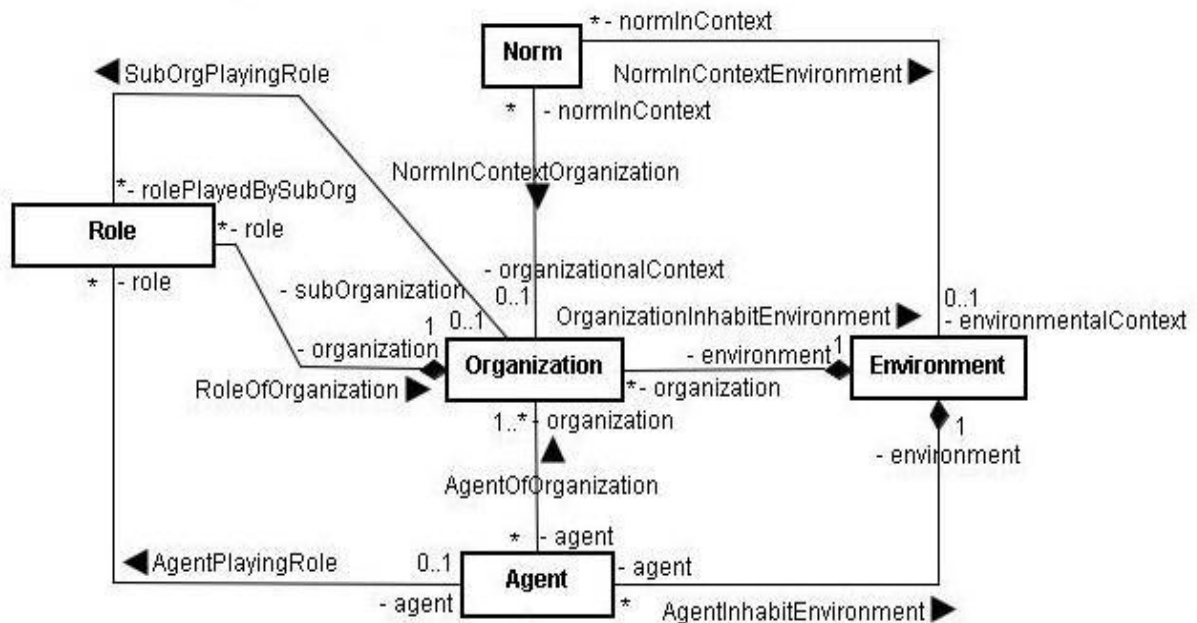


Figure 3.10 The NormML metaclasses used to define the context of a norm

3.2.7 Modeling norms with NormML

In order to exemplify the use of NormML to model the norms of a MAS, consider the norms that govern the simplified version of a *web store* presented in Section 3.1.

N1 (Figure 3.11) states a permission (*deontic concept*) to the sellers (*involved entities*) of the organization *WebStore* (*context*) to update (*action*) the price of the goods (resource of the action) before they receive from the manager the message of opens for sale (*activation constraint*).

Norm N2 (Figure 3.12) applies an obligation (*deontic concept*) to the sellers of the organization *WebStore* (as norm N1) to delete the good's advertisement (*action*) if the stock of the good is empty (*activation constraint*).

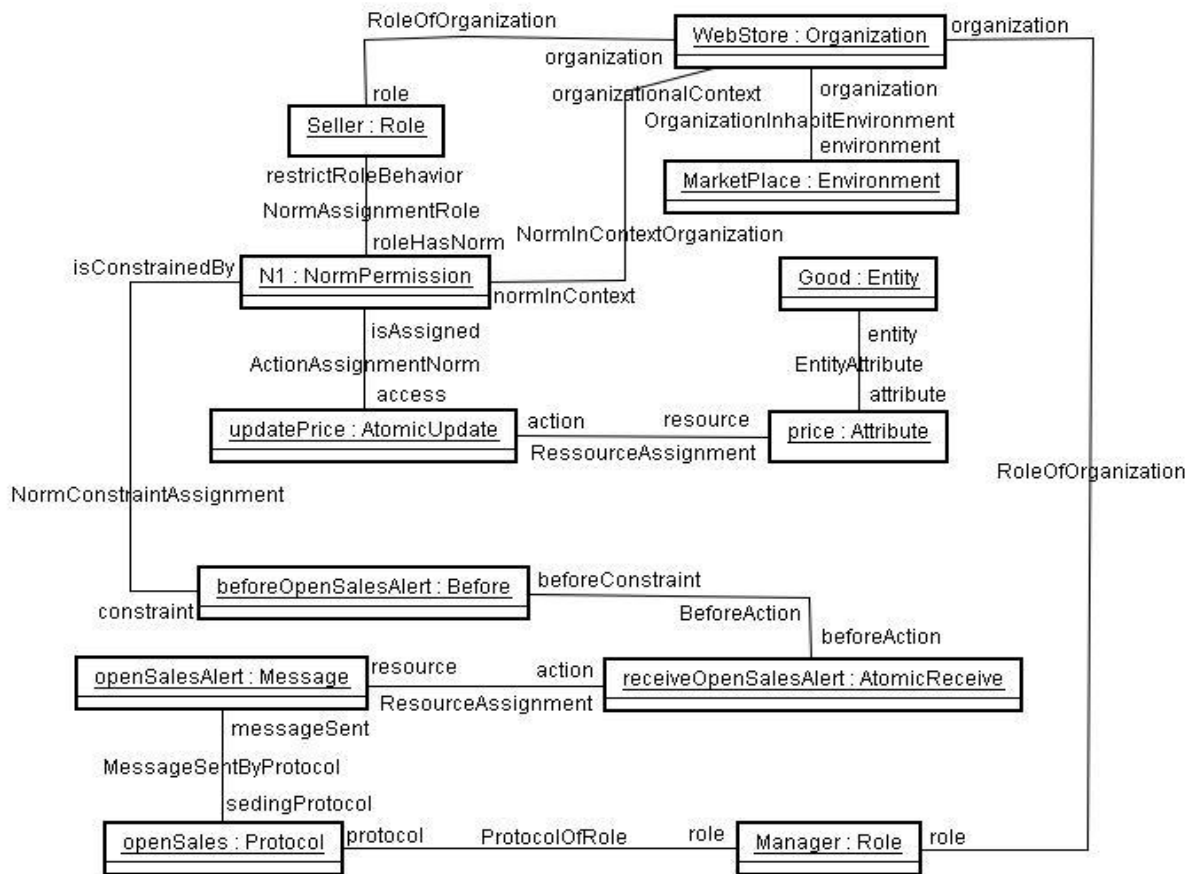


Figure 3.11 Norm N1

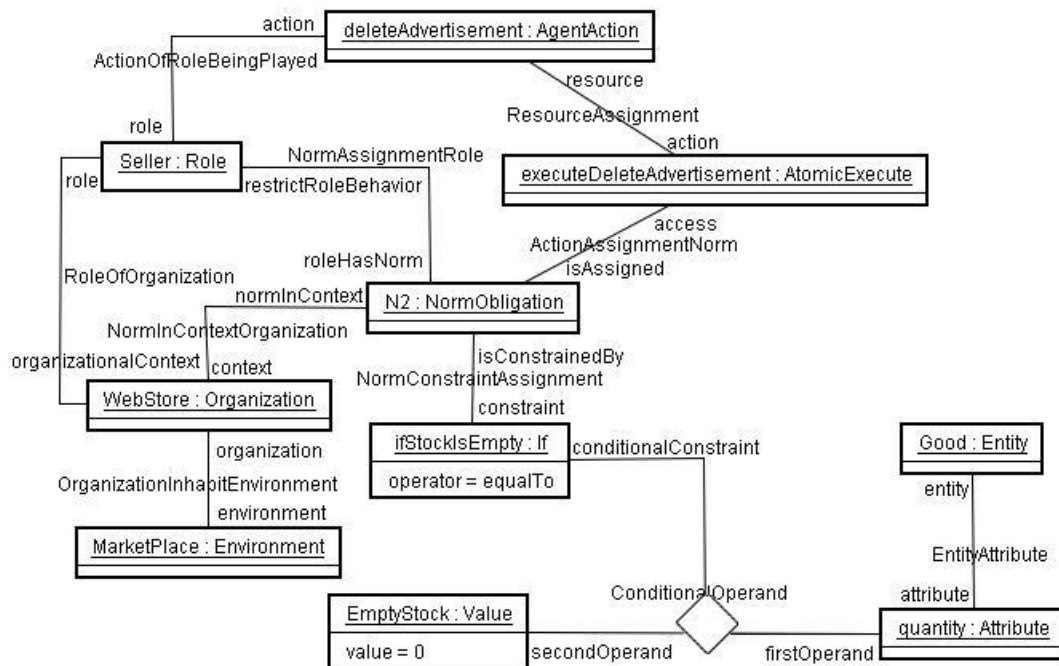


Figure 3.12 Norm N2

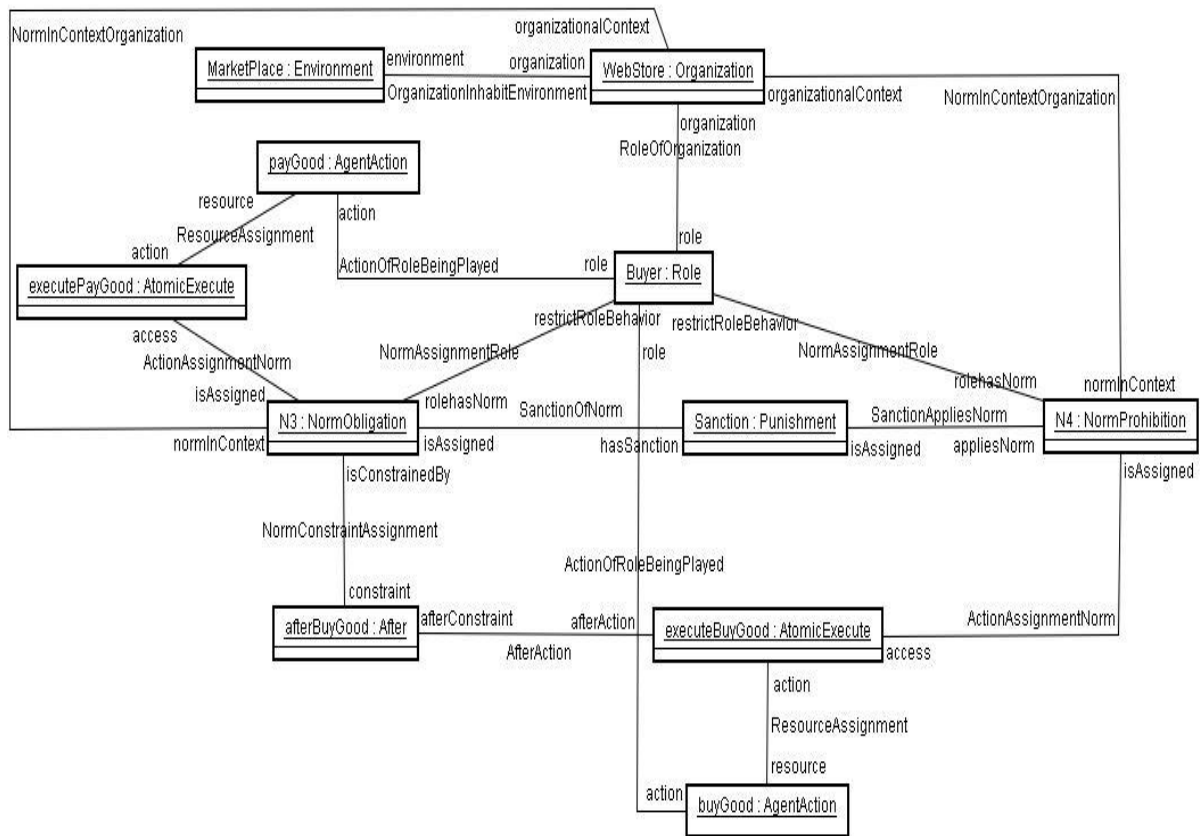


Figure 3.13 Norms N3 and N4

N3 also states an obligation (*deontic concept*) to the buyers of the organization WebStore (*involved entity*) to pay for the good (*action*) after the given buyer buy it (*activation constraint*). Norm N3 applies a punishment (*sanction*) that is another norm too (norm N4). If a buyer violates N3, N4 states to the given buyer (*involved entity*) a prohibition (*deontic concept*) to buy goods (*action*). Figure 3.13 shows the model of norms N3 and N4.

3.3 THE WELL-FORMEDNESS RULES OF NORMML METAMODEL

After modeling the norms of a MAS, they should be validated. The process of validating a norm encompasses two steps. First, the norm, as an instance of the NormML metamodel, is checked according to the invariants of the metamodel. Not all the norms that can be instantiated from the metamodel are well-formed. The

invariants check if the norm is well-formed according to the restrictions of the metamodel elements.

The current version of NormML has a set of operations described in OCL to check the invariants of the norms models. Below we describe some examples of well-formedness rules of the NormML metamodel. Those were chosen since they represent rules that are related to the specification of the norms themselves and discuss some of the representations of the main elements of a norm.

- **WFR1:** *A norm must restrict the behavior of an Agent, a Role, an Agent playing a Role, an Organization, a Sub-organization playing a Role or an Environment.*

context Set(Norm)

inv InvolvedEntitiesNotNull:

```
self-> forAll(n:Norm|if(if((n.restrictAgentBehavior)->isEmpty()) then(((n.restrictRoleBehavior)->notEmpty()) or ((n.agentPlayingRole)->notEmpty()) or ((n.restrictOrganization)-> notEmpty()) or ((n.subOrgPlayingRole)->notEmpty()) or ((n.restrictEnvironment)-> notEmpty())) else (if((n.restrictRoleBehavior)->isEmpty()) then(((n.restrictAgentBehavior)-> notEmpty()) or ((n.agentPlayingRole)->notEmpty()) or ((n.restrictOrganization)->notEmpty()) or ((n.subOrgPlayingRole)->notEmpty()) or ((n.restrictEnvironment)->notEmpty())) else(if((n.agentPlayingRole)->isEmpty()) then(((n.restrictAgentBehavior)-> notEmpty()) or ((n.restrictRoleBehavior)->notEmpty()) or ((n.restrictOrganization)->notEmpty()) or ((n.subOrgPlayingRole)->notEmpty()) or ((n.restrictEnvironment)->notEmpty())) else (if((n.restrictOrganization)-> isEmpty()) then(((n.restrictAgentBehavior)-> notEmpty()) or ((n.restrictRoleBehavior)->notEmpty()) or ((n.agentPlayingRole)->notEmpty()) or ((n.subOrgPlayingRole)->notEmpty()) or ((n.restrictEnvironment)-> notEmpty())) else (if((n.subOrgPlayingRole)-> isEmpty()) then(((n.restrictAgentBehavior)-> notEmpty()) or ((n.restrictRoleBehavior)->notEmpty()) or ((n.agentPlayingRole)->notEmpty()) or ((n.restrictOrganization)->notEmpty()) or ((n.restrictEnvironment)->notEmpty()))else (if((n.restrictEnvironment)->isEmpty()) then(((n.restrictAgentBehavior)-> notEmpty()) or ((n.restrictRoleBehavior)->notEmpty()) or ((n.agentPlayingRole)->notEmpty()) or ((n.restrictOrganization)->notEmpty()) or ((n.subOrgPlayingRole)->notEmpty())))))))) then(true)else(false)endif)endif)endif)endif)endif)endif)endif
```

This rule concentrates on the relationships that define the *involved entities* of a norm. A norm must restrict the behavior of at least one entity.

- **WFR2:** *An AtomicExecute action must be related to a Method or an AgentAction resource.*

context Action

inv AtomicExecuteCorrectAccess:

if((self.oclIsTypeOf(AtomicExecute)) and

((self.resource.oclIsTypeOf(Method)) or (self.resource.oclIsTypeOf(AgentAction))))

then(true)else(false)endif

As explained in Section 3.2.3, each *action* can only be used to access a specific group of resources. The WFR2 matches the *AtomicExecute* action to the *Method* or the *AgentAction* resources.

- **WFR3:** *The action to be executed by an entity that is defined in the before clause of a Between cannot also be defined in the after clause of such Between to be executed over the same resource.*

context Between

inv BetweenDefinesPeriodOfTime:

if((self.beforeAction=self.afterAction) and

(self.beforeAction.resource=self.afterAction.resource))

then(false)else(true)endif

This rule address the relationships between the *Between* and the *Action* metaclasses, used to define one kind of *activation constraint* of a norm. If the actions in the before of a *Between* and in the after of a *Between* are the same and are related to the same resource, this situation does not constitute a time period, but a moment in the time. So it is impossible to configure a period to the norm be active.

- **WFR4:** *A Reward to an entity cannot apply a NormProhibition or a NormObligation to the same entity.*

context Sanction

inv CorrectReward:

if ((self.oclIsTypeOf(Reward) and

```

((self.appliedNorm.ocIsTypeOf(NormProhibition) or
self.appliedNorm.ocIsTypeOf(NormObligation)) and
(((self.appliedNorm.restrictAgentBehavior)-> intersection(self.isAssigned.restrictAgentBehavior))-
>NotEmpty()) and ((self.appliedNorm.restrictRoleBehavior)->
intersection(self.isAssigned.restrictRoleBehavior))->NotEmpty()) and
((self.appliedNorm.agentPlayingRole)-> intersection(self.isAssigned.agentPlayingRole))->NotEmpty())
and ((self.appliedNorm.restrictOrganization)-> intersection(self.isAssigned.restrictOrganization))-
>NotEmpty()) and ((self.appliedNorm.subOrgPlayingRole)->
intersection(self.isAssigned.suborgPlayingRole))->NotEmpty()) and
((self.appliedNorm.restrictEnvironment)-> intersection(self.isAssigned.restrictEnvironment))-
>NotEmpty())) then(false)else(true)endif

```

This rule guarantees that if the *sanction* of a norm is a *Reward*, it cannot define a prohibition or an obligation to the same entity because this situation does not represent an incentive that is the main purpose of a reward.

- **WFR5:** *A norm must be defined in the context of an Organization or an Environment and cannot be defined in the scope of both at the same time.*

```

context Set(Norm)
inv ContextNotNull:
self-> forAll(n:Norm|if(if((n.organizationalContext)->isEmpty())
then((n.environmentalContext)->notEmpty())
      else (if((n.environmentContext)->isEmpty())
            then ((n.organizationalContext)->notEmpty()))
            then(true)else(false)endif)endif)endif

```

This rule concentrates on the relationships that define the *context* of a norm. Every norm must belong to one context (organization or environment).

All the well-formedness rules of the NormML metamodel are described in Appendix C.

3.4 CHECKING FOR CONFLICTS

After verifying the well-formedness of the norms, the second step to validate the norms is to check if there are conflicts among them. Our language provides a set of operations described in OCL to check for conflicts between the norms in NormML. The norms are checked in pairs by considering the situations presented in the following subsections, if one of the cases analyzed in each item (or subsection) returns “true”, so the analyses continues to the next item (or subsection), see Figure 3.14 below.

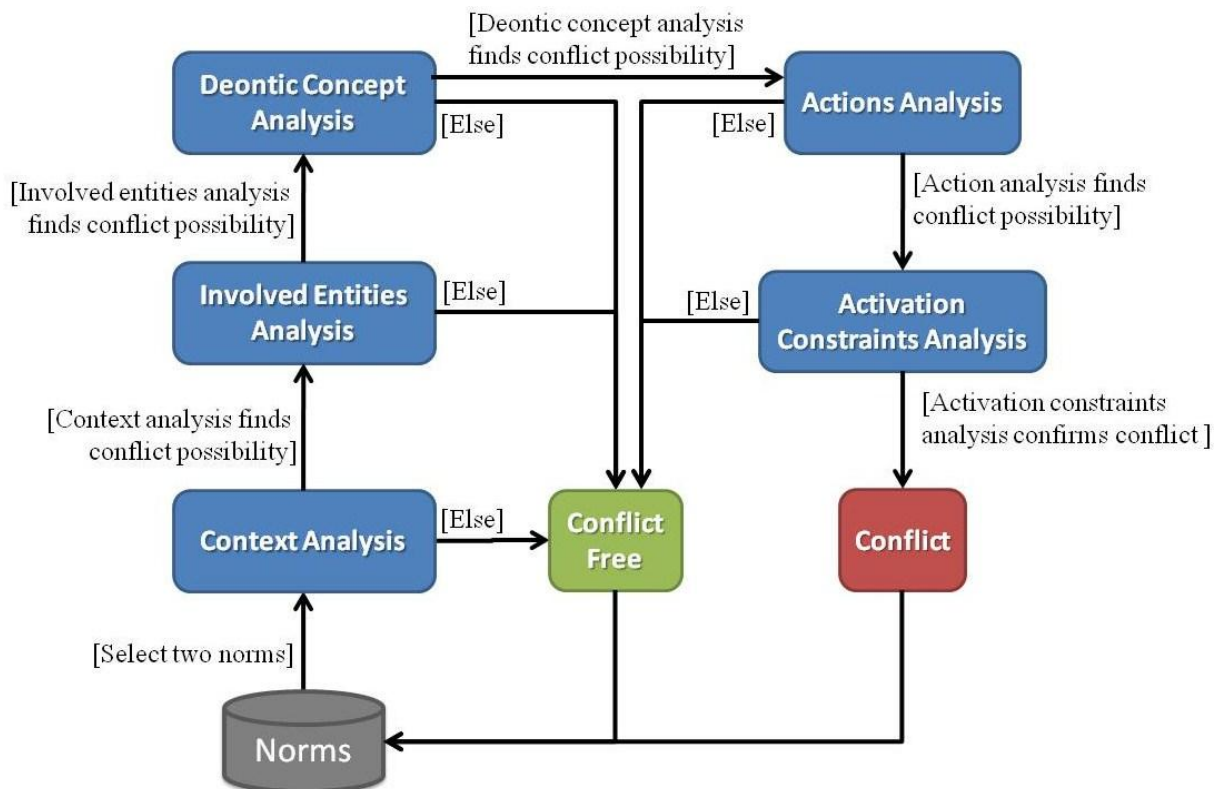


Figure 3.14 NormML check for conflicts analysis

As stated before, a norm in NormML is composed of the following elements: *deontic concept*, *involved entities*, *actions*, *activation constraints*, *sanctions* and *context*. The operations, as the analysis, follow a top-down approach since they start by checking (i) if the norms are defined in the same or related context, (ii) if they apply to the same or related entities, (iii) if they state related deontic concepts, (iv) if they restrict the same or related actions and, finally, (v) if they are active in periods that intersect. The operation below describes our approach calling the specific operations that analyze the main elements of two norms.

```

context Set{Norm}:: checkingForConflicts(n1:Norm,n2:Norm):Boolean
body: if(n1.relatedContexts(n2))
    then(if(n1.relatedEntities(n2))
        then(if(n1.deonticConceptConflicts(n2))
            then(if(n1.relatedActions(n2))
                then(if(n1.activationConstraintsIntersects(n2))
                    then(true)
                    else(false)endif)
                else(false)endif)
            else(false)endif)
        else(false)endif)
    else(false)endif
else(false)endif

```

In order to exemplify the checking for conflicts in sections 3.4.1 to 3.4.6, let's consider norms N5 and N6 of the simplified web store case.

- **N5:** Buyers are prohibited, in the context of the organization WebStore that inhabits the environment MarketPlace, to return a good it has bought.
- **N6:** Buyers are permitted, in the context of the organization WebStore that inhabits the environment MarketPlace, to return a good it has bought between the period of exchange (e.g. between 01/03/2011 and 31/03/2011).

3.4.1 Context analysis

While checking for conflicts between two norms, the first element to be analyzed is the *context* of the norms. If the contexts of the norms are not related, there is no need to keep looking for conflicts because the norms defined in different contexts are not related to each other, and thus cannot conflict. For instance, a given agent can be prohibited to execute an action in a context and permitted to execute the same action in another context. It is important to check for conflicts: (i) if the norms are defined in the same context; (ii) if one norm is defined in the context of an environment, and the other in the context of an organization that inhabits such an

environment; and (iii) if one norm is defined in the context of an organization and the other in the same hierarchy of organizations.

E.g. of case (i): Both N5 and N6 are defined in the context of the organization WebStore that inhabits the environment MarketPlace. Therefore, it is important to check if these norms are in conflict. The operation below is able to check if two norms are in the same organizational context.

context Set{Norm}::sameOrganizationalContext(n1:Norm,n2:Norm):Boolean

body: n1.organizationalcontext = n2.organizationalcontext

3.4.2 Involved entities analysis

The second element to be analyzed is the *involved entities* of the norms. If the entities of the norms are not related, i.e., if they apply to different entities, they cannot be in conflict. Thus, it is necessary to check for conflicts: (i) between norms applied to the same entity; (ii) between a norm defined to a role and a norm defined to an agent or a sub-organization that can play that role; (iii) between norms applied to different roles played by the same agent or sub-organization; (iv) between norms applied to roles in the same hierarchy of roles; and (v) between the norms of an organization and norms of roles, agents and sub-organizations of this organization.

E.g. of case (i): N5 and N6 are restricting the behavior of the same role (buyer) of the organization WebStore. Since both norms are defined in same context and applied to the same subject, these two norms can be in conflict. The operation below can be used to check if two norms restrict the behavior of the same roles.

context Set{Norm}::restrictSameRoleBehavior(n1:Norm,n2:Norm):Boolean

body: (n1.restrictRoleBehavior->intersection(n2.restrictRoleBehavior))->notEmpty()

3.4.3 Deontic concept analysis

After the verification of the involved entities, the next element to be investigated is the *deontic concept* of the norms. Two norms may be in conflict if: (i) one norm states a permission and another states a prohibition; (ii) one norm states an obligation and another states a prohibition; and (iii) one norm states a permission and another one states an obligation in the period the permission is not activated.

Moreover, a special case needs to be considered when both norms state an obligation or both norms state a prohibition to related actions, this fourth (iv) case will be explained in more details in Section 3.4.4.

E.g. of case (i): N5 states a prohibition and N6 states a permission applied to the same subject executing in same contexts. The operation below checks if two norms define deontic concepts that may characterize a conflict defined in cases (i), (ii) and (iii) above.

```

context Set{Norm}::checkDeonticConcept(n1:Norm,n2:Norm):Boolean
body: if((n1.ocIsTypeOf(NormProhibition))and(n2.ocIsTypeOf(NormObligation)))
    then(true)
    else(if((n1.ocIsTypeOf(NormProhibition))and(n2.ocIsTypeOf(NormPermission)))
        then(true)
        else(if((n1.ocIsTypeOf(NormObligation))and(n2.ocIsTypeOf(NormPermission)))
            then (true)
            else (false) endif) endif) endif

```

3.4.4 Action analysis

After the checking of the deontic concept, the next element to be examined is the *action* of the norms. In case the deontic concepts of the norms are in one of the situations (i), (ii) or (iii) of Section 3.4.3, it is important to check for conflicts if: (i) the actions being regulated by the norms are of the same type on the same resource; and (ii) if the actions being regulated by the norms are of related types (as defined in the dialect action hierarchy) on related resources. In order to exemplify case (ii), consider the case that one norm states an *AgentUpdate* or an *AgentFullAccess* to one *Agent* and the other norm states one of the actions that can be associated with the beliefs, goals, plans or agent actions of the same *Agent*, e.g. an *AtomicAchieve*

on one *Goal* of the *Agent*. Table B.1 of Appendix B describes the complete list of related types of actions according to the dialect action hierarchy.

Moreover, a special case needs to be considered when the situation (iv) of Section 3.4.3 occurs. In this case it is important to verify if the actions being regulated by the norms are semantically opposite and restrict the access of the same resource (see Appendix D to the complete list of semantically opposite actions). An example of this case occurs when, for instance, one norm defines an *AtomicEnter* and the other an *AtomicLeave* to the same *Organizations* or *Environment* and the deontic concepts associated with the norms are both an obligation or both a prohibition. If one norm states an obligation to enter a particular organization or environment and another one states an obligation to leave the same organization or environment, these norms may be in conflict if the period for fulfilled the norms intersects. The same is valid to prohibitions.

E.g. of case (i): N5 and N6 regulate the execution of the same action (return good). Both norms can be in conflict since they are applied to the same subject, executing in same contexts and regulating the same action. The operation below checks if two norms are regulating the same action over the same resource.

```
context Set{Norm}::regulateSameAction(n1:Norm,n2:Norm):Boolean
body: if((n1.access = n2.access) and (n1.access.resource = n2.access.resource))
    then(true) else(false) endif
```

3.4.5 Activation constraints analysis

Finally, two norms may be in conflict: (i) if the periods established by actions and dates of the invariants Before, After, Between and If intersect; (ii) in case of two If conditions, if the values related to the same attribute or belief intersects (e.g.: $x > 10$ and $x = 15$); and (iii) in case of two If conditions, if the values related to the same goal are equal.

E.g. of case (i): N5 and N6 are defined in time periods that intersects since N5 is always activated, i.e., it is not restricted to any condition, and N6 is activated in a

period between two dates. The operation below checks if one of the norms is not constrained by any period of time.

```
context Norm::isNotConstrained():Boolean
body: self.constraint->size()==0
```

Consequently, N5 and N6 are in conflict because both norms are applied to the same subject (buyer), executing in the same context (WebStore), regulating the same action (return good) and defined in time periods that intersects.

In Appendix H we illustrate all the NormML conflict cases implemented by the check for conflicts operations.

3.4.6 Sanctions analysis

The analyses of the norms that are used as sanctions are different from the others norms. The top-down approach is followed in the same way, but they will be only compared to norms that are applied as sanctions of the same norm and are of the same type, i.e., all the rewards that a given norm applies will be checked together two by two, and similarly, all the punishments will be verified two by two. That occurs because the rewards will be activated when the norm is fulfilled and the punishments will be activated when the norm is violated, so the period of activation between the rewards and punishments will not intersect.

The operation below returns all the rewards of a norm, so they can be analyzed.

```
context Norm::getRewards():Bag(Sanction)
body: self.isAssigned->collect(s:Sanction|s.ocIsTypeOf(Reward))
```

Note that there is no need to compare the norms applied as sanctions with the other norms because we can only determine if a conflict between them will occur at run time.

3.5 THE NORMML CONCRETE SYNTAX

The previous sections presented the *abstract syntax* of the normative modeling language NormML, by showing how norm can be modeled by instantiating the metaclasses of the NormML metamodel and how to validate these models by verifying their well-formedness and checking for conflicts between the norms.

In this section we present a *concrete syntax* to NormML that was inspired on the concrete syntax of SecureUML. The aim of the concrete syntax is to represent the graphical models of NormML. Accordingly, it will be denoted by *graphical models* the models M that the system designer sees and works with, and it will be denoted by *abstract models* \overline{M} the object diagrams that represent the models M as instances of the NormML metamodel. As a result, each element of M is mapped to a set of elements in \overline{M} according to the semantics of its graphical representation. Of course, the mapping must satisfy the following property: if M is a well-formed graphical model, i.e., it satisfies all the invariants of the graphical model, then \overline{M} is a well-formed abstract model that satisfies all the invariants of the metamodel.

To illustrate the use of the concrete syntax of NormML to construct a graphical model M , N1, N2, N3, N4, N5 and N6 presented in this chapter will be used. Such graphical model is illustrated in Figure 3.15.

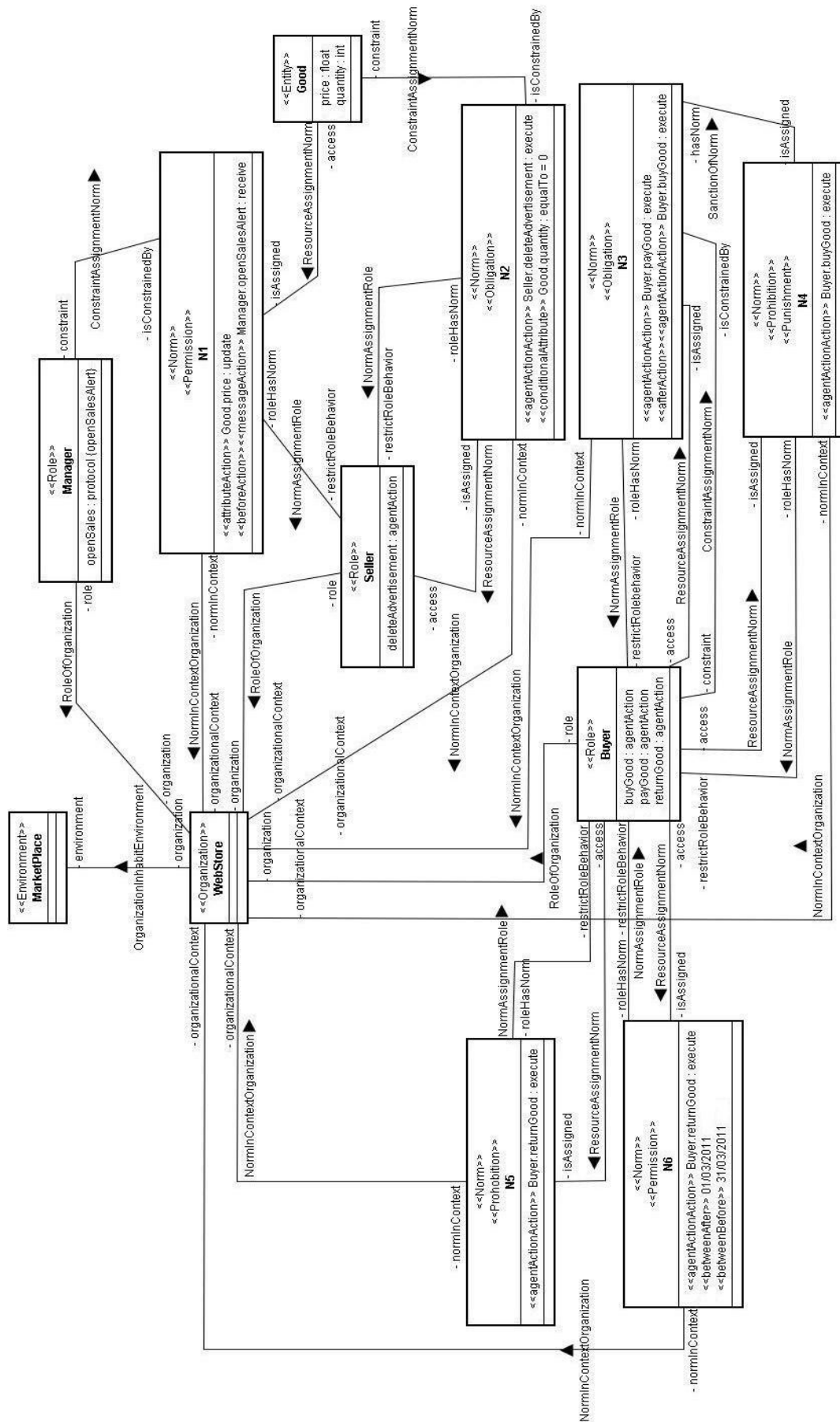


Figure 3.15 Web store norms graphical model

3.5.1 Creating the graphical models

The NormML concrete syntax uses the representation of UML classes adorned with stereotypes to model: agents, roles, environments, organizations, entities and the norms themselves. The stereotypes used refers to the metaclasses of the NormML metamodel used to model these elements. For instance, the role *Manager* is represented by a UML class with a <<*Role*>> stereotype.

The agent classes can have attributes to represent its beliefs, goals, plans and actions, and the role classes can have attributes to represent its goals, protocols and actions to be executed by the agents. For instance, the role *Seller* has a *deleteAdvertisement* attribute of the type <<*agentAction*>> which means that *deleteAdvertisement* is an agent action to be executed by agents that play the role *Seller*.

The entities classes may have attributes and methods in its respective compartments, similar to the entity *Good* that owns a *price* and a *quantity* attributes.

Norms also have stereotypes used to define their *deontic concepts*. For instance, norm *N1* is represented by a UML class with a <<*Norm*>> and a <<*Permission*>> stereotypes. If a norm is a sanction, it also has a stereotype labeling the sanction type: reward or punishment (see norm *N4* in Figure 3.15).

A norm has an attribute called “resource action” that indicates the action being regulated and the resource accessed by such action. The action is identified by a name and by a stereotype. The stereotype is used to restrict the kinds of actions that can be used to manipulate the resource and the name of the action indicates the specific kind in this set. For instance, consider *N1* that regulates the *updating* of the resource *price* of the entity *Good*. The stereotype <<*attributeAction*>> specifies that the attribute refers to an action (*read*, *update*, *achieve* or *full access*) over an *attribute* and the name of the action that is *update* indicates that the action being restricted is the updating of something, in this case, the updating of the attribute *price* of the entity *Good*, which is a resource. Note that it is important to specify the entity that has such attribute before the dot “.” and by the relationship *RessourceAssignmentNorm*, for instance in *N1* the entity is *Good*.

Norms can also have “*constraints*” attributes that represent the *activation constraints* of the norm. Such constraint can be an “action constraint”, a “conditional constraint” or a “date constraint”.

In case of an “action constraint”, the attribute is composed of one stereotype to indicate the kind of the action constraint being defined (before, after or between) followed by the constraint that is a “resource action” attribute. The stereotypes that can be used to indicate the kind of the action constraints are: <<*beforeAction*>>, <<*afterAction*>>, <<*beforeBetweenAction*>> or <<*afterBetweenAction*>>. Norm *N1* in Figure 3.15 defines a <<*beforeAction*>> action constraint restricted by the receiving of a message called *openSalesAlert* to be sent by the role *Manager*. Note that it is important to specify the role that has such message before the dot “.” and by using the relationship *ConstraintAssignmentNorm*.

In case of a “conditional constraint”, the attribute is composed of a stereotype indicating the conditional constraint type followed by the name of an attribute, belief or goal, the name of the operator and the name of another attribute, belief or goal, or a value. Note that it is also important to specify and relate such attribute, belief or goal to the entity that owes it. Norm *N2* in Figure 3.15 illustrates a conditional constraint attribute stating that the attribute *quantity* of *Good* must be *equal to 0*.

In the case of a “date constraint”, the attribute is composed of a stereotype indicating the date constraint type followed by the date. Norm *N6* in Figure 3.15 shows two date constraint attributes *01/03/2011* and *31/03/2011* labeled by the stereotypes <<*afterBetween*>> and <<*beforeBetween*>> respectively, indicating that *N6* is active in the interval between these two dates.

As occurs in the abstract models, *agents* and *organizations* must be related to (i) the *roles* they play (by the relationships *AgentPlayingRole* and *SubOrgPlayingRole*, respectively), (ii) the *organizations* they belong (by the relationships *AgentOfOrganization* and *OrganizationComposition*, respectively), and (iii) the *environment* they inhabit (by the relationships *AgentInhabitEnvironment* and *OrganizationInhabitEnvironment*, respectively). Also, *roles* must be related to the *organizations* they belong by the *RoleOfOrganization* relationship.

A complete list of the graphical model stereotypes and what they represent is described in Appendix E.

3.5.2 Mapping from concrete to abstract syntax

As stated before, each element of M is mapped to a set of elements in \overline{M} according to the graphical representation semantics. The mapping of a model M to a model \overline{M} must occur in order to construct a well-formed model \overline{M} that complies with the abstract metamodel and its well-formedness rules. In Appendix F the complete mapping between the *graphical models* M of the concrete syntax and the *abstract models* \overline{M} of the abstract syntax is traced.

In order to exemplify the transformation of the concrete syntax of a norm to its abstract syntax, let's consider the set of transformation rules below:

- Rule1.** For each *Environment* env of M , insert in \overline{M} an object \overline{env} of the class *Environment*.
- Rule2.** For each *Organization* org of M , insert in \overline{M} an object \overline{org} of the class *Organization*.
- Rule3.** For each *OrganizationInhabitEnvironment* relationship of M between env and org , insert in \overline{M} an *OrganizationInhabitEnvironment* link between \overline{env} and \overline{org} .
- Rule4.** For each *Entity* e of M , insert in \overline{M} an object \overline{e} of the class *Entity* and, for each *Attribute* a of an *Entity* e of M , insert in \overline{M} (i) an object \overline{a} of the class *Attribute* and (ii) an *EntityAttribute* link between \overline{a} and \overline{e} .
- Rule5.** For each *Role* r of M , insert in \overline{M} an object \overline{r} of the class *Role*.
- Rule6.** For each *Protocol* pro of a *Role* r of M , insert in \overline{M} (i) an object \overline{pro} of the class *Protocol*; (ii) a *ProtocolOfRole* link between \overline{pro} and \overline{r} ; (iii) an object \overline{mess} of the class *Message* and a *MessageSentByProtocol* link between \overline{pro} and \overline{mess} for each "sent message" $mess$ of pro ; and (iv) an object \overline{mess} of the

class *Message* and a *MessageReceivedByProtocol* link between \overline{pro} and \overline{mess} for each “received message” *mess* of *pro*.

Rule7. For each *RoleOfOrganization* relationship of *M* between *org* and *r*, insert in \overline{M} a *RoleOfOrganization* link between \overline{org} and \overline{r} .

Rule8. For each *Norm n* of *M* that states a permission, insert in \overline{M} an object \overline{n} of the class *NormPermission*.

Rule9. For each *resource action* attribute *res* of *Norm n* of *M* that is an “attributeAction” and has the action type “update”, must be inserted in \overline{M} (i) an object \overline{act} of the class *AtomicUpdate*; (ii) an *ActionAssignmentNorm* link between \overline{act} and \overline{n} ; and (iii) a *RessourceAssignement* link between \overline{act} and the object \overline{a} which name is equal to *res*.

Rule10. For each *action constraint* attribute *acon* of *Norm n* of *M* that is a “beforeAction” and has a *res* that is a “messageAction” and has the action type “receive”, insert in \overline{M} (i) an object \overline{act} of the class *AtomicReceive*; (ii) an object \overline{bef} of the class *Before*; (iii) a *BeforeAction* link between \overline{act} and the object \overline{bef} ; (iv) a *NormConstraintAssignment* link between \overline{n} and the object \overline{bef} ; and (v) a *RessourceAssignement* link between \overline{act} and the \overline{mess} object which name is equal to *acon*.

Rule11. For each *NormInContextOrganization* relationship of *M* between *org* and *n*, insert in \overline{M} a *NormInContextOrganization* link between \overline{org} and \overline{n} .

Rule12. For each *NormAssignmentRole* relationship of *M* between *r* and *n*, insert in \overline{M} a *NormAssignmentRole* link between \overline{r} and \overline{n} .

Let’s take norm *N1* from Figure 3.15 as an example of mapping with the purpose of constructing the abstract model \overline{M} of Figure 3.11. By using rules 1 and 2, the objects of the Environment *MarketPlace* and the Organization *WebStore* are

created. Rule 3 defines the *OrganizationInhabitEnvironment* relationship between the *WebStore* and the *MarketPlace* and ensures that an organization will always inhabit an environment in \overline{M} .

By using rule 4 the *Good* instance of Entity and the *price* and *quantity* instances of Attribute are created. Rule 4 also creates the relationship *EntityAttribute* between *Good* and its attributes *price* and *quantity* to guarantee that the attribute will belong to its entity in \overline{M} .

Rule 5 generates the Role instances *Manager* and *Seller*, and rule 7 assigns them to the Organization *WebStore* through the relationship *RoleOfOrganization*. With rule 7 it is ensured that each role will belong to an organization in \overline{M} .

Rule 6 guarantees that each message will belong to its protocol and each protocol will belong to its role in \overline{M} . Thus, with rule 6 the *openSales* Protocol is created and related to its Message *openSalesAlert* (by the *MessageSentByProtocol* relationship) and its owner Role (by the *ProtocolOfRole* relationship).

Rule 8 generates the NormPermission's instance *N1* and by rule 9 its atomic update action *updatePrice* is created and related to *N1* (by the *ActionAssignmentNorm* relationship) and to the Attribute *price* (by the *ResourceAssignment* relationship). Then, rule 9 guarantees that the norm will restrict the access of an action over a resource.

By using rule 10 the *beforeOpenSalesAlert* instance of Before and the *receiveOpenSalesAlert* instance of AtomicReceive are created. The *beforeOpenSalesAlert* is related to *N1* by the *NormConstraintAssignment* relationship and the *receiveOpenSalesAlert* is related to *openSalesAlert* by the *ResourceAssignment* relationship. Rule 10 ensures that the before constraint will be related to an action as stated in the WFR56 (see Appendix C).

Finally, rule 11 defines the organizational context of the *WebStore* to *N1* by the *NormInContextOrganization* relationship and rule 12 assigns *N1* to the Role *Seller* by the *NormAssignmentRole* relationship. Rules 11 and 12 comply with the WFR1 and WFR6 respectively (see Appendix C). As a result, the well-formed model \overline{M} of *N1* is created as it was illustrated before in Figure 3.11. The creation of a well-formed model \overline{M} of *N1* is possible because the transformation rules follow all the restrictions of the abstract metamodel and the well-formedness rules as stated above. The same is valid for all the transformation rules of Appendix F.

Note that, the model M of N1 is a well-formed concrete model. When a given model M is not correct according to the concrete syntax, then a not well-formed model \overline{M} will be create. E.g. Rule 3 ensures that the organization *WebStore* will inhabit the environment *MarketPlace* in \overline{M} , but that is only possible because the *OrganizationInhabitEnvironment* relationship between the *WebStore* and the *MarketPlace* is described in M .

Recall that, in our approach, the checking for conflicts between norms using OCL depends on the mapping from graphical models to abstract models (see Appendix F). This is because the operations will not be evaluated on the graphical models, but rather on the corresponding abstract models.

CHAPTER 4: RELATED WORK

Although there are some works, such as the MAS modeling languages AUML, MAS-ML and ANote (Noya and Lucena, 2005) and the MAS methodology MESSAGE (Caire *et al.*, 2002) that do not support the modeling of norms, there are already many others that make possible the modeling of several elements of a norm. From the set of two MAS modeling languages (Danc, 2008, and Wagner, 2003), seven MAS methodologies (Cossentino, 2005, Garcia-Ojeda *et al.*, 2008, Giorgini *et al.*, 2006, Juan *et al.*, 2002, Omicini, 2001, Padgham and Winikoff, 2004, and Zambonelli *et al.*, 2003) and three MAS organization models (Dignum, 2004, Ferber *et al.*, 2009, and Hübner *et al.*, 2002) analyzed, no one is able to model all the properties of the main elements that compose a norm and being described in Section 3.1.

In this chapter we discuss those modeling languages, methodologies and organization models showing how they represent the concepts related to the norms and employ these elements when the designers are modeling the norms (Section 4.1).

In addition to the elements of the norm, another interesting characteristic to be considered when analyzing the modeling languages, methodologies and organizational models is the ability to detect conflicts between the norms of the system at design phase. Thus, Section 4.2 presents an investigation about the support to the checking for conflicts between norms in those works. We compare such works with NormML, the modeling language being proposed in this work.

4.1 MAIN ELEMENTS OF A NORM

- **Deontic concept:** Most modeling languages and methodologies make available the deontic concept of obligation in order to describe the actions that agents must execute. Methodologies such as Secure Tropos (ST) (Giorgini *et al.*, 2006), SODA (Omicini, 2001), Prometheus (Padgham and Winikoff, 2004) and the organization model proposed in MOISE+ (Hübner *et al.*, 2002) only offer the concepts of obligation and permission since they consider that

everything that is not permitted is automatically prohibited. In the ST methodology the concept of obligation can be represented by the delegation relationship and the concept of permission by the ownership and trust relationships. NormML, different from the majority, includes all the three deontic concepts (obligation, permission and prohibition) in the modeling of norms.

- **Involved entities:** All works analyzed propose a way to describe the entities to which the norm applies (elements checked in Table 4.1). The majority provides support to describe a norm for a particular role while others provide support to also describe a norm for other entities. Some works (Cossentino, 2005, Hübner *et al.*, 2002, Juan *et al.*, 2002, and Zambonelli *et al.*, 2003) do not allow the description of norms that apply to a group of individuals. This fact does not imply that the works analyzed do not support the modeling of such entities, however they do not provide ways to apply norms to them. The ST methodology also allows the designer to describe the system itself as an entity and to define norms that can be applied to the system as a whole. By using NormML it is possible to describe norms to individuals, groups of individuals or all the entities of the system.

- **Actions:** All the modeling languages, methodologies and models analyzed provide a way to restrict non-communicative actions. But, the same is not true about communicative actions. In (Hübner *et al.*, 2002, Juan *et al.*, 2002, Omicini, 2001, and Wagner, 2003) the restriction of communicative actions is not available. In ROADMAP (Juan *et al.*, 2002), that is one of the proposed extensions for Gaia, the designer can only restrict the access to objects, roles and protocols of the system. NormML supports the modeling of both kinds of actions, communicative and non-communicative over a large set of resources.

- **Activation constraints:** The works analyzed present several ways to describe the period during while a norm is active, i.e., to describe the restrictions for their activation and deactivation (see more details in Table 4.1). In the MASQ and OperA organizational models it is possible to define the activation of a second norm by the violation or fulfillment of the first norm. The

ST methodology is the only one that does not provide any kind of activation constraint representation since their norms are always active. According to (Molesini *et al.*, 2009), the SODA formalism is still being developed so we cannot affirm the types of restrictions that such methodology will support. By using NormML all the activation constraints of the Table 4.1 can be modeled.

- **Sanctions:** A small number of languages and methodologies consider that norms can be violated, and only a few of them provide a way for describing sanctions. The AORML (Wagner, 2003) language assumes that commitments (or obligations) between entities of the system can be violated, and, as consequence, a sanction should be applied. But the language does not offer a way to describe this sanction. The organizational models OperA (Dignum, 2004), MASQ (Ferber *et al.*, 2009) and MOISE+ consider that norms can be violated, and, excluding MOISE+, they have mechanisms to describe punishments.

The O-MaSE (Garcia-Ojeda *et al.*, 2008) methodology groups norms into two kinds of policies: law policies and guidance policies. Only the guidance policies can be violated but there is not a way to define sanctions for such violations. The Gaia (Zambonelli *et al.*, 2003) and PASSI (Cossentino, 2005) methodologies express norms as organization rules that cannot be violated, and so there is no need to define a sanction mechanism. None of the analyzed languages or methodologies allows the description of rewards in case of the fulfillment of a norm. However, NormML supports the definition of both punishments and rewards.

- **Context:** All languages, methodologies and organizational models define the norms in an organizational context. The AORML language also offers support to express obligations between two agents (as commitments) in the context of an interaction. Besides AORML, methodologies such as PASSI, Prometheus, Gaia and the organizational model OperA also allow the description of norms in such a context. Moreover, in OperA and Gaia it is possible to describe a norm in a context that represents the transition of scenes. Besides describing norms in an organizational context, NormML also provides the environmental context.

Recall that the main elements that compose a norm were found out after investigating fourteen implementation and specification languages used to describe and implement norms (Aldewereld *et al.*, 2006, Cholvy, 1999, Cranefield, 2007, Fornara and Colombetti, 2008, García-Camino *et al.*, 2005, García-Camino *et al.*, 2006, Governatori and Rotolo, 2004, Lomuscio and Sergot, 2004, Lopes-Cardoso and Oliveira, 2010, López y López *et al.*, 2002, López y López, 2003, Silva, 2008, Vasconcelos *et al.*, 2007 and Vigano and Colombetti, 2008).

Our objective while investigating those implementation and specification works was to try to consider all mentioned elements mentioned in order to do a deep investigation of the norms composition to develop a complete normative modeling language. We understand that none of the works presented in this section have as the main purpose of modeling norms as NormML does, what justifies the absence of some elements in their proposals.

Thus, Table 4.1 just summarizes the discussion about the modeling of the main elements of a norm by the related work studied and our proposed modeling language: NormML.

		AML	AORML	Gaia	O-MaSE	PASSI	Prometheus	ROADMAP	ST	SODA	MASQ	MOISE+	Opera	NormML
Deontic Concept	Permission	•	•	•			•	•	•	•	•	•	•	•
	Prohibition	•	•	•	•						•		•	•
	Obligation	•	•	•	•	•	•		•	•	•	•	•	•
Involved Entities	Agent	•	•		•				•	•			•	•
	Role	•	•	•		•	•	•	•	•		•	•	•
	Agent playing role	•			•						•			•
	Groups of individuals	•	•		•		•		•	•	•		•	•
	All in the system				•		•		•	•			•	•
Actions	Communicative Actions	•		•	•	•	•		•		•		•	•
	Non-communicative Actions	•	•	•	•	•	•	•	•	•	•	•	•	•
Activation Constraints	Execution of actions	•		•	•	•	•				•		•	•
	Time intervals	•		•	•	•	•	•			•		•	•
	Achievement of states	•		•	•	•	•				•		•	•
	Temporal aspects	•	•	•	•	•	•				•	•	•	•
	Fulfillment and violation of a norm										•		•	•
Sanctions	Punishment										•		•	•
	Reward													•
Context	Environment													•
	Organization	•	•	•	•	•	•	•	•	•	•	•	•	•
	Interaction	•		•			•		•				•	
	Transition of scene			•									•	

Table 4.1 Main elements of a norm

4.2 CHECKING FOR CONFLICTS

In this section we investigate about the support provided by the analyzed approaches for checking conflicts among norms is presented. First, in Section 4.2.1 we point out how the MAS modeling languages, methodologies and organizational models deal with the norms conflicts, and then, in Section 4.2.2 other approaches are discussed.

4.2.1 Modeling languages, methodologies and organizational models

From works exposed previously, only the AORML modeling language, the ST methodology and the OperA organizational model consider norm conflicts. Table 4.2 compares the analysis done at the verification for conflicts of these approaches to NormML.

The AORML language assumes that there is a normative inconsistency when there is at the same time a permission and a prohibition, or a prohibition and an obligation to the same action. It considers that obligations already have a permission embedded, so there is no conflict in this sense. Although the language considers that conflicts can occur, it does not have an automatic mechanism to detect these conflicts.

The ST methodology defines eight properties to be used for the verification of conflicts in its models, including two for the validation of conflicts between the system's obligations and permissions. The analysis is done only between norms of the same entity when they are defined in the same context. Although norms can be defined in different but related contexts, they do not check conflicts between them. For instance, it is important to check for conflicts between norms defined in the context of an interaction of roles and norms defined in the context of the organization where these roles are being played. In this case the contexts *organization* and *interaction* are related and conflicts between norms must be checked. Moreover, since all the norms have no activation constraints, they do not take this characteristic into account when checking for conflicts. The ST methodology has a tool to graphical

model norms (SecTro, 2011), but the tool do not support the automatic verification of their properties.

		AORML	ST	OperA	NormML
Context analysis					•
Involved Entities	Same entity analysis	•	•	•	•
	Other entities relations analysis				•
Deontic concept	Oposite deontic concepts analysis	•	•	•	•
	Same deontic concept analysis				•
Actions	Same action analysis	•	•	•	•
	Other actions relations analysis				•
Activation constraints analysis		•			•
Sanctions analysis					•
Automatic analysis (by tool)				•	•

Table 4.2 Checking for conflicts analysis

The OperA organizational model allows the automatic verification of conflicts between the norms that apply to a given entity. However, such mechanism does not give support to the checking of conflicts between norms applied to different entity types, i.e., between the norms applied to a group and the norms applied to roles to be played in the group or agents that are executing in such group. In addition, it also does not give support for checking conflicts among norms defined in different contexts and considering different activation conditions.

As shown in Table 4.2 and detailed in Section 3.4, NormML has a set of operations for conflicts verification in a top-down approach that considers the possibility of conflicts between the norms by analyzing each main element that compose the norms, including *context* and *sanctions* that are not considered by the

works presented. In Chapter 7, a tool for automatic execution of this verification is presented.

4.2.2 Other approaches that deal with norm conflicts

Cholvy (1999) addresses the problem of norms consistency and the need for its verification. In Cholvy (1999) a SOL-resolution (Inoue, 1992) was proposed to prove that conflicts exist and identify them. The deontic concept analysis includes both *opposite deontic operators* checking (contradictions) , e.g. if one norm states a permission to perform an action and another norm states a prohibition to perform it, and *same deontic operator* checking (dilemmas) , e.g. if one norm states a obligation to perform an action and another norm states a obligation to not perform it. Different roles played by the same entity and values of predicates are considered during the analysis.

In Oren *et al.* (2008) the authors point out that there are conflicts between obligations and prohibitions, and permissions and prohibitions to the same agent or role to execute actions over the same states. They also consider that there are conflicts between obligations related to states that are mutually exclusive. The norms analyzed in this work do not have any kind of activation constraint.

In Gaertner *et al.* (2007), García-Camino *et al.* (2006), Kagal and Finin (2005), Kollingbaum *et al.* (2008) and Vasconcelos *et al.* (2007) the authors consider that there is a normative conflict when one norm states an obligation or a permission and the other norm states a prohibition on the same agent or role to execute the same action at time intervals that intersect. In Gaertner *et al.* (2007) and Kollingbaum *et al.* (2008) they extend the analysis to actions that are of the same domain. In Kagal and Finin (2005) only communicative actions are mentioned.

In Kollingbaum *et al.* (2007) and Kollingbaum and Norman (2006) the authors point out that there are conflicts between permissions and prohibitions, and inconsistencies between obligations and prohibitions to the same role to execute the same action. They also consider that there are inconsistencies between obligations related to the execution of actions at the same time but they do not emphasize it. The

norms analyzed in this work can have a set of states as activation constraints but they do not consider this in its conflict detection.

None of the works reviewed considers the special case of conflicts between obligations and permissions that may occur when an agent is obliged to execute an action when it has not a permission to do so. Also, none of them consider all the main elements of the norm while checking for conflicts as NormML does.

There are research in the area of norm conflicts that investigate conflicts between the norms and other elements of the system as goals of the system (Modgil and Luck, 2009) or conflicts between deadlines of the norms (Lopes-Cardoso and Oliveira, 2008, Lopes-Cardoso and Oliveira, 2010), but these questions are out of the scope of our work.

CHAPTER 5: EXAMPLE APPLICATION

In this chapter we provide an example of an application in the area of Conference Management to illustrate the use of norms in a MAS. This example was chosen since it has been used by several authors, such as Zambonelli *et al.* (2001), Dignum (2004) and Harmon *et al.* (2008), to illustrate their approaches.

5.1 CONFERENCE MANAGEMENT SYSTEM

In order to exemplify our approach, we describe a simplified version of a *conference management system* of a *local conference*. The local conference is being represented as an organization that inhabits the *conference society* environment. Consider the following conference management process illustrated in Figure 5.1.



Figure 5.1 Conference management process of the local conference

The local conference defines the following roles to be played by the agents: *organizer*, *conference chair*, *website manager*, *reviewer*, *author* and *speaker*. The roles *conference chair*, *website manager* and *reviewer* are sub-roles of the role *organizer*.

The conference chair runs the conference and is responsible for the coordination of the review and publication processes. During the paper submission period, authors can submit papers, and after that the reviewers must review the papers received until the notification deadline. Then, the conference chair communicates the authors about the status of their papers: accepted or rejected.

Authors who had their papers accepted must register on the conference. When registered, the authors guarantee that their accepted papers will be published in the conference proceedings and they can present the paper as speakers in the conference.

Taking into account what was exposed, we describe a set of eleven norms that govern the simplified version of the local conference management system. For some of the norms we have specified the sanctions (punishments or rewards) the agent should receive if it violates or fulfills the norm. Note that those sanctions are also norms that are activated when the related norm is violated or fulfilled. All norms are defined in the context of the organization *LocalConference* that inhabits the environment *ConferenceSociety*.

- **N1:** Organizers are prohibited to submit papers.
- **N2:** Reviewers are allowed to submit papers.
- **N3:** The conference chair has the permission to extend the *submission deadline* at the *submission deadline* if the number of papers received < 50.
- **N4:** Reviewers are prohibited to review their own papers.
- **N5 (Punishment for the violation of N4):** Reviewers have their role canceled.
- **N6 (Punishment for the violation of N4):** The conference chair is obliged to discard the paper.
- **N7:** Reviewers are obliged to review the papers before the *notification deadline*.
- **N8:** The conference chair is obliged to send an *author notification* to the authors at the *notification deadline*.
- **N9:** Authors are obliged to register at the conference before the *registration deadline* if the paper was accepted.

- **N10 (Punishment for the violation of N9):** The conference chair is obliged to exclude the paper from the list of publication.
- **N11 (Reward for the fulfillment of N9):** Authors are permitted to commit as speakers.

In Section 6.1, we show how to use the modeling language NormML to model those norms and verify if there are any conflicts between them. Also in sections 6.2, 6.3 and 6.4 this application is used to compare NormML with the most relevant MAS modeling language, methodology and organizational model.

CHAPTER 6: EVALUATION

In the previous chapters, we presented the normative modeling language NormML that allows the modeling and validating of the norms of a MAS and the checking for conflicts between them (Chapter 3). We have also discussed how the modeling of norms and the checking for conflicts are represent in the MAS modeling languages, methodologies and organization models (Chapter 4).

In this chapter we use the norms described in Chapter 5 that are used to govern a simplified version of a *conference management system* of a *local conference* with the aim to validate the approach presented in this dissertation.

In Section 6.1 we show how to use the modeling language NormML to model those norms and verify if there are any conflicts between them. In the following section, the example is used to compare NormML with the AORML modeling language (Section 6.2), the Gaia methodology (Section 6.3) and the OperA organizational model (Section 6.4). The three of them were chosen because they are the modeling language, methodology and organizational model with more items checked in Table 4.1 of Chapter 4. Thus, the final remarks are addressed in Section 6.5.

6.1 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH NORMML

The *local conference management system* defines a set of eleven norms inserted in the context of the organization *LocalConference* to restrict the behavior of entities playing different kinds of roles. From this set of norms: (i) six norms are obligations, two are prohibitions and three are permissions; (ii) one restricts a communicative action; (iii) five are activated by the achievement of systems states and two are activated by dates; and (iv) four are sanctions. Table 6.1 presents the main elements of the norms of the *local conference management system* with more details.

Norm	Deontic concept	Involved entities	Action	Activation constraints	Sanctions	Context
N1	Prohibition	All agents playing the role <i>Organizer</i>	Non-communicative action "submit paper"	-	-	The organization <i>Local Conference</i>
N2	Permission	All agents playing the role <i>Reviewer</i>	Non-communicative action "submit paper"	-	-	The organization <i>Local Conference</i>
N3	Permission	All agents playing the role <i>Conference Chair</i>	Non-communicative action "extend the submission deadline"	If number of papers received < 50	-	The organization <i>Local Conference</i>
N4	Prohibition	All agents playing the role <i>Reviewer</i>	Non-communicative action "review paper"	If author of paper = name of Reviewer	Punishment N5 and punishment N6	The organization <i>Local Conference</i>
N5	Obligation	All agents playing the role <i>Reviewer</i>	Non-communicative action "cancel role <i>Reviewer</i> "	Violation of N4	-	The organization <i>Local Conference</i>
N6	Obligation	All agents playing the role <i>Conference Chair</i>	Non-communicative action "discard paper"	Violation of N4 and If author of paper = name of Reviewer that violates N4	-	The organization <i>Local Conference</i>
N7	Obligation	All agents playing the role <i>Reviewer</i>	Non-communicative action "review papers"	Before the notification deadline (e.g. 31/03/2011)	-	The organization <i>Local Conference</i>
N8	Obligation	All agents playing the role <i>Conference Chair</i>	Communicative action of sending an "author notification"	If notification deadline (e.g. 31/03/2011)	-	The organization <i>Local Conference</i>
N9	Obligation	All agents playing the role <i>Author</i>	Non-communicative action "register at the conference"	Before the registration deadline (e.g. 31/04/2011) and if author have paper accepted	Punishment N10 and reward N11	The organization <i>Local Conference</i>
N10	Obligation	All agents playing the role <i>Conference Chair</i>	Non-communicative action "exclude the paper"	Violation of N9 and if author of paper = name of author that violates N9	-	The organization <i>Local Conference</i>
N11	Permission	All agents playing the role <i>Author</i>	Non-communicative action "commit role <i>Speaker</i> "	Fulfillment of N9	-	The organization <i>Local Conference</i>

Table 6.1 Main elements of the norms of the local conference management system

Considering the information of the main elements that composes the norms of Table 6.1, the NormML graphical model of the *local conference management system* was created. Due to the available space, the graphical model of the *local conference management system* will be presented in pieces. Figure 6.1 shows the graphical model of the norms *N1* and *N2*.

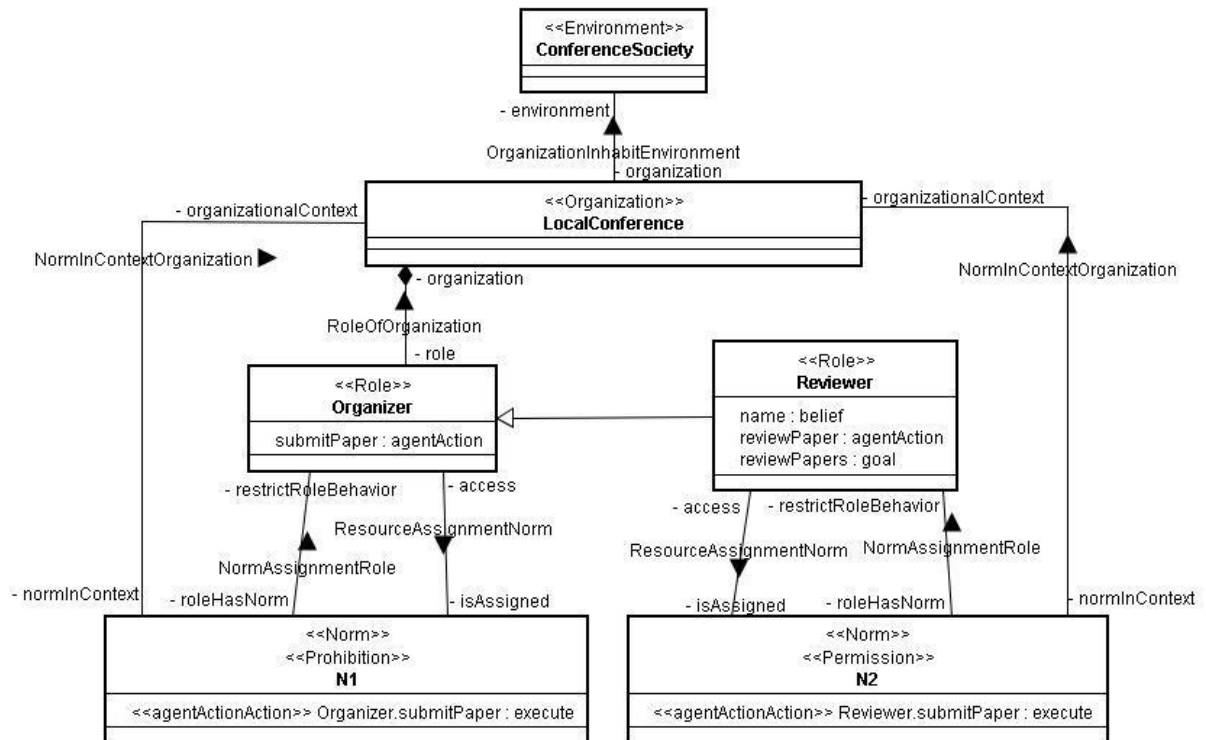


Figure 6.1 NormML graphical model of *N1* and *N2*

The *LocalConference* organization is represented as an UML class with the stereotype *<<Organization>>* and the *ConferenceSociety* environment is represented as an UML class with the stereotype *<<Environment>>*. The *LocalConference* organization belongs to the *ConferenceSociety* environment as represented by the relationship *OrganizationInhabitEnvironment*. *LocalConference* is composed of the role *Organizer* (roles are also represented as an UML class with the *<<Role>>* stereotype) that is extended by the role *Reviewer* and *ConferenceChair* (represented by the generalization relationship). As a result, *Reviewer* and *ConferenceChair* implicitly inherit the agent action *submitPaper* defined in *Organizer* (represented as an attribute of the type *agentAction*).

The *Reviewer* role has an agent action called *reviewPaper*, a belief called *name* and a goal called *reviewPapers*. Beliefs and goals are represented as attributes of the type *belief* or *goal* respectively (see the *Reviewer* class in Figure 6.1).

Norms are represented as UML classes with the stereotype <<Norm>>, a second stereotype describing its *deontic concept* (e.g. <<Prohibition>>, <<Permission>> or <<Obligation>>), and third stereotype describing its *sanction* type only used if the norm is a sanction (e.g. <<Punishment>> or <<Reward>>). All norms are in the *context* of the *LocalConference* organization (represented by the relationship *NormInContextOrganization*).

N1 restricts the behavior of the role *Organizer* (represented by the relationship *NormAssignmentRole*) stating a prohibition to the execution of the *action submitPaper* (represented as an attribute with the stereotype <<agentActionAction>> and the type *execute*). *N2* restricts the behavior of the role *Reviewer* stating a permission to the execution of the same *action*.

The *Conference* is modeled in Figure 6.2 as an *Entity* and has the attribute *numberOfPapers* of the type *int*. The papers of the conference are modeled as an *Entity* called *Paper* and it has the attribute *author* of the type *String*. Entities are represented as UML classes with the stereotype <<Entity>> and their attributes are represented by attributes in the attribute compartment of the classes.

The *ConferenceChair* role has an agent action called *extendSubmissionDeadline* and a protocol called *authorNotificationProtocol* which contains the message *authorNotification*. Protocols are represented as an attribute of the type *protocol* that has a constraint indicating its messages (see the *ConferenceChair* class in Figure 6.2). *N3* in Figure 6.2 restricts the behavior of the role *ConferenceChair* by stating a permission to the execution of the *action extendSubmissionDeadline* if the number of papers of the conference is < 50 (represented as an attribute with the stereotype <<conditionalAttribute>>, the type *lessThan*, and the initial value 50) and if it is 28/02/2011 that is the date of the submission deadline (represented as an attribute with the stereotype <<if>>).

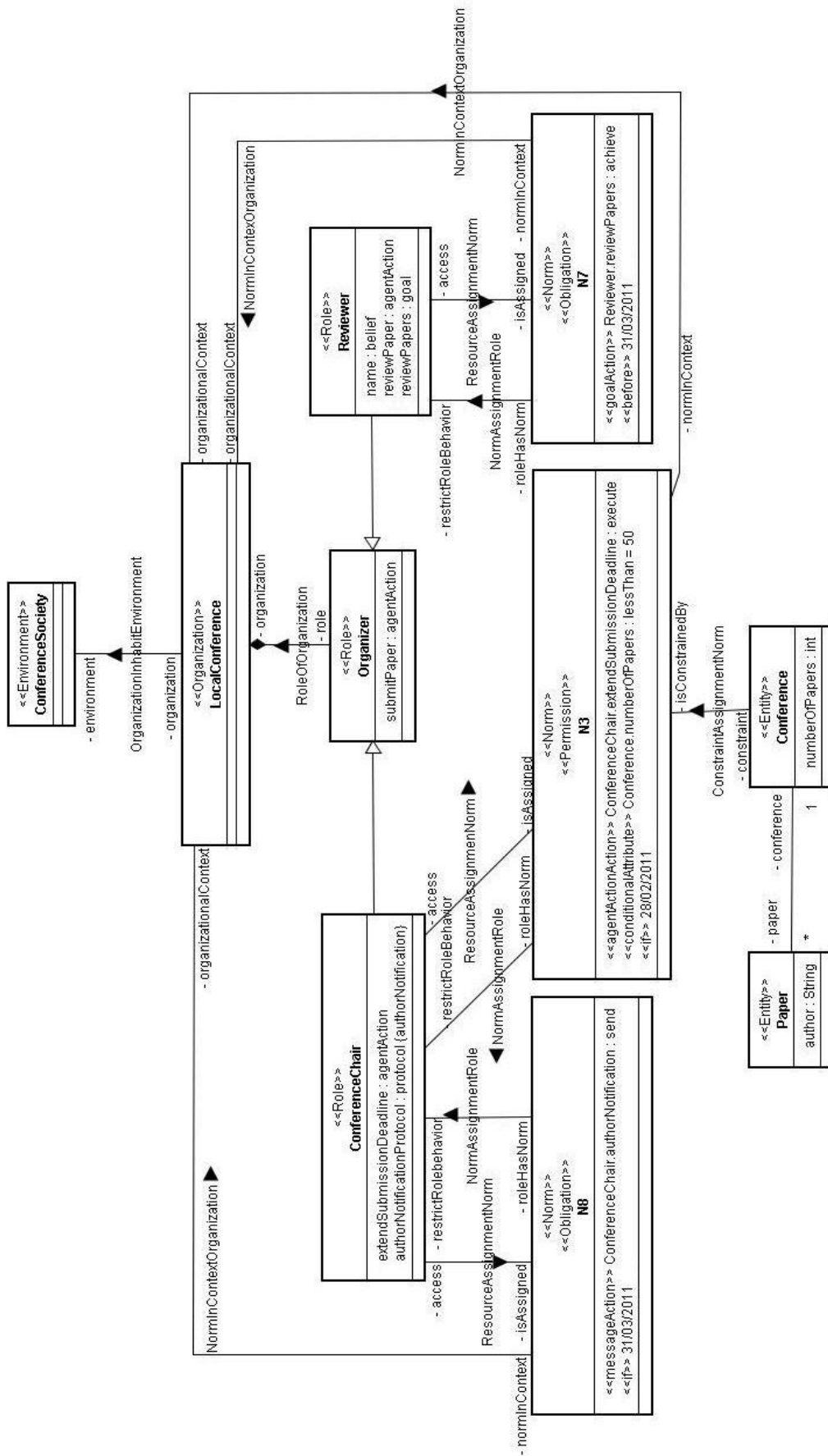


Figure 6.2 NormML graphical model of N3, N7 and N8

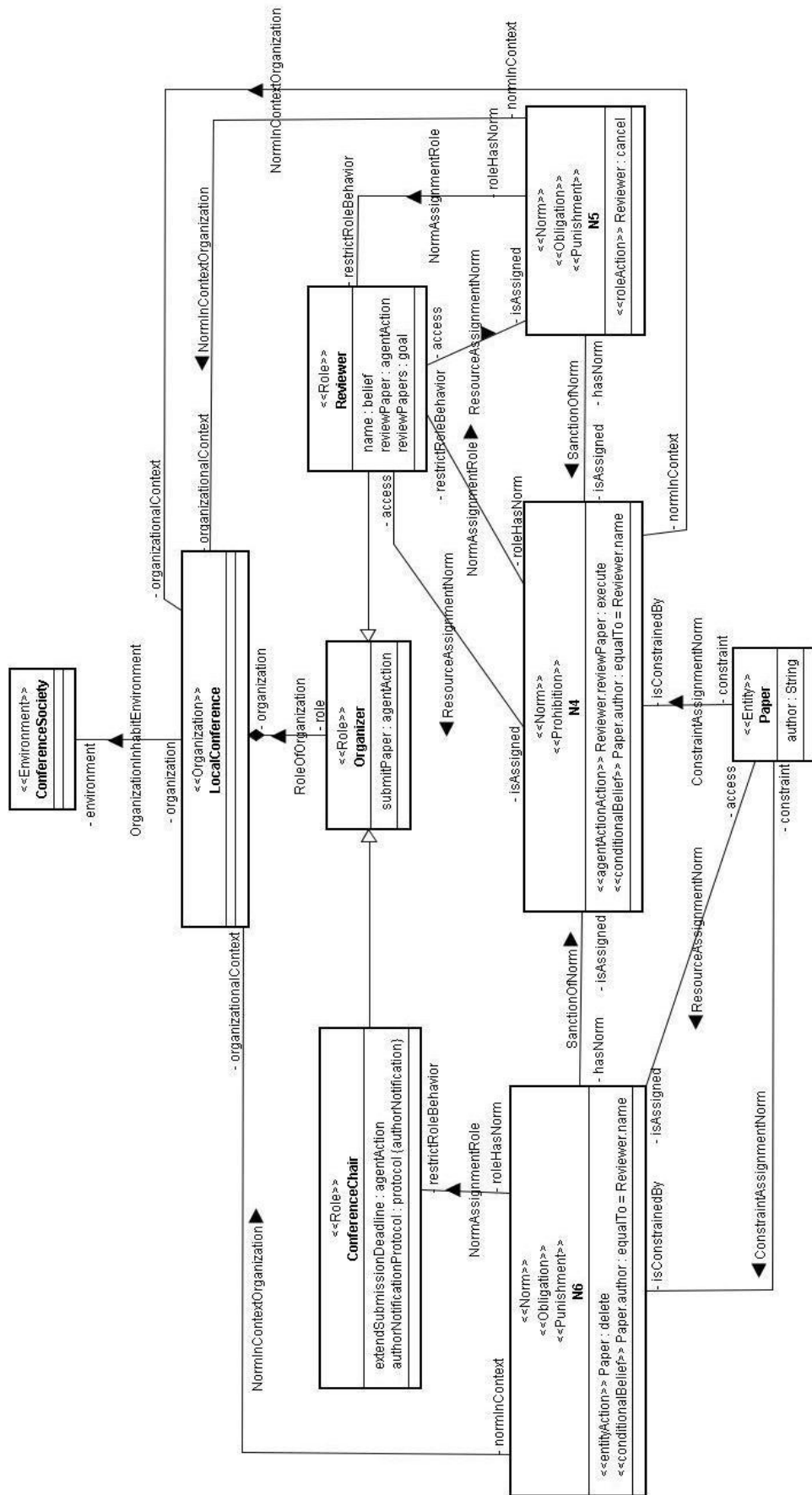


Figure 6.3 NormML graphical model of N4, N5 and N6

Figure 6.3 shows the graphical model of the norms *N4*, *N5* and *N6*. *N4* restricts the behavior of the role *Reviewer* stating a prohibition to the execution of the action *reviewPaper* if the author of the paper is equal to its name (represented as an attribute with the stereotype `<<conditionalBelief>>`, the type *equalTo* and the initial value referring to the *name* belief of the *Reviewer* role).

N5 restricts the behavior of the role *Reviewer* by stating an obligation to cancel the role *Reviewer* (represented as an attribute with the stereotype `<<roleAction>>` and the type *cancel*) as a punishment for the violation of the norm *N4*. *N6* restricts the behavior of the role *ConferenceChair* by stating an obligation to delete the paper (represented as an attribute with the stereotype `<<entityAction>>` and the type *delete*) if the author of the paper is equal to the name of the reviewer (represented as an attribute with the stereotype `<<conditionalBelief>>`, the type *equalTo* and the initial value referring to the *name* belief of the *Reviewer* role). *N6* is defined as a punishment for the violation of the norm *N4*. Both *N5* and *N6* are sanctions of *N4* and for this reason their classes have a *SanctionOfNorm* relationship with *N4*.

N7 in Figure 6.2 restricts the behavior of the role *Reviewer* by stating an obligation to the achievement of the goal *reviewPapers* (represented as an attribute with the stereotype `<<goalAction>>` and the type *achieve*) before the date *31/03/2011* that is the date of the notification deadline (represented as an attribute with the stereotype `<<before>>`). And, *N8* also in Figure 6.2 restricts the behavior of the role *ConferenceChair* by stating an obligation to send the message *authorNotification* (represented as an attribute with the stereotype `<<messageAction>>` and the type *send*) if it is *31/03/2011* (represented as an attribute with the stereotype `<<if>>`).

Figure 6.4 shows the graphical model of the norms *N9*, *N10* and *N11*. *LocalConference* is also composed of the roles *Author* and *Speaker*. The *Author* role has an agent action called *registerAtConference*, a belief called *name* and a goal called *havePaperAccepted*. The norm *N9* restricts the behavior of the role *Author* by stating an obligation to the execution of the action *registerAtConference* before the date *31/04/2011* (date of the registration deadline) if the *Author* achieved its goal *havePaperAccepted* (represented as an attribute with the stereotype `<<conditionalGoal>`, the type *equalTo* and the initial value *true*).

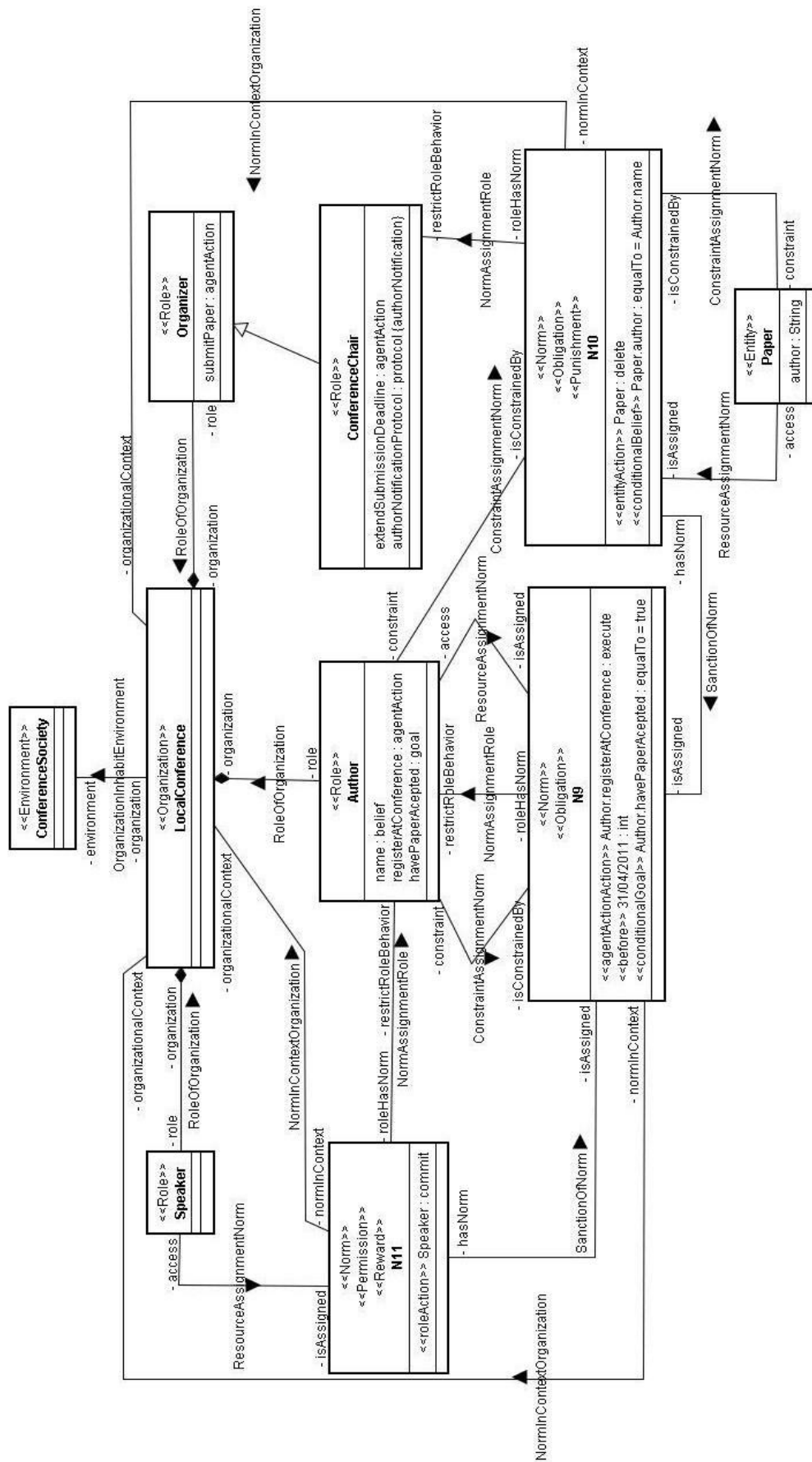


Figure 6.4 NormML graphical model of N9, N10 and N11

In case of violation of *N9*, *N10* states a punishment by restricting the behavior of the role *ConferenceChair* as an obligation to delete the paper if the author of the paper is the name of the author that violated *N9*.

In case of fulfillment of *N9*, *N11* states a reward by describing a permission to the role *Author* to commit with the role *Speaker* (represented as an attribute with the stereotype <<*roleAction*>> and the type *commit*).

In Appendix G all the abstract models of the norms of the *local conference management system* are illustrated.

NormML also offers a check for conflicts mechanism (previously detailed in Section 3.4) that is capable of verifying all the norms two by two with the aim of finding out conflicts between them. By applying such mechanism in the norms of the *local conference management system*, it detected that norms *N1* and *N2* are in conflicts because they are both in the same context (the LocalConference organization), restricting the behavior of related entities (the role *Organizer* is super-role of the role *Reviewer*), with opposite deontic concept operators (*prohibition* and *permission*), to execute the same action (*submitPaper*) in periods of time that intersects since both have no activation constraints, i.e. they are always active.

6.2 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH AORML

AORML is a modeling language to the modeling of organizations and organizational systems. In AORML there are no distinctions between agents and roles, and organizations are modeled as institutional agents.

In AORML the agents have three kinds of obligations. Two are represented by the *commitment/claim* relationship and describe a compromise between two agents. The last one is represented by the *hasDutyTo* relationship between agents and actions or messages, which means that the agent must execute such action or send/receive such message. In the same way permissions and prohibitions can be modeled respectively by the *hasRightTo* and *hasNoRightTo* relationships.

The *commitment/claim* relationship can be seen as an obligation in the context of an interaction, and the *hasDutyTo*, *hasRightTo* and *hasNoRightTo* relationships as

norms of an organizational context. In the *commitment/claim* relationship a deadline can be described as activation constraint. But in the *hasDutyTo*, *hasRightTo* and *hasNoRightTo* relationships there is no way to describe any kind of activation constraint.

Taking into account such characteristics, the norms *N3*, *N4*, *N5*, *N6*, *N9*, *N10* and *N11* of the *local conference management system* could not be fully modeled with AORML because of their activation constraints. And the norms *N7* and *N8* could only be modeled if they are represented as commitments in an interaction context which was not the context defined by the application.

In AORML, the violation of a norm is only considered when an agent does not fulfill a commitment but the language does not provide ways to describe sanctions. Thus, none of the sanctions of the example can be modeled with AORML.

The AORML language assumes that there is a normative inconsistency when there is at the same time a permission and a prohibition (which is the case of norms *N1* and *N2*), or a prohibition and an obligation, to the same action. Although the language considers that such kind of conflicts can occur, it does not have a mechanism to detect these conflicts as NormML does.

6.3 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH GAIA

Gaia is a methodology to the analyses and design of MAS that is basically represented as an organization with a set of roles interacting. In Gaia it is possible to describe *organizational rules* to the roles of an organization. Those organizational rules can be viewed as obligations, when they describe things that a role must do (*liveness rules*) or guarantees (*safety rules*), or they can be viewed as prohibitions, when they describe things a role must not do (*liveness rules*).

Expressions based on temporal logic (Manna and Pnueli, 1992 and Manna and Pnueli, 1995) can be described as *organizational rules*, including communicative and non-communicative actions and the achievement of states. Each *organizational rule* can have one activation and one deactivation condition.

In Gaia, only static permissions can be described to the reading and modifying of objects of the system. The static permissions, and the obligations and prohibitions of the organizational rules cannot be violated by the roles, thus the language do not address the question of punishments or any kind of sanction.

After analyzing the example of the *local conference management system*, we have found out that norms *N5*, *N6*, *N10* and *N11* could not be modeled in Gaia because they are activated by the violation/fulfillment of other norms. Norms *N2*, *N3* and *N11* could not be described too because they state permissions over the execution of actions, and by using Gaia it is not possible to describe permissions.

Assuming hypothetically that norm *N2* could be modeled by the methodology, the conflict between the norms *N1* and *N2* could not be found by this methodology because it does not have a mechanism to detect norms conflicts.

6.4 MODELING THE LOCAL CONFERENCE MANAGEMENT SYSTEM WITH OPERA

OperA is an organizational model that prescribes a formal structure for organizational processes, including normative issues. With OperA obligations, permissions and prohibitions can be described to agents, roles, agents playing roles, groups of agents and all agents of the system in the context of an organization, an interaction scene or a transition of scene.

The norms restrict the execution of communicative and non-communicative actions and can be activated by one or more events as an *activation condition*, a *maintenance condition*, an *expiration condition* or a *deadline*. The events can refer to actions, time expressions, changes of system states and the fulfillment/violation of another norm. Punishments can also be described to a norm that was violated.

Given the foregoing, only the norm *N11* of the *local conference management system* could not be modeled with OperA because it is a reward of norm *N9* and by using the organizational model it is only possible to model punishments.

The OperA organizational model allows the automatic verification of conflicts between the norms that apply to a given entity. However, such mechanism does not give support to the checking of conflicts between norms applied to different entity

types, i.e., between norms applied to related entities. For instance, the mechanism does not give support to the checking of conflicts between norms applied to roles in the same hierarchy as it is the case of norms *N1* and *N2*.

6.5 FINAL REMARKS

In this chapter we showed that NormML is able to model different kinds of norms, as the norms of the *local conference management system*, encompassing all the main elements that compose the norms, and after all, checking for conflicts between them.

The same is not possible when we try to execute such task with the MAS modeling languages, notations of methodologies and organizational models available, because, as presented before in Chapter 4, they do not support all the main elements of the norms or they do not offer a mechanism to detect norms conflicts. Even considering the works that model more elements, following Table 4.1 of Chapter 4, we demonstrated that they fail. The AORML modeling language and the Gaia methodology failed in modeling a set of elements. Although the OperA organizational model is more complete while describing norms and checking for conflicts, it does not provide a graphical representation to the norms.

Differently from the related work analyzed, NormML was designed specifically with the aim to model the norms of MAS and check for conflicts between them.

CHAPTER 7: THE NORMML TOOL KIT

In this chapter we present the NormML Tool Kit that allows the creation of NormML models, the automatic validation of them and the checking for norms conflicts.

The NormML Tool Kit is composed of two plugins: *NormML Editor* and *NormML Conflict Checker*. They both were developed as plugins to the Eclipse framework (The Eclipse Foundation, 2011). The Eclipse IDE (Integrated Development Environment) has quickly grown in the developing software community over the last few years becoming a popular IDE for software development, especially for Java based applications. The Eclipse IDE is open source and its framework is easily extensible, therefore the building of plugins for this IDE became a common practice.

Each plugin used in the NormML Tool Kit has a set of features to cover all the steps of the design of the MAS norms. Figure 7.1 illustrates the process used by the NormML Tool Kit by showing the activity and identifying the plugins used in each one.

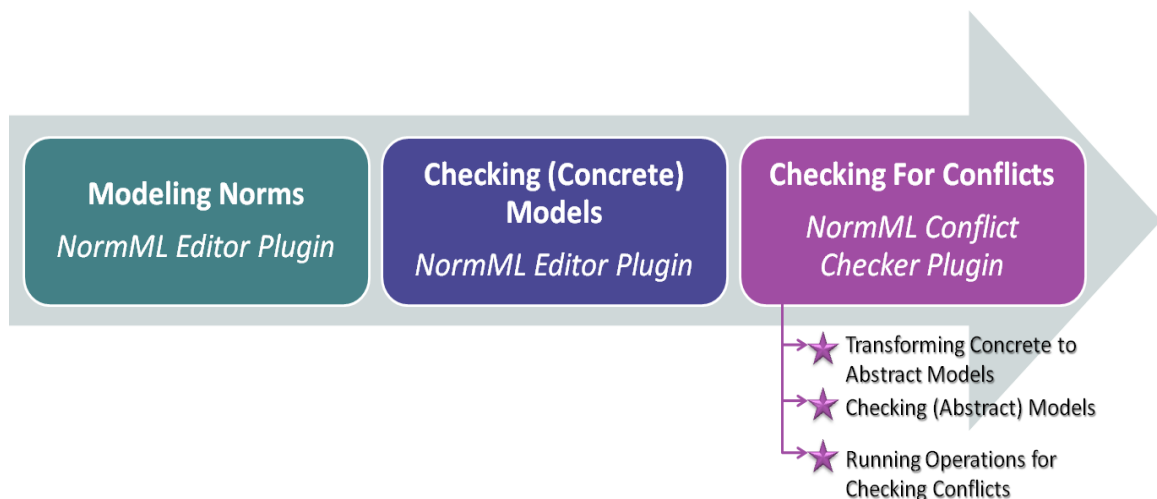


Figure 7.1 The NormML Tool Kit process

The NormML Editor plugin is used to model the norms following the *concrete syntax* proposed by the NormML modeling language described in Section 3.5. Such editor also gives support to the checking of such concrete models in order to guarantee its syntactical correctness. It checks if the models are well-formed according to the metamodel that represents the concrete syntax (see Section 7.2).

After checking the concrete models, the norm designer can use the NormML Conflict Checker plugin to check the conflicts among the norms modeled. In order to do so, the plugin transforms the concrete models into abstract models and validates the abstract models. Such validation is done by checking the well-formedness of such models according to the NormML metamodel. If all the norms could be validated, the operations for checking the conflicts are executed.

The validation of the abstract models could be skipped if the user would always do the checking of the concrete models since the transforming of concrete to abstract models is correct. Another reason to maintain the validation of the abstract models apart of the checking of the concrete models is to allow the user to check for conflicts between the norms independently of the use of the NormML Editor to construct the models. Thus, the user could use any editor to construct the norms models, for instance a simple text editor, and still use the NormML Conflict Checker plugin.

The next sections detail each feature of the NormML Tool Kit discussing the technology used and the support given by its activity. The norms of Chapter 5 are used to illustrate the applicability of the tool.

7.1 MODELING NORMS

The modeling of norms can be done in the NormML Editor plugin developed by using the Eclipse Modeling Framework Project (EMF) (Eclipse Modeling Framework Project, 2011). EMF is a modeling framework and a code generation provider for the building of modeling tools and other applications based on a structured data model. EMF consists of three elementary parts: EMF(core), EMF.Edit and EMF.Codegen. EMF(core) has a metamodel (Ecore) for describing models that was used in the development of the NormML Editor to define the NormML concrete metamodel.

EMF.Edit provides generic reusable classes for building editors from EMF models and EMF.Codegen provides the generation of everything needed to build a complete editor from an EMF model by using the editor interfaces. Both of them were employed to generate the NormML Editor from the NormML concrete metamodel

described in Ecore. The NormML Editor has a wizard to assist the creation of norms diagram that are saved as a *.normml file (Figure 7.2).

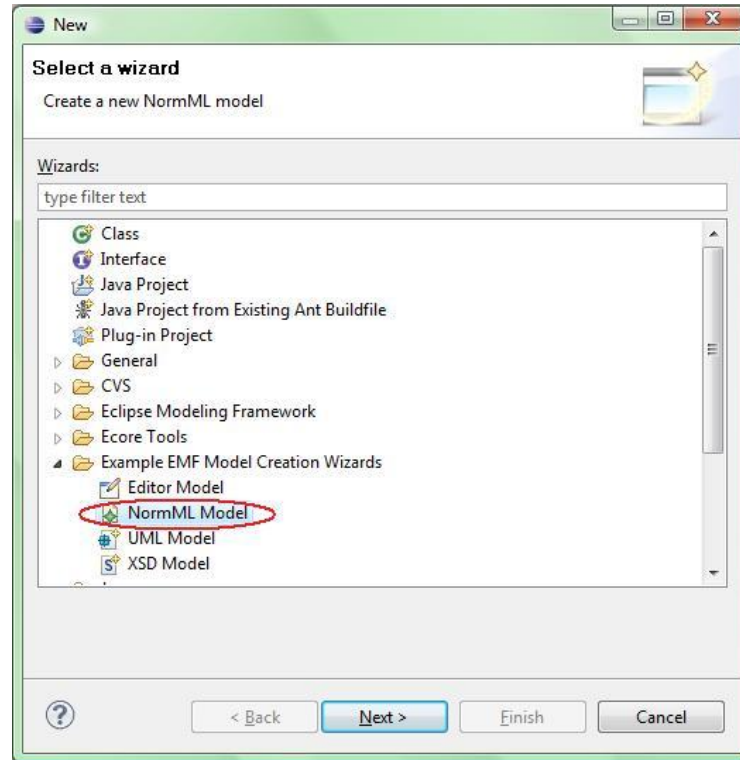


Figure 7.2 The NormML Editor wizard

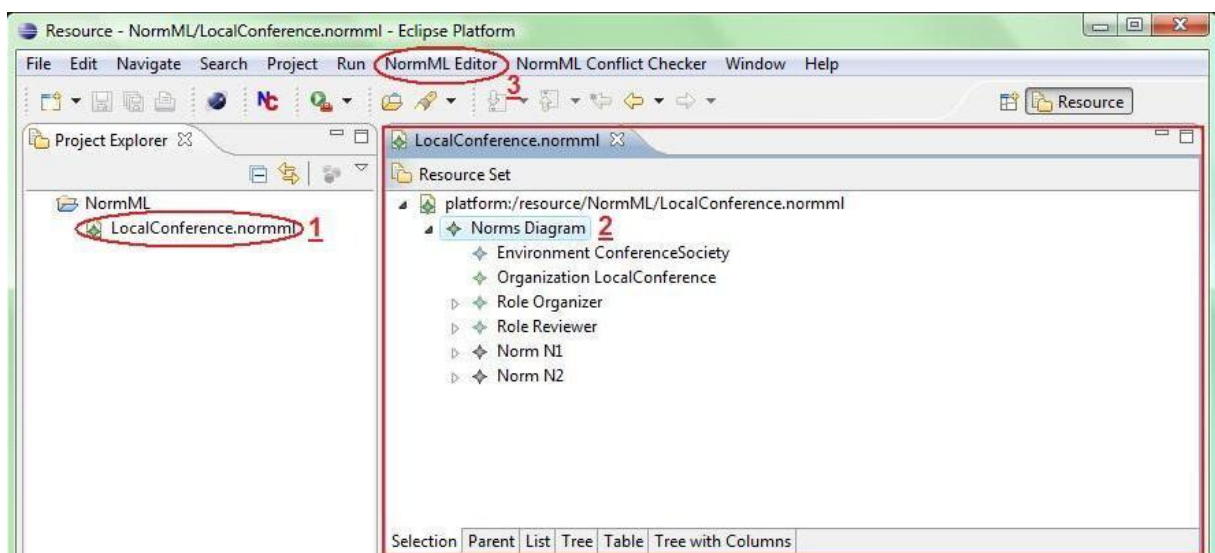


Figure 7.3 Eclipse perspective of the NormML Editor plugin

Figure 7.3 shows the Eclipse perspective when the *.normml file (selected in item 1 of Figure 7.3) is opened with the NormML Editor window (highlighted by item 2 of Figure 7.3) and its menu (stressed by item 3 of Figure 7.3).

The NormML model is represented as a tree of nodes. The entities of the NormML model (agents, roles, organizations, environments, entities and norms) are created as nested nodes of the Norms Diagram element and their properties and relationships can be edited in the Eclipse Properties View. Figure 7.4 illustrates the creation of norm *N1* of Chapter 5 in the Norms Diagram (highlighted by item 1 of Figure 7.4) and the assignment of the norm to the role *Organizer* in the Properties View (item 2 of Figure 7.4).

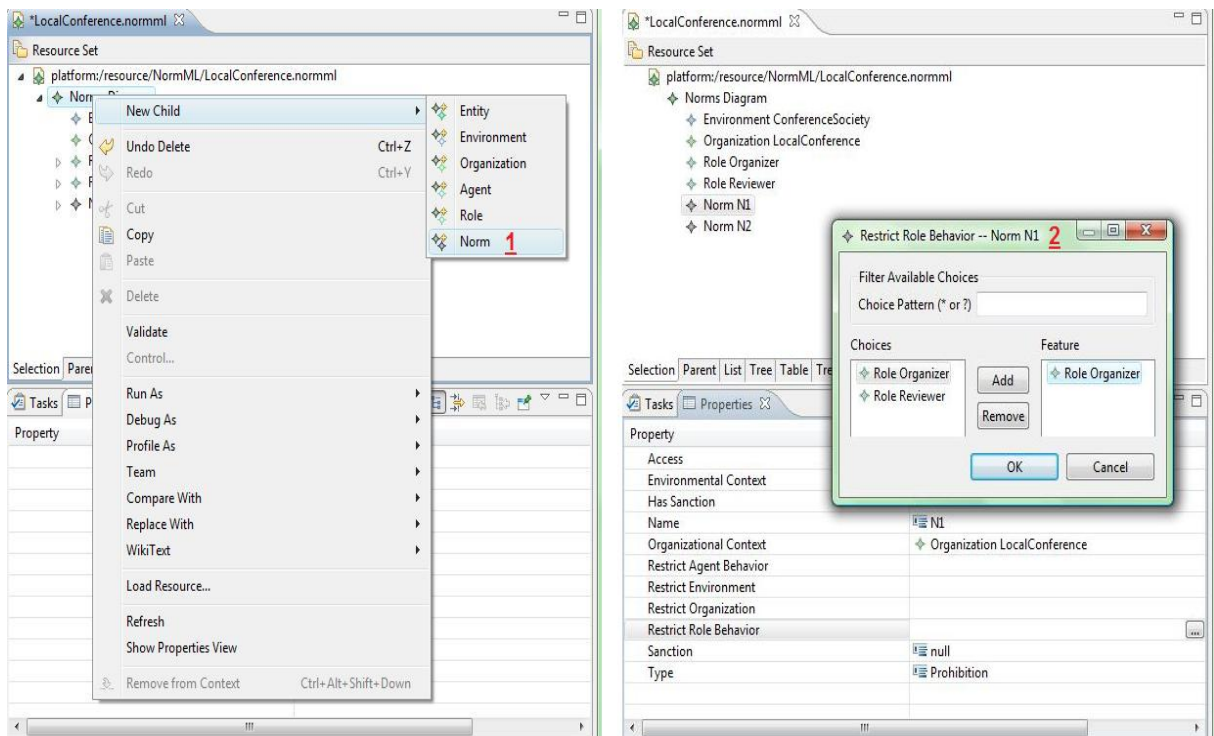


Figure 7.4 Creation and edition of *N1* in the NormML Editor plugin

7.2 CHECKING (CONCRETE) MODELS

The checking (or validation) of the concrete models is also done by using the NormML Editor plugin. Such validation uses the automatic support provided by EMF to validate models following the constraints of the defined metamodel. Constraints such as OCL invariants can be defined by the creation of the EAnnotation element of the Ecore editor.

Figure 7.5 shows an OCL invariant example called “*NormContextCanNotBeNull*” written by using the EAnnotation element. Since such invariant applies to norm, it was included in the *Norm* class of the concrete syntax metamodel. The invariant guarantees that each *Norm* will be in one context (organizational or environmental context).

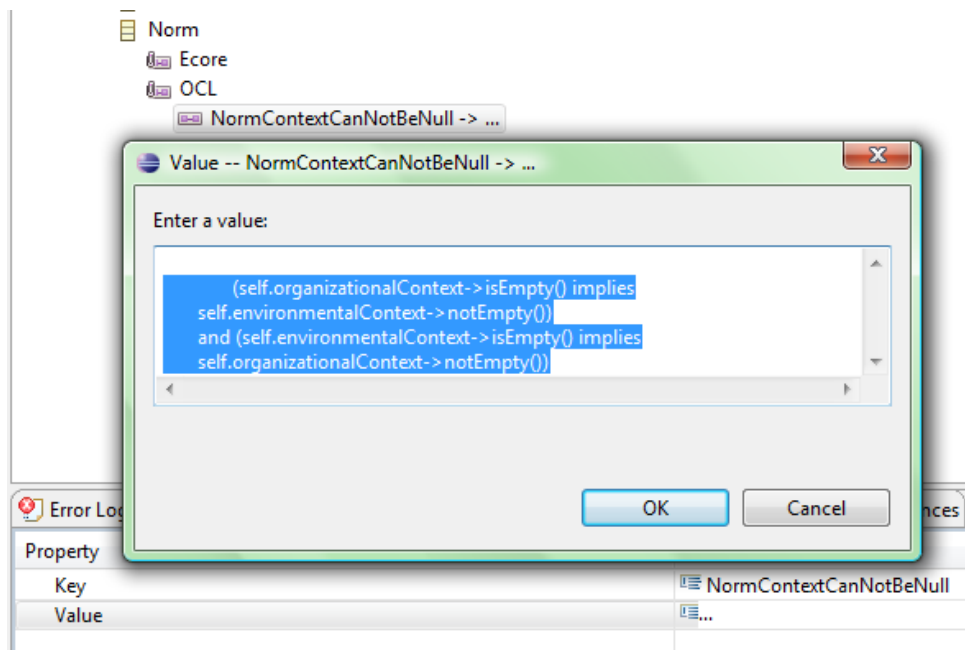


Figure 7.5 OCL invariant example in the Ecore editor

Supposes that *N1* from Chapter 5 were modeled without inform its context (the *LocalConference* organization) as indicated in item 1 of Figure 7.6. By selecting the “*Validate*” item at the NormML Editor menu (see item 2 of Figure 7.6), an alert message would appear informing that the “*NormContextCanNotBeNull*” invariant was violated (see Figure 7.7).

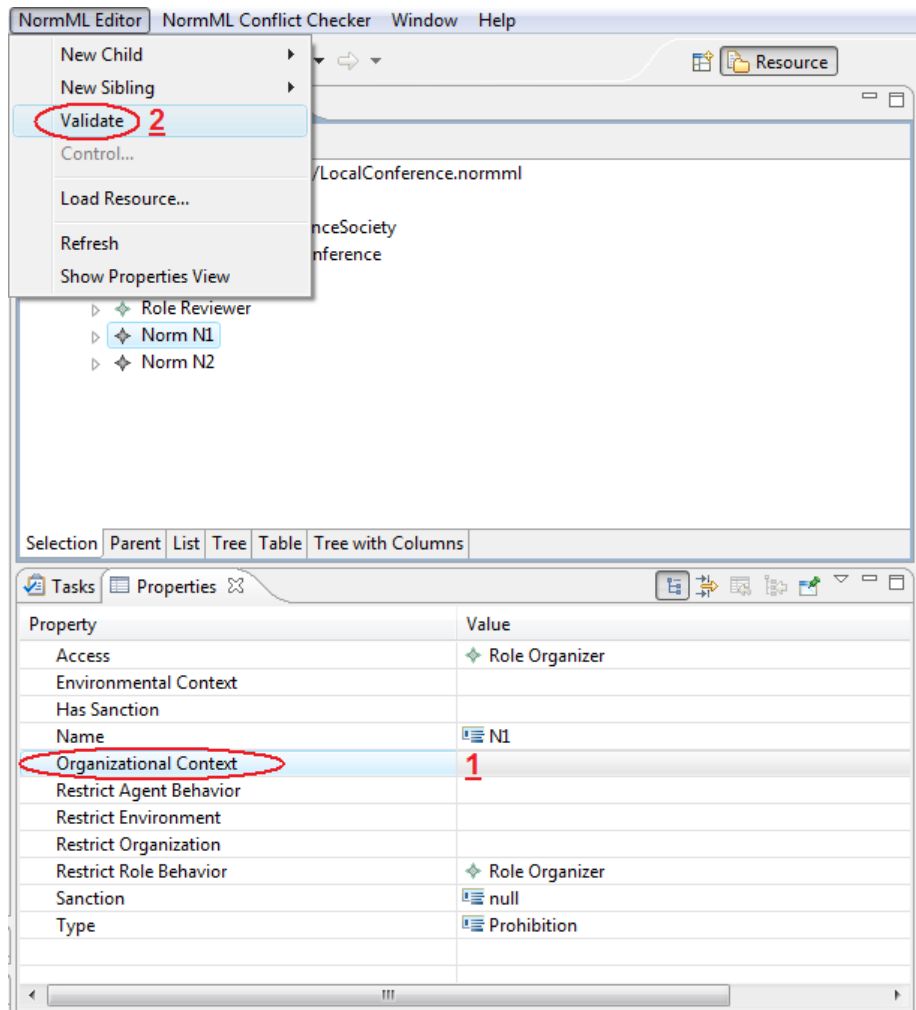


Figure 7.6 N1 violation of "NormContextCanNotBeNull" invariant

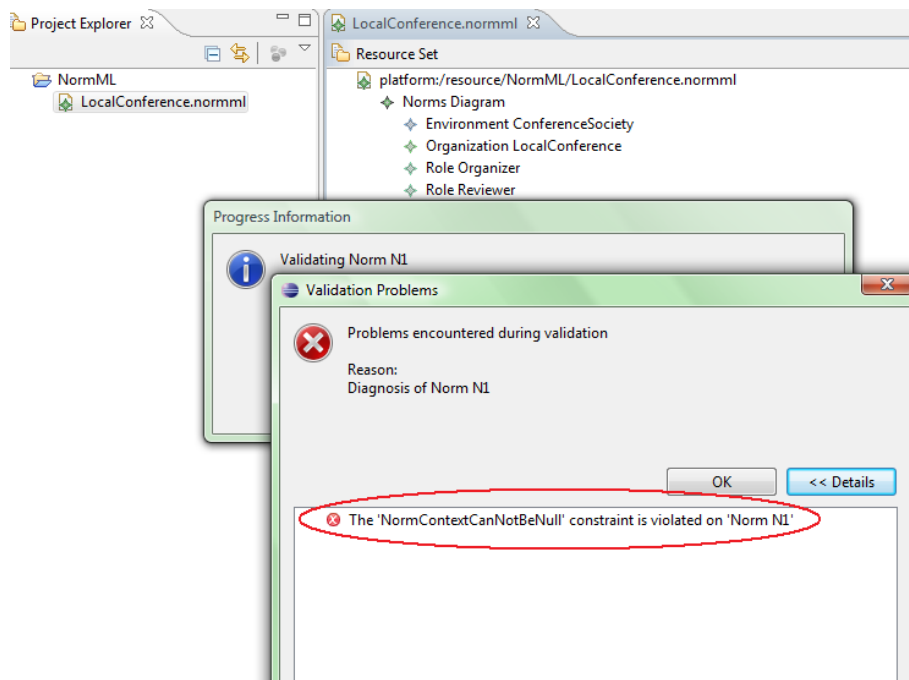


Figure 7.7 N1 violation of "NormContextCanNotBeNull" invariant result

7.3 CHECKING FOR CONFLICTS

Finished the modeling and validating of the NormML concrete models in the NormML Editor plugin, the norm designer can use the NormML Conflict Checker plugin to check the conflicts among the norms modeled. In order to do so, the plugin transforms the concrete models into abstract models and checks (or validates) the abstract models.

By selecting the “*Check for Conflicts*” item at the NormML Conflict Checker menu (highlighted in Figure 7.8) the transformation will be executed (see Section 7.3.1), and after it, well-formedness rules operations. If the model is well-formed, then the check for conflicts operations will be executed. Both well-formedness rules operations and check for conflicts operations were implemented with EOS (Eye OCL Software) (The EOS Component, 2011) as demonstrated in Section 7.3.2.

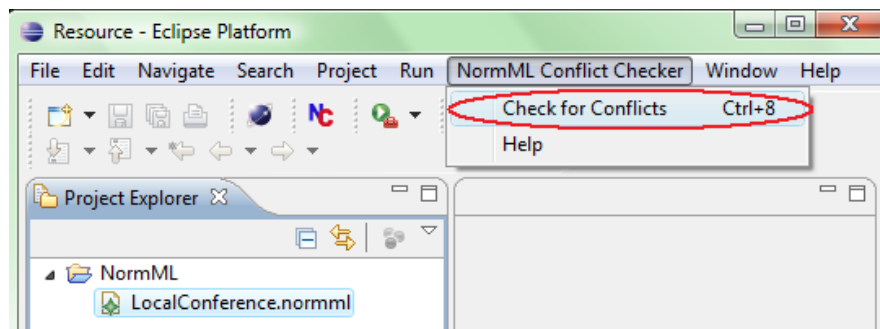


Figure 7.8 NormML Conflict Checker menu

If there are any conflicts between the norms of the model, an alert message will be shown as illustrated in Figure 7.9 to the *local conference management system* model verification.

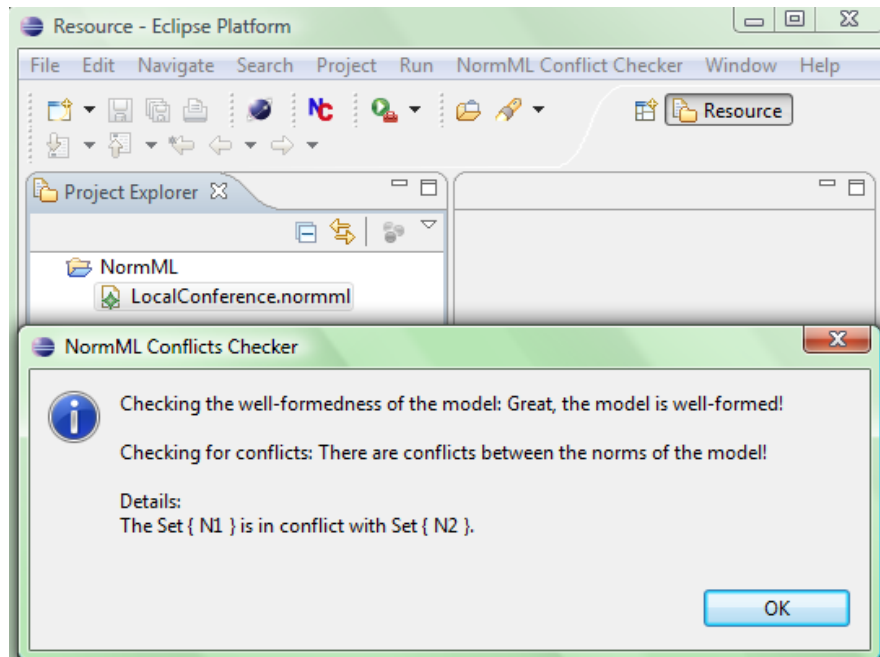


Figure 7.9 Check for conflicts result of the NormML Conflict Checker plugin

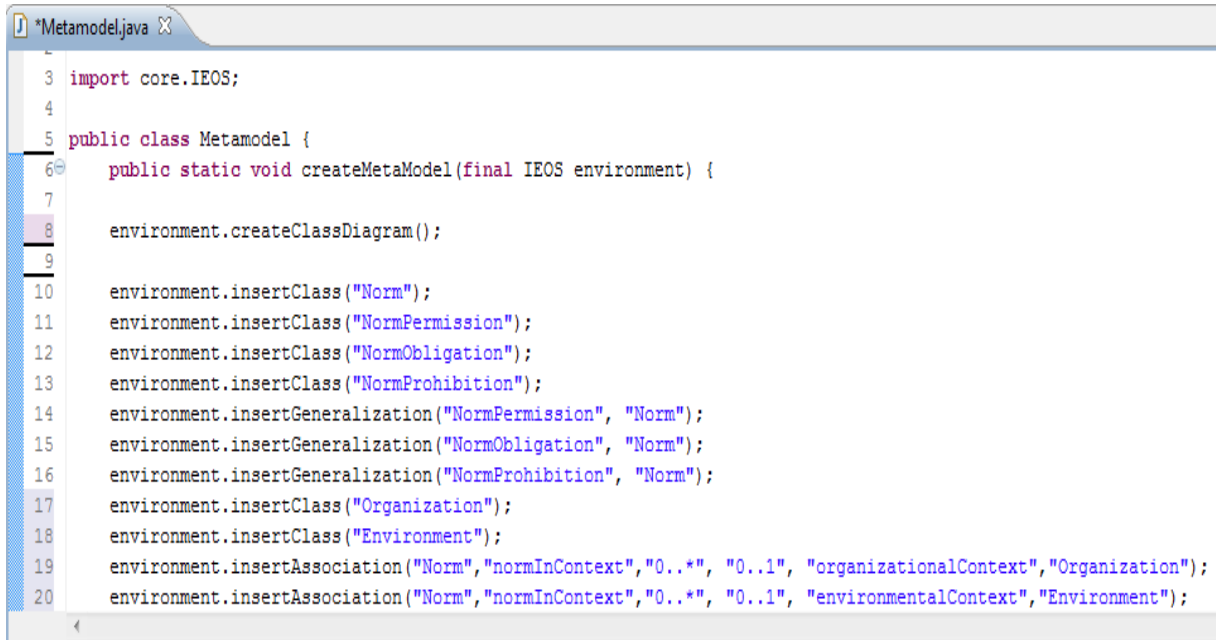
7.3.1 Transforming concrete to abstract models

While the concrete models are described in XML (Extensible Markup Language) as *.normml files, the abstract models are described by using EOS in *.java files. EOS is a Java API (Application Programming Interface) that allows the description of class and object diagrams, the implementation of operations in OCL and their execution over object diagrams. In our case, a class diagram is used to model the NormML metamodel and object diagrams are used to describe the abstract models of NormML, which are composed of elements that are instances of the metaclasses of the NormML metamodel.

Figure 7.10 shows a piece of the NormML metamodel as a class diagram implemented in EOS and Figure 7.11 illustrates the abstract model of norm *N1* as an object diagram also described in EOS.

The metaclasses of the NormML metamodel are included in the class diagram as classes by using the “insertClass” method and by informing the class name as parameter. Line 10 in Figure 7.10 illustrates the creation of the *Norm* metaclass. The *association* relationships between the metaclasses of the NormML metamodel are included in the class diagram by using the “insertAssociation” method and informing

the classes names, association ends names and multiplicities as parameters. Line 19 in Figure 7.10 depicts the creation of the *NormInContextOrganization* relationship.



```

3 import core.IEOS;
4
5 public class Metamodel {
6     public static void createMetaModel(final IEOS environment) {
7
8         environment.createClassDiagram();
9
10        environment.insertClass("Norm");
11        environment.insertClass("NormPermission");
12        environment.insertClass("NormObligation");
13        environment.insertClass("NormProhibition");
14        environment.insertGeneralization("NormPermission", "Norm");
15        environment.insertGeneralization("NormObligation", "Norm");
16        environment.insertGeneralization("NormProhibition", "Norm");
17        environment.insertClass("Organization");
18        environment.insertClass("Environment");
19        environment.insertAssociation("Norm", "normInContext", "0..*", "0..1", "organizationalContext", "Organization");
20        environment.insertAssociation("Norm", "normInContext", "0..*", "0..1", "environmentalContext", "Environment");

```

Figure 7.10 NormML metamodel described as a class diagram with EOS

The *generalization* relationships between the metaclasses of the NormML metamodel are included in the class diagram by using the “insertGeneralization” method and informing the sub-class and super-class names. Line 16 in Figure 7.10 shows the generalization between the *NormProhibition* and *Norm* metaclasses.

The instances of the metaclasses of the NormML metamodel are included in the object diagram as objects by using the “insertObject” method and by informing the object name and the class name as parameter. The creation of the *LocalConference* instance in line 10 of Figure 7.11 is an example. The relationships between the instances of the model are included in the object diagram as links by using the “insertLink” method and by informing the classes names, object names and association ends names as parameters. The creation of the *OrganizationInhabitEnvironment* relationship between the *LocalConference* organization and the *ConferenceSociety* environment described in line 12 of Figure 7.11 is an example.



```

2
3 import core.*;
4
5 public class ObjectDiagram {
6
7     public static void createObjectDiagram(final IEOS environment){
8         environment.createObjectDiagram();
9
10        environment.insertObject("Organization","LocalConference");
11        environment.insertObject("Environment","ConferenceSociety");
12        environment.insertLink("Organization","LocalConference","organization","environment","ConferenceSociety","Environment");
13        environment.insertObject("Role","Organizer");
14        environment.insertLink("Organization","Organizer","organization","role","Buyer","Role");
15        environment.insertObject("AgentAction","submitPaper");
16        environment.insertLink("AgentAction","submitPaper","action","role","Organizer","Role");
17        environment.insertObject("NormProhibition","N1");
18        environment.insertObject("AtomicExecute","executeSubmitPaper");
19        environment.insertLink("Organization","LocalConference","organizationalContext","normInContext","N1","Norm");
20        environment.insertLink("Role","Organizer","restrictRoleBehavior","roleHasNorm","N1","Norm");
21        environment.insertLink("AtomicExecute","executeSubmitPaper","access","isAssigned","N1","Norm");
22        environment.insertLink("AtomicExecute","executeSubmitPaper","action","resource","submitPaper","AgentAction");
23

```

Figure 7.11 N1 abstract model described as an object diagram with EOS

A transformer developed in XSLT (Extensible Stylesheet Language Transformations) (W3Schools, 2011a) was implemented to automatically transform the NormML concrete models into NormML abstract models. XSLT is a XML-based language used for the transformation of XML documents into new documents of any kind, based on the content of the XML documents. In the transformation process, XSLT uses XPath (W3Schools, 2011b) to define parts of the source document that should match one or more predefined templates in the transformation file. When a template is applied, XSLT will transform the matching part of the source document into the result document according to the template.

In our context, the XSLT transformer receives as source document the *.normml file of the concrete model described in XML and creates as a result document a *.java file with the abstract model described following the EOS syntax. The XSLT transformer executes a set of XSL transformation rules presented in Appendix F.

For instance, Figure 7.12 illustrates the code in XSL of the implementation of Rule 1. The template searches the source document for an “environment” element and creates on the result document the EOS command line to create an instance of

the Environment class, replacing the name of the instance by the name of the environment in the source document (see line 21 of Figure 7.12).

Rule1. For each *Environment env* of M , insert in \overline{M} an object \overline{env} of the class *Environment*.

```

19 </xsl:template>
20 <xsl:template match="environment">
21 environment.insertObject("Environment", "<xsl:value-of select="@name"/>");
22 </xsl:template>

```

Figure 7.12 XSL template that implements Rule 1

7.3.2 Running operations to check the abstract models

With the abstract models described with EOS the user can check for conflicts between the norms of the models in the NormML Conflict Checker plugin. The check for conflicts OCL operations were implemented in EOS in order to evaluate the object diagrams.

Also, the well-formedness rules described in OCL were implemented with EOS to execute a last verification in the models before the checking for conflicts operations to assure that the abstract models are well-formed. Figure 7.13 shows an example of OCL operation described with EOS.

```

25 environment.insertOperation("Set(Norm)", "checkingForConflictsMain", "Boolean",
26 "if(self->exists(n1,n2:Norm| (n1<>n2) and(n1.checkingForConflicts(n2))) then(true)else(false)endif", new Object[0]);
27

```

Figure 7.13 Main OCL operation of the check for conflicts described with EOS

The method “insertOperation” is used to create an OCL operation with EOS and it receives as parameters the context of the operation, the name of the operation,

the type of the element to be returned, the operation itself and a list of parameters to be used in the operation.

Figure 7.13 illustrates the creation of the OCL operation “checkinfForConflictsMain” that has a set of Norm elements as context and a Boolean value as return type. This operation calls, for each two different norms of the set, another operation (“checkinfForConflicts”) that will check for conflicts between them.

In the NormML Conflict Checker we reuse some of the check for conflicts operations and well-formedness rules implemented with EOS by Alcântara and Marinho (2010). They developed a tool to create NormML models and check for conflicts using EOS according to the preliminary version of NormML presented in (Figueiredo and Silva, 2010b).

CHAPTER 8: CONCLUSION AND FUTURE WORK

In this dissertation we have presented a modeling language called NormML that is able to model the norms of a MAS and to check the conflicts between these norms at *design time*.

We have pointed out the main elements that compose a norm and discussed how several MAS modeling languages and the notations provide by methodologies and organizational models give support to the modeling of these elements and to the checking of conflicts between norms. We have also emphasized the contributions of the normative modeling language NormML when compared with other modeling languages and notations used by methodologies and organization models in chapters 5 and 6.

As showed in Chapter 3, by using NormML it is possible (i) to model norms associated with different contexts; (ii) to regulate the behavior of individual and groups of individuals (or organizations); (iii) to define norms that restrict the execution of actions (including dialogical actions) and the achievement of states; (iv) to define activation constraints based on the definition of periods between actions, periods of time and predicates (values associated with attributes, beliefs and goals); (v) to define sanctions associated with the norms; (vi) to validate the models according to the well-formedness rules of the language; and (vii) to check for conflicts among the norms of a model.

In order to support the language we have developed a set of plugins to the Eclipse IDE called NormML Tool Kit that allows the user to construct norms models and validate them by executing the well-formedness rules and the check for conflicts operations over such models.

During the development of this work, preliminary versions of the NormML modeling language were presented in Figueiredo *et al.* (2011) and Figueiredo and Silva (2011), Figueiredo and Silva (2010a), Figueiredo and Silva (2010b) and Silva *et al.* (2010).

The mains contributions of this dissertation are:

- The investigation of the main elements that compose the norms;

- The review of the MAS modeling languages, methodologies, organizational models and other approaches that propose solutions for indentifying norms' conflicts;
- The NormML modeling language itself, to model norms and its main elements;
- The set of well-formedness rules for the validation of the models of the language;
- The set of operations to check for conflicts between norms that consider the main elements that compose the norms;
- The elaboration of a concrete syntax to the language that allows the user to create the norms' models;
- The set of transformations rules used to transform the concrete into abstract models;
- The tool used to (i) model and validate norms using NormML; (ii) check for conflicts between norms models; and (iii) automatically transform concrete models in abstract models.

In this work we focus on the modeling of the *static aspects* of the norms, i.e. the elements related to its composition. However, it is our intension to define a sequence diagram for NormML to describe the sequence of the executed actions and states achieved. By using such diagram it will be possible to: (i) represent *dynamic aspects* as the creation, cancellation and delegation of a norm; (ii) define norms in an interaction context; (iii) check norms' conflicts that depend on the sequence of the executed actions and states achieved; and (iv) identify the norms that are active and the ones that were violated or fulfilled. It is also our aim to develop a tool for modeling norms using the graphical representation of NormML.

REFERENCES

ALDEWERELD, H., DIGNUM, F., GARCIA-CAMINO, A., NORIEGA, P., RODRIGUEZ-AGUILAR, J. and SIERRA, C. *Operationalisation of norms for usage in electronic institutions*. In: Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems, 2006. p. 223–225.

BASIN, D., CLAVEL, M., DOSER, J. and EGEEA, M. *Automated analysis of security-design models*. Information and Software Technology, Volume 51, Issue 5, May 2009. p. 815–831.

BASIN, D., DOSER, J. and LODDERSTEDT, T. *Model driven security: from UML models to access control infrastructures*. ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 15, Issue 1, January 2006. p. 39–91.

BRADSHAW, J. M. *Software Agents*. MIT Press Cambridge, MA, USA, 1997.

CAIRE, G., COULIER, W., GARIJO, F., GOMEZ, J., PAVON, J., LEAL, F., CHAINHO, P., KEARNEY, P., STARK, J., EVANS, R. and MASSONET, P. *Agent Oriented Analysis Using Message/UML*. In: Proceedings AOSE '01 Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II, Springer-Verlag London, UK, 2002. p. 119–135.

CHOLVY, L. *Checking regulation consistency by using SOL-resolution*. In: ICAIL '99 Proceedings of the 7th international conference on Artificial intelligence and law, ACM New York, NY, USA, 1999.

CLAVEL, M., SILVA, V., BRAGA, C. and EGEEA, M. *Model-driven security in practice: an industrial experience*. In: ECMDA-FA '08 Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications, Springer-Verlag Berlin, Heidelberg, 2008. p. 326–337.

COSENTINO, M. *From requirements to code with the PASSI methodology*. In: Agent-Oriented Methodologies, B. Henderson-Sellers and P. Giorgini (Editors), Idea Group Inc., Hershey, PA, USA, 2005. p. 79–106.

CRANEFIELD, S. *Modeling and monitoring social expectations in multi-agent systems*. In: Coordination, Organizations, Institutions, and Norms in Agent Systems II, Springer-Verlag Berlin, Heidelberg, 2007. p. 308–321.

DANC, J. *Formal specification of AML*. Department of Computer Science Faculty of Mathematics, Physics and Informatics Comenius University Formal Specification of AML Master's Thesis, Ján Danc, Advisor: Mgr. Bratislava, 2008.

DEUTCH, M. and GERARD, H. B. *A study of normative and informational social influence upon judgment*. *Journal of Abnormal and Social Psychology*, 51, 1955. p. 629-636.

DIGNUM, V. *A model for organizational interaction: based on agents, founded in logic*. PhD dissertation, Universiteit Utrecht, SIKS dissertation series 2004-1, 2004.

DIGNUM, V. *The Role of Organization in Agent Systems*. In: *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, Author(s)/Editor(s): Virginia Dignum (Utrecht University, The Netherlands), 2009. p. 1-16.

ECLIPSE MODELING FRAMEWORK PROJECT. *Eclipse Modeling Framework Project*. <http://www.eclipse.org/modeling/emf/>, Accessed: Jan. 31, 2011.

EOS. *The EOS Component*. <http://www.bm1software.com/eos/>, Accessed: Jan. 31, 2011.

ETZIONI, O. and WELD, D. S. *Intelligent agents on the Internet: Fact, fiction, and forecast*. *IEEE Expert: Intelligent Systems and Their Applications* archive, Volume 10, Issue 4, August 1995. p. 44-49.

FERBER, J., STRATULAT, T. and TRANIER, J. *Towards an integral approach of organizations: the MASQ approach in multi-agent systems*. In: *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, Author(s)/Editor(s): Virginia Dignum (Utrecht University, The Netherlands), 2009. p. 51-75.

FERRAIOLO, D.F. and KUHN, D.R. *Role-Based Access Control*. In: *15th National Computer Security Conference*, 1992. p. 554–563.

FIGUEIREDO, K. and SILVA, V. *Modeling and Validating Norms in Multi-agent Systems*. In: *Workshop of Theses and Dissertations in Software Engineering (WTES) at Brazilian Conference on Software: Theory and Practice (CBSOFT 2010)*, Salvador, 2010a. p. 49-54.

FIGUEIREDO, K. and SILVA, V. *NormML: A Modeling Language to Model Norms*. In: *AutoSoft - Autonomous Software Systems (Workshop) at Brazilian Conference on Software: Theory and Practice (CBSOFT 2010)*, Salvador, 2010b. p.11-20.

FIGUEIREDO, K. and SILVA, V. *NormML: A Modeling Language to Model Norms*. In: III International Conference on Agents and Artificial Intelligence (ICAART 2011), Rome, 2011.

FIGUEIREDO, K., SILVA, V. and BRAGA, C. *A Modeling Language to Model Norms*. M. De Vos, N. Fornara, J. Pitt and G. Vouros (Eds.) Coordination, Organizations, Institutions, and Norms in Agent Systems VI, (COIN@AAMAS 2010 post-proceedings), LNAI 6541, Springer-Verlag, 2011.

FØLLESDAL, D., HILPINEN, R. *Deontic Logic: An Introduction*. In: HILPINEN, R. (Ed.). *Deontic Logic: Introductory and Systematic Readings*. Dordrecht: D. Reidel Publishing Company, 1971. p. 1-38.

FORNARA, N. and COLOMBETTI, M. *Specifying and enforcing norms in artificial institutions*. In: Proceedings of the 4th European Workshop on Multi-Agent Systems Coordination, Organizations, Institutions, and Norms in Agent Systems III Lecture Notes in Computer Science, Volume 4870/2008, 2008. p. 316-329.

FRANKLIN, S. and GRAESSER, A. *Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents*. In: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, New York, Springer-Verlag, 1996.

GAERTNER, D., GARCIA-CAMINO, A. and VASCONCELOS, W. *Distributed Norm Management in regulated Multi-Agent Systems*. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, ACM New York, NY, USA, 2007.

GARCIA-CAMINO, A., NORIEGA, P. and RODRIGUEZ-AGUILAR, J. *Implementing norms in electronic institutions*. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, ACM New York, NY, USA, 2005. p. 667–673.

GARCIA-CAMINO, A., NORIEGA, P. and RODRIGUEZ-AGUILAR, J. A. *An Algorithm for Conflict Resolution in Regulated Compound Activities*. In: ESAW'06 Proceedings of the 7th international conference on Engineering societies in the agents world VII, Springer-Verlag Berlin, Heidelberg, 2007.

GARCIA-OJEDA, J., DELOACH, S., ROBBY, O. and VALENZUELA, J. O. *MaSE: a customizable approach to developing multiagent development processes*. In: Michael Luck (eds.), *Agent-Oriented Software Engineering VIII: The 8th*

International Workshop on Agent Oriented Software Engineering, LNCS 4951, Springer: Berlin, 2008. p. 1–15.

GIORGINI, P., MOURATIDIS, H. and ZANNONE, N. *Modelling security and trust with Secure Tropos*. In: Integrating Security and Software Engineering: Advances and Future Vision, 2006. p. 160-189.

GOVERNATORI, G. and ROTOLO, A. *Defeasible logic: agency, intention and obligation*. In: Deontic Logic in Computer Science, 7th International Workshop on Deontic Logic in Computer Science, LNAI 3065, Springer, 2004. p. 114–128.

HARMON, S. J. and DELOACH, S. A. *Trace-based Specification of Law and Guidance Policies for Multiagent Systems*. Engineering Societies in the Agents World VIII, Springer-Verlag Berlin, Heidelberg, 2008.

HÜBNER, J. F., SICHTMAN, J. S. and OLIVIER, B. *A model for the structural, functional and deontic specification of organizations in multiagent systems*. In: SBIA '02 Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence, Springer-Verlag London, UK, 2002.

INOUE, K. *Linear resolution for consequence finding*. Journal of Artificial Intelligence, Volume 56, Issue 2-3, Aug. 1992. p. 301-353.

JUAN, T., PIERCE, A. and STERLING, L. *ROADMAP: extending the Gaia methodology for complex open systems*. In: AAMAS '02 Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, ACM New York, NY, USA, 2002. p. 3–10.

KAGAL, L. and FININ, T. *Modeling Conversation Policies using Permissions and Obligations*. In: van Eijk, R., Huget, M., Dignum, F., eds.: Developments in Agent Communication Volume 3396 of LNCS., Springer, 2005. p. 123–133.

KOLLINGBAUM, M. and NORMAN, T. J. *Informed Deliberation During Norm-Governed Practical Reasoning*. In: Boissier, O and Padget, J and Dignum, V and Lindemann, G, Eds. Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, Springer-Verlag, 2006.

KOLLINGBAUM, M., NORMAN, T. J., PREECE, A. and SLEEMAN, D. *Norm Conflicts and Inconsistencies in Virtual Organisations*. In: Coordination, Organizations, Institutions, and Norms in Agent Systems II, Springer-Verlag, 2007.

KOLLINGBAUM, M., VASCONCELOS, W., GARCIA-CAMINO, A. and NORMAN, T. J. *Conflict Resolution in Norm-regulated Environments via Unification*

and Constraints. In: Proceedings of the 5th international conference on Declarative agent languages and technologies V, 2008.

LOMUSCIO, A. and SERGOT, M. *A formalization of violation, error recovery, and enforcement in the bit transmission problem*. Journal of Applied Logic, Volume 2, Number 1, 2004. p. 93–116.

LOPES-CARDOSO, H. and OLIVEIRA E. C. *A Context-based Institutional Normative Environment*. In: Coordination, Organizations, Institutions and Norms in Agent Systems IV, Springer-Verlag Berlin, Heidelberg, 2008.

LOPES-CARDOSO, H. and OLIVEIRA E. C. *Monitoring Directed Obligations with Flexible Deadlines: a Rule-based Approach*. In: Declarative Agent Languages and Technologies VII, 2010. p. 51-67.

LÓPEZ y LÓPEZ, F. *Social power and norms: impact on agent behavior*. PhD thesis, University of Southampton, Department of Electronics and Computer Science, 2003.

LÓPEZ y LÓPEZ, F., LUCK, M. and D'INVERNO, M. *Constraining autonomy through norms*. In: AAMAS '02 Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2, ACM New York, NY, USA, 2002. p. 674–681.

MANNA, Z. and PNUELI, A. (1992) *The Temporal Logic of Reactive and Concurrent Systems*. The temporal logic of reactive and concurrent systems, Springer-Verlag New York, Inc. New York, NY, USA, 1992.

MANNA, Z. and PNUELI, A. *Temporal Verification of reactive Systems – Safety*. Temporal verification of reactive systems: safety, Springer-Verlag New York, Inc. New York, NY, USA, 1995.

MEYER, J. J. and WIERINGA, R. J. *Deontic logic in computer science: normative system specification*. Deontic logic in computer science: normative system specification, John Wiley and Sons Ltd. Chichester, UK, 1993.

MODGIL, S. and LUCK, M. *Argumentation Based Resolution of Conflicts between Desires and Normative Goals*. In: Argumentation in Multi-Agent Systems, Springer-Verlag Berlin, Heidelberg, 2009.

MOLESINI, A., DENTI, E. and OMICINI, A. *RBAC-MAS & SODA: experimenting RBAC in AOSE engineering societies in the agents world*. In: Engineering Societies in the Agents World IX, Springer-Verlag Berlin, Heidelberg, 2009.

NOYA, R. C. and LUCENA, C. J. P. *The ANote Modeling Language for Agent-Oriented Specification*. In: Software Engineering for Multi-Agent Systems III, Lecture Notes in Computer Science, 2005, Volume 3390/2005, 2005. p. 198–212.

OBJECT MANAGEMENT GROUP. *Unified Modeling Language*. <http://www.uml.org/>, Accessed: Jan. 31, 2011. 2001a.

OBJECT MANAGEMENT GROUP. *OCL Specification*. <http://www.omg.org/docs/ptc/03-10-14.pdf>, Accessed: Jan. 31, 2011. 2011b.

ODELL, J., PARUNAK, H. and BAUER, B. *Extending UML for agents*. In: Proceedings of of the Agent-Oriented Information Systems, Workshop at the 17th National conference on Artificial Intelligence, 2000. p. 3–17.

OMG AGENT PLATFORM SPECIAL INTEREST GROUP. *Agent Technology Glossary*. <http://www.objs.com/agent/agent-glossary-v02.html>, Accessed: Jan. 31, 2011.

OMICINI, A. *SODA: societies and infrastructures in the analysis and design of agent-based systems*. In: First international workshop, AOSE 2000 on Agent-oriented software engineering, Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2001.

OREN, N., LUCK, M., MILES, S. and NORMAN, T. J. *An argumentation inspired heuristic for resolving normative conflict*. In: Proceedings of the International Workshop on Coordination, Organisations, Institutions and Norms in Agent Systems (COIN@AAMAS 2008), Estoril, Portugal, 2008. p. 41–56.

PADGHAM, L. and WINIKOFF, M. *Developing intelligent agent systems: a practical guide*. John Wiley and Sons, 2004. 225 pages.

PRABHUPADA, A. C. B. S. *Bhagavadgītā As It Is*. Ed. Bhaktivedanta Book Trust, 1968.

RAO, A. S. and GEORGEFF, M. P. *BDI agents: From theory to practice*. In: Proceedings of the First Intl. Conference on Multiagent Systems ICMAS95, 1995.

SECTRO. *SecTro Tool, Secure Tropos*. <http://sectro.securetropos.org/>, Accessed: Jan. 31, 2011.

SILVA, V. *From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behaviour of agents*. In: Autonomous Agents and Multi-Agent Systems, Volume 17, Issue 1, August 2008. p. 113–155.

SILVA, V., BRAGA, C. and FFIGUEIREDO, K. *A Modeling Language to Model Norms*. In: Workshop on Coordination, Organization, Institutions and Norms in agent

systems at International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS10), Toronto, 2010. p. 25-32.

SILVA, V., CHOREN R. and LUCENA, C. *MAS-ML: a multi-agent system modelling language*. In: International Journal of Agent-Oriented Software Engineering, Special Issue on Modeling Languages for Agent Systems, Inderscience Publishers, vol.2, no.4, 2008. p. 382-421.

SILVA, V., DURAN, F., GUEDES, J. and LUCENA, C. *Governing Multi-Agent Systems*. In: Journal of Brazilian Computer Society, Special Issue in Software Engineering for Multi-Agent Systems, volume 13, number 2, 2007.

SILVA, V., GARCIA, A., BRANDAO, A., CHAVEZ, C., LUCENA, C., ALENCAR, P. *Taming Agents and Objects in Software Engineering*. In: Garcia, A.; Lucena, C.; Zamboneli, F.; Omicini, A; Castro, J. (Eds.), Software Engineering for Large-Scale Multi-Agent Systems, Springer-Verlag, LNCS 2603, 2003. p. 1-26.

THE ECLIPSE FOUNDATION. *Eclipse*. <http://www.eclipse.org/>, Accessed: Jan. 31, 2011.

VASCONCELOS, W., KOLLINGBAUM, M. and NORMAN, T. *Resolving conflict and inconsistency in norm-regulated virtual organizations*. In: AAMAS '07 Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, ACM New York, NY, USA, 2007.

VIGANÒ, F. and COLOMBETTI, M. *Model checking norms and sanctions in institutions*. In: COIN'07 Proceedings of the 2007 international conference on Coordination, organizations, institutions, and norms in agent systems III, Springer-Verlag Berlin, Heidelberg, 2008.

VON WRIGHT, G.H. *Deontic Logic*. Mind, volume 60, 1951. 1–5.

W3SCHOOLS. *XSLT Tutorial*. <http://www.w3schools.com/xsl/>, Accessed: Jan. 31, 2011. 2011a.

W3SCHOOLS. *XPath Tutorial*. <http://www.w3schools.com/xpath/default.asp> xpath tutorial 2011, Accessed: Jan. 31, 2011. 2011b.

WAGNER, G. *The Agent-Object-Relationship meta-model: towards a unified view of state and behavior*. Information Systems, Volume 28 Issue 5, July 2003. p. 475–504.

WOOLDRIDGE, M. *Agent-based software engineering*. IEE Proceedings – Software, 144(1), 1997. p. 26-37.

ZAMBONELLI, F., JENNINGS, N. R. and WOOLDRIDGE, M. J. *Developing multiagent systems: the Gaia methodology*. In: ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 12, Issue 3, July 2003. p. 417–470.

ZAMBONELLI, F., JENNINGS, N. R. and WOOLDRIDGE, M. J. *Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems*. In: International Journal of Software Engineering and Knowledge Engineering, Volume 11, Number 3, 2001. p. 303-328.

APPENDIX A: The NormML extension of SecureUML

NormML is a non-conservative extension of the SecureUML language, and, for this reason, not all elements of SecureUML were maintained in the extension. In the following we present the main differences between the two modeling languages.

The metamodel extension

Figure A.1 shows the elements from the SecureUML metamodel highlighting the ones which were preserved in the NormML metamodel. The gray colored metaclasses and the relationships illustrated by a continuous line are still part of the NormML metamodel. The white colored classes and the relationships illustrated by a traced line were excluded during the NormML extension.

The *User* metaclass and all the attributes of the metaclasses of SecureUML were removed, and the *Permission* and the *AutorizationConstraint* metaclasses of SecureUML were replaced by the *Norm* and *NormConstraint* metaclasses of NormML. Because of that some relationships were also modified:

- The *ConstraintAssignment* relationship between the *Permission* and the *AutorizationConstraint* metaclasses defined in SecureUML was replaced by the *NormConstraintAssignment* relationship between the *Norm* and *NormConstraint* metaclasses in NormML;
- The *PermissionAssignment* relationship between the *Permission* and the *Role* metaclasses defined in SecureUML was replaced by the *NormAssignmentRole* relationship between the *Norm* and *Role* metaclasses in NormML; and
- The *UserAssignment* relationship between the *Role* and the *User* metaclasses defined in SecureUML was replaced by the *AgentPlayingRole* relationship between the *Agent* and *Role* metaclasses and the

The well-formedness rules extension

As a non-conservative extension of the SecureUML, the invariants “Default role” and “Default permission” of the SecureUML language (Basin *et al.*, 2006) were discarded in the NormML language since they cannot be applied in the metaclasses and relationships of the NormML metamodel.

The “Role hierarchy” invariant that guarantees an acyclic role hierarchy was maintained without changes (see WRF07 in Appendix C), and the “Resource action association” invariants were extended to attend the new types of resources and actions of the NormML metamodel (see WFR16 to WFR48 in Appendix C and Table 3.1 in Chapter 3). For instance, an *AtomicCreate* action must be related to an *Entity* in SecureUML. In NormML, an *AtomicCreate* action must be related to an *Entity*, an *Agent*, a *Role*, a *Plan*, a *Belief*, a *Protocol*, an *Organization* or an *Environment* resource as described below.

context AtomicCreate

inv atomicCreateTargetsCorrectResource:

```
if((self.resource.ocIsTypeOf(Entity) or (self.resource.ocIsTypeOf(Agent) or
(self.resource.ocIsTypeOf(Role) or (self.resource.ocIsTypeOf(Plan) or
(self.resource.ocIsTypeOf(Belief) or (self.resource.ocIsTypeOf(Protocol) or
(self.resource.ocIsTypeOf(Organization) or (self.resource.ocIsTypeOf(Environment)))
then(true)else(false)endif
```

The “Action hierarchy” invariants were also extended to attend the new types of composite and atomic actions of the NormML metamodel (see Table B.1 in Appendix B). For instance, the following invariant was added to guarantee that the new composite action *MessageFullAccess* of NormML metamodel are composed of the correct subordinated actions (*AtomicSend* and *AtomicReceive*).

context MessageFullAccess

inv containsSubactions:

```
self.subordinatedactions = self.resource.action->select(a|a.ocIsTypeOf(AtomicSend))
->union(self.resource.action->select(a|a.ocIsTypeOf(AtomicReceive)))
```

APPENDIX B: NormML dialect action hierarchy

In the NormML metamodel each resource kind has a set of actions that can be used to control the access to the resource. Those actions can be atomic or composite actions and they are connected by using hierarchies. The composite actions are composed by other atomic or composite actions, according to the relations between the resources. In Table B.1 a mapping between the composite actions and its subordinate actions is described.

Composite action type	Subordinated actions
EntityRead	<i>read</i> for all attributes and association ends of the entity, and <i>execute</i> for all side-effect free methods of the entity.
EntityUpdate	<i>update</i> for all attributes of the entity, <i>update</i> for all association ends of the entity, and <i>execute</i> for all non-side-effect free methods of the entity
EntityFullAccess	<i>create</i> , <i>read</i> , <i>update</i> , and <i>delete</i> of the entity.
AttributeFullAccess	<i>read</i> , <i>update</i> and <i>achieve</i> of the attribute.
AssociationEndFullAccess	<i>read</i> and <i>update</i> of the association end.
EnvironmentUpdate	<i>full access</i> for all organizations of the environment, and <i>full access</i> for all agents of the environment.
EnvironmentFullAccess	<i>create</i> , <i>delete</i> , <i>enter</i> , <i>leave</i> and <i>update</i> of the environment.
OrganizationUpdate	<i>full access</i> for all sub-organizations of the organization, and <i>full access</i> for all roles of the organization.
OrganizationFullAccess	<i>create</i> , <i>delete</i> , <i>enter</i> , <i>leave</i> and <i>update</i> of the organization.
AgentUpdate	<i>full access</i> for all beliefs of the agent, <i>full access</i> for all goals of the agent, <i>full access</i> for all agent actions of the agent, and <i>full access</i> for all plans of the agent.
AgentFullAccess	<i>create</i> , <i>delete</i> and <i>update</i> of the agent.
PlanExecute	<i>execute</i> for all agent actions of the plan.
PlanFullAccess	<i>create</i> , <i>delete</i> , <i>update</i> and <i>execute</i> of the plan.
BeliefFullAccess	<i>create</i> , <i>delete</i> and <i>update</i> of the belief.
GoalFullAccess	<i>achieve</i> , <i>commit</i> and <i>cancel</i> of the goal.
RoleUpdate	<i>full access</i> for all protocols of the role, <i>full access</i> for all belief of the role, <i>full access</i> for all goals of the role, and <i>full access</i> for all agent actions of the role.

Composite action type	Subordinated actions
RoleFullAccess	<i>create, delete, commit, cancel and update</i> of the role.
MessageFullAccess	<i>receive</i> and <i>send</i> of the message.
ProtocolSend	<i>send</i> for all messages of the protocol.
ProtocolReceive	<i>receive</i> for all messages of the protocol.
ProtocolFullAccess	<i>create, delete, send</i> and <i>receive</i> of the protocol.

Table B.1 NormML dialect action hierarchy

APPENDIX C: The well-formedness rules of NormML

WFR1: A norm must be in the context of an Organization or an Environment and cannot be defined in the scope of both at the same time.

The context of a norm determines the scope where the norm is applied, thus, a norm can have only one context that can be an environment or an organization (see Figure 3.10).

WFR2: An Agent can only play a Role in the Organization that owns such Role, i.e., that has defined such Role.

To play a role, an agent must belong to the organization that owns such role by the relationship AgentOfOrganization (see Figure 3.10).

WFR3: Only Suborganizations play Roles.

Organizations must belong to another organization to play its roles, thus only sub-organizations can play roles in their super-organizations.

WFR4: A SubOrganization can only play a Role in the Organization that owns such Role, i.e., that has defined such Role.

To play a role, a sub-organization must belong to a super-organization that owns such role by the relationship OrganizationComposition (see Figures 3.2 and 3.9).

WFR5: A SubOrganization must inhabit the same Environment of its super-organization.

If an organization is a sub-organization then it cannot inhabit an environment different of its super-organization's environment, because the sub-organization is part of the super-organization.

WFR6: A norm must restrict the behavior of an Agent, a Role, an Agent playing a Role, an Organization, a SubOrganization playing a Role or an Environment.

A norm must restrict the behavior of an entities, and, according to the NormML metamodel, these are the possible involved entities of the norm (see Section 3.2.2).

WFR7: The subRole of a RoleHierarchy cannot be the superRole of the same RoleHierarchy.

This rule is necessary to avoid cycles in the role hierarchy.

WFR8: The norms applied to an Agent restrict the actions of such Agent.

A norm must restrict the behavior of the entity involved in the norm, so a norm cannot regulate the access of an agent action that is not one of the actions of the agent itself.

WFR9: The norms applied to a Role restrict the actions of the Agents playing such Role.

A norm must restrict the behavior of the entity involved in the norm, so the behavior of the agents that play the role will be regulated by the norms applied to the role they are playing. .

WFR10: The norms applied to an Agent playing a Role restrict the actions defined by the Agent when playing such Role.

A norm must restrict the behavior of the entity involved in the norm, so the behavior of the agent will only be regulated by such norm when it is playing such role. Note that it will occur to the agents identified in the norm.

WFR11: The norms applied to an Organization restrict the actions of all Agents that play Roles in such Organization and its SubOrganizations.

A norm must restrict the behavior of the entity involved in the norm, so at least one agent that plays roles in such organization or its sub-organizations must have the action regulated by the norm.

WFR12: The norms applied to an Environment restrict the actions of all the Agents of such Environment.

A norm must restrict the behavior of the entity involved in the norm, so at least one agent that inhabits such environment must have the action regulated by the norm.

WFR13: The norms applied to an Environment must be defined in the context of such Environment.

If a norm regulates the behavior of the agents that inhabits an environment, it cannot be defined in the context of an organization or of another environment because it will be regulating the behavior of entities in a reduced scope or out of its appropriated scope.

WFR14: The norms applied to an Organization must be defined in the context of such Organization, in the context of its organization hierarchy, or in the context of the Environment inhabited by the Organization.

If a norm regulates the behavior of the agents that play roles in an organization, it cannot be defined in the context of another organization out of its organization hierarchy or in the context of an environment different of the organization's environment because it will be regulating the behavior of entities of out of its scope.

WFR15: The subOrganization of an OrganizationComposition cannot be the superOrganization of the same OrganizationComposition.

This rule is necessary to avoid cycles in the organization composition.

WFR16: An AtomicCreate action must be related to an Entity, an Agent, a Role, a Plan, a Belief, a Protocol, an Organization or an Environment resource.

WFR17: An AtomicUpdate action must be related to an Attribute, an AssociationEnd, a Plan or a Belief resource.

WFR18: An AtomicDelete action must be related to an Entity, an Agent, a Role, a Plan, a Belief, a Protocol, an Organization or an Environment resource.

WFR19: An AtomicRead action must be related to an Attribute or an AssociationEnd resource.

WFR20: An AtomicExecute action must be related to a Method or an AgentAction resource.

WFR21: An AtomicReceive action must be related to a Message resource.

- WFR22:** An AtomicSend action must be related to a Message resource.
- WFR23:** An AtomicAchieve action must be related to an Attribute or a Goal resource.
- WFR24:** An AtomicEnter action must be related to an Organization or an Environment resource.
- WFR25:** An AtomicLeave action must be related to an Organization or an Environment resource.
- WFR26:** An AtomicCommit action must be related to a Role or a Goal resource.
- WFR27:** An AtomicCancel action must be related to a Role or a Goal resource.
- WFR28:** An EntityRead action must be related to an Entity resource.
- WFR29:** An EntityUpdate action must be related to an Entity resource.
- WFR30:** An EntityFullAccess action must be related to an Entity resource.
- WFR31:** An AttributeFullAccess action must be related to an Attribute resource.
- WFR32:** An AssociationEndFullAccess action must be related to an AssociationEnd resource.
- WFR33:** A MessageFullAccess action must be related to a Message resource.
- WFR34:** An AgentUpdate action must be related to an Agent resource.
- WFR35:** An AgentFullAccess action must be related to an Agent resource.
- WFR36:** A RoleUpdate action must be related to a Role resource.
- WFR37:** A RoleFullAccess action must be related to a Role resource.
- WFR38:** An OrganizationUpdate action must be related to an Organization resource.
- WFR39:** An OrganizationFullAccess action must be related to an Organization resource.
- WFR40:** An EnvironmentUpdate action must be related to an Environment resource.
- WFR41:** An EnvironmentFullAccess action must be related to an Environment resource.
- WFR42:** A PlanExecute action must be related to a Plan resource.
- WFR43:** A PlanFullAccess action must be related to a Plan resource.
- WFR44:** A ProtocolReceive action must be related to a Protocol resource.
- WFR45:** A ProtocolSend action must be related to a Protocol resource.
- WFR46:** A ProtocolFullAccess action must be related to a Protocol resource.
- WFR47:** A BeliefFullAccess action must be related to a Belief resource.
- WFR48:** A GoalFullAccess action must be related to a Goal resource.

The rules **WFR16** to **WFR28** match the actions to their correct resources (see Section 3.2.3, Table 3.1 and Appendix A for more details).

- WFR49:** The subAgentAction of an AgentActionHierarchy cannot be the superAgentAction of the same AgentActionHierarchy.
This rule is necessary to avoid cycles in the agent action hierarchy.
- WFR50:** The subordinateAgentAction of an AgentActionComposition cannot be the compositeAgentAction of the same AgentActionComposition.
This rule is necessary to avoid cycles in the agent action composition.
- WFR51:** A Message must belong to a Protocol as MessageReceivedByProtocol or MessageSentByProtocol.
A message cannot be apart from an interaction protocol (see Figure 3.6).

WFR52: A Goal is the goal of an Agent or a Role.

A goal must belong to an entity. Such entity can be an agent identified by the GoalOfAgent relationship or a role identified by the GoalOfRole relationship (see Figure 3.6).

WFR53: A Belief is the belief of an Agent or a Role.

A belief must belong to an entity. Such entity can be an agent identified by the BeliefOfAgent relationship or a role identified by the BeliefOfRole relationship (see Figure 3.6).

WFR54: An AgentAction is the action of an Agent or an action of a Role being played.

A goal must belong to an entity. Such entity can be an agent identified by the ActionOfAgent relationship or a role identified by the ActionOfRoleBeingPlayed relationship (see Figure 3.6).

WFR55: A norm cannot have more than one Before, After, Between and If constraints.

A norm can only have one norm constraint of each type to avoid duplications. In case of more than one clause need to be described to a norm constraint type, it can be associated with the same norm constraint.

WFR56: A Before or an After constraint must be related to one Date or Action.

The before and after of a norm are time constraints, thus they need to be associated with a date or with the execution of an action (see Figure 3.7).

WFR57: The before and the after of a Between constraint must be related to one Date or Action.

The between of a norm is a time constraint, thus it need to be associated with two dates or with the execution of two actions in order to define a time interval (see Figure 3.7).

WFR58: The Action of an entity in the before of a Between constraint cannot be in the after of the same Between to the same entity in the same context, and vice-versa.

If the action in the before of the between is equal to the action in the after of the same between to the same entity in the same context, then the between does not constitute a valid time interval.

WFR59: The Date in the before of a Between constraint cannot be equal or superior to the Date in the after of the same Between.

If the date in the before of the between is equal or superior to the date in the after of the same between, then the between does not constitute a valid time interval.

WFR60: A If constraint must be related to one Date or two Operands.

The if constraint is a time constraint and also a conditional constraint, thus it needs to be associated with a date or with a clause with values (see Section 3.2.4).

WFR61: If a Norm has an Attribute related to its If constraint, then the entity of the norm must have permission to read this Attribute.

If the entity of the norm does not have permission to read the attribute, it will not know when to fulfill the norm.

WFR62: A Norm that regulates the execution of a given Action cannot be conditioned by the execution of the same Action by the same entity.

This rule is necessary to avoid cycles in the norm constitution.

WFR63: The value and the operator attributes of a Value and a If cannot be null.

This rule is necessary to make mandatory the attributes above, thus the norm can be read.

WFR64: A Norm described in a Sanction cannot be the same Norm that has the Sanction.

This rule is necessary to avoid cycles in the norm's sanctions constitution.

WFR65: A Reward to an entity cannot apply a NormProhibition or a NormObligation to the same entity.

A reward must be a prize to the entity that fulfilled the norm, thus it does not make sense to apply a prohibition or an obligation to the same entity.

WFR66: A Punishment to an entity cannot apply a NormPermission to the same entity.

A punishment must be a penalty to the entity that violated the norm, thus it does not make sense to apply a permission to the same entity.

APPENDIX D: Semantically opposite actions

In the NormML metamodel each resource kind has a set of actions that can be used to restrict the access to the resource. Some of those actions are semantically opposite actions. In Table E.1 a mapping between semantically opposite actions and the resources they control is described.

Semantically opposite actions	Resources
AtomicCreate and AtomicDelete	Entity, Agent, Role, Plan, Belief, Protocol, Organization and Environment
AtomicEnter and AtomicLeave	Organization and Environment
AtomicCommit and AtomicCancel	Role and Goal
AtomicSend and AtomicReceive	Message
ProtocolSend and ProtocolReceive	Protocol

Table D.1 NormML semantically opposite actions

APPENDIX E: List of the graphical model stereotypes of the NormML concrete syntax

In Table E.1 a mapping between graphical model stereotypes and what they represent is described.

Stereotype	Represents
<<Environment>>	An Environment element
<<Organization>>	An Organization element
<<Agent>>	An Agent element
<<Role>>	A Role element
<<Entity>>	An Entity element
<<Norm>>	A Norm element
<<Permission>>, <<Prohibition>> and <<Obligation>>	The type (or deontic concept) of a Norm element
<<Reward>> and <<Punishment>>	The sanction type of a Norm element
<<agentAction>>, <<agentActionAction>>, <<beliefAction>>, <<goalAction>>, <<planAction>>, <<roleAction>>, <<messageAction>>, <<protocolAction>>, <<environmentAction>>, <<organizationAction>>, <<entityAction>>, <<attributeAction>>, <<methodAction>> and <<associationEndAction>>	The resource type being accessed by a Norm element
<<beforeAction>>, <<afterAction>>, <<betweenBeforeAction>> and <<afterBetweenAction>>	An <i>action constraint</i> attribute of a Norm element
<<conditionalAttribute>>, <<conditionalGoal>> and <<conditionalBelief>>	A <i>conditional constraint</i> attribute of a Norm element
<<before>>, <<after>>, <<between>> and <<if>>	A <i>date constraint</i> attribute of a Norm element

Table E.1 NormML graphical model stereotypes

APPENDIX F: From NormML concrete models to abstract models

- For each *Environment* env of M , insert in \overline{M} an object \overline{env} of the class *Environment*.
- For each *Organization* org of M , insert in \overline{M} an object \overline{org} of the class *Organization*.
- For each *OrganizationInhabitEnvironment* relationship of M between env and org , insert in \overline{M} an *OrganizationInhabitEnvironment* link between \overline{env} and \overline{org} .
- For each composition relationship between two organizations org^1 (suborganization) and org^2 of M , insert in \overline{M} an *OrganizationComposition* link between $\overline{org^1}$ (suborganization) and $\overline{org^2}$.
- For each *Entity* e of M , insert in \overline{M} an object \overline{e} of the class *Entity* and, for each *Attribute* a of an *Entity* e of M , insert in \overline{M} (i) an object \overline{a} of the class *Attribute* and (ii) an *EntityAttribute* link between \overline{a} and \overline{e} . Also, for each *Method* m of an *Entity* e of M , insert in \overline{M} (i) an object \overline{m} of the class *Method* and (ii) an *EntityMethod* link between \overline{m} and \overline{e} .
- For each *Association* relationship ass of M between e^1 and e^2 , insert in \overline{M} (i) an object \overline{ass} of the class *Association*; (ii) two objects $\overline{ass-end^1}$ and $\overline{ass-end^2}$ of the class *AssociationEnd*; (iii) an *AssocEndAssoc* link between $\overline{ass-end^1}$ and \overline{ass} ; (iv) an *AssocEndAssoc* link between $\overline{ass-end^2}$ and \overline{ass} ; (v) an *EntityAssocEnd* link between $\overline{ass-end^1}$ and $\overline{e^1}$; and (vi) an *EntityAssocEnd* link between $\overline{ass-end^2}$ and $\overline{e^2}$.
- For each *Agent* ag of M , insert in \overline{M} an object \overline{ag} of the class *Agent*.
- For each *AgentOfOrganization* relationship of M between org and ag , insert in \overline{M} an *AgentOfOrganization* link between \overline{org} and \overline{ag} .
- For each *AgentInhabitEnvironment* relationship of M between env and ag , insert in \overline{M} an *AgentInhabitEnvironment* link between \overline{env} and \overline{ag} .
- For each *Belief* b of an *Agent* ag of M , insert in \overline{M} (i) an object \overline{b} of the class *Belief* and (ii) a *BeliefOfAgent* link between \overline{b} and \overline{ag} .
- For each *Goal* g of an *Agent* ag of M , insert in \overline{M} (i) an object \overline{g} of the class *Goal* and (ii) a *GoalOfAgent* link between \overline{g} and \overline{ag} .
- For each *AgentAction* $ag-act$ of an *Agent* ag of M , insert in \overline{M} (i) an object $\overline{ag-act}$ of the class *AgentAction* and (ii) an *ActionOfAgent* link between $\overline{ag-act}$ and \overline{ag} .
- For each *Plan* p of an *Agent* ag of M , insert in \overline{M} (i) an object \overline{p} of the class *Plan*; (ii) a *PlanOfAgent* link between \overline{p} and \overline{ag} ; (iii) a *ActionOfPlan* link between \overline{p} and $\overline{ag-act}$ for each agent action $ag-act$ of p ; and (iv) a *GoalOfPlan* link between \overline{p} and \overline{g} for each goal g of p .

- For each *Role* r of M , insert in \overline{M} an object \overline{r} of the class *Role*.
- For each *RoleOfOrganization* relationship of M between org and r , insert in \overline{M} a *RoleOfOrganization* link between \overline{org} and \overline{r} .
- For each inheritance relationship between two roles r^1 (subrole) and r^2 of M , insert in \overline{M} a *RoleHierarchy* link between $\overline{r^1}$ (subrole) and $\overline{r^2}$.
- For each *Protocol* pro of a *Role* r of M , insert in \overline{M} (i) an object \overline{pro} of the class *Protocol*; (ii) a *ProtocolOfRole* link between \overline{pro} and \overline{r} ; (iii) an object \overline{mess} of the class *Message* and a *MessageSentByProtocol* link between \overline{pro} and \overline{mess} for each “sent message” $mess$ of pro ; and (iv) an object \overline{mess} of the class *Message* and a *MessageReceivedByProtocol* link between \overline{pro} and \overline{mess} for each “received message” $mess$ of pro .
- For each *Belief* b of a *Role* r of M , insert in \overline{M} (i) an object \overline{b} of the class *Belief* and (ii) a *BeliefOfRole* link between \overline{b} and \overline{r} .
- For each *Goal* g of a *Role* r of M , insert in \overline{M} (i) an object \overline{g} of the class *Goal* and (ii) a *GoalOfRole* link between \overline{g} and \overline{r} .
- For each *AgentAction* $ag-act$ of a *Role* r of M , insert in \overline{M} (i) an object $\overline{ag-act}$ of the class *AgentAction* and (ii) an *ActionOfRoleBeingPlayed* link between $\overline{ag-act}$ and \overline{r} .
- For each *AgentPlayingRole* relationship of M between ag and r , insert in \overline{M} an *AgentPlayingRole* link between \overline{ag} and \overline{r} .
- For each *SubOrgPlayingRole* relationship of M between org and r , insert in \overline{M} an *SubOrgPlayingRole* link between \overline{org} and \overline{r} .
- For each *Norm* n of M that states a permission, insert in \overline{M} an object \overline{n} of the class *NormPermission*.
- For each *Norm* n of M that states a prohibition, insert in \overline{M} an object \overline{n} of the class *NormProhibition*.
- For each *Norm* n of M that states an obligation, insert in \overline{M} an object \overline{n} of the class *NormObligation*.
- For each *NormInContextOrganization* relationship of M between org and n , insert in \overline{M} a *NormInContextOrganization* link between \overline{org} and \overline{n} .
- For each *NormInContextEnvironment* relationship of M between env and n , insert in \overline{M} a *NormInContextEnvironment* link between \overline{env} and \overline{n} .
- For each *NormAssignmentAgent* relationship of M between ag and n , insert in \overline{M} a *NormAssignmentAgent* link between \overline{ag} and \overline{n} .
- For each *NormAssignmentRole* relationship of M between r and n , insert in \overline{M} a *NormAssignmentRole* link between \overline{r} and \overline{n} .
- For each *NormAssignmentOrganization* relationship of M between org and n , insert in \overline{M} a *NormAssignmentRole* link between \overline{org} and \overline{n} .
- For each *NormAssignmentEnvironment* relationship of M between env and n , insert in \overline{M} a *NormAssignmentRole* link between \overline{env} and \overline{n} .

- For each *NormAssignmentAgentPlayingRole* relationship of M between r , ag and n , insert in \overline{M} a *NormAssignmentAgentPlayingRole* link between \overline{r} , \overline{ag} and \overline{n} .
- For each *NormAssignmentSubOrgPlayingRole* relationship of M between r , org and n , insert in \overline{M} a *NormAssignmentSubOrgPlayingRole* link between \overline{r} , \overline{org} and \overline{n} .
- For each *resource action* attribute res of *Norm* n of M , must be inserted in \overline{M} (i) an object \overline{act} of the class *Action* related to the action type of res ; (ii) an *ActionAssignmentNorm* link between \overline{act} and \overline{n} ; and (iii) a *RessourceAssignment* link between \overline{act} and the object of the type defined in the stereotype of res which name is equal to res .
E.g.: For each *resource action* attribute res of *Norm* n of M that is an “attributeAction” and has the action type “update”, insert in \overline{M} (i) an object \overline{act} of the class *AtomicUpdate*; (ii) an *ActionAssignmentNorm* link between \overline{act} and \overline{n} ; and (iii) a *RessourceAssignment* link between \overline{act} and the object \overline{a} which name is equal to res .
- For each *action constraint* attribute $acon$ of *Norm* n of M that has a res , must be inserted in \overline{M} (i) an object \overline{act} of the class *Action* related to the action type of res ; (ii) a *norm constraint* object of the type defined in the stereotype of $acon$; (iii) the correct link between \overline{act} and the *norm constraint* object; (iv) a *NormConstraintAssignment* link between \overline{n} and the *norm constraint* object; and (v) a *RessourceAssignment* link between \overline{act} and the object of the type defined in the stereotype of res which name is equal to res .
E.g.: For each *action constraint* attribute $acon$ of *Norm* n of M that is a “beforeAction” and has a res that is a “messageAction” and has the action type “receive”, insert in \overline{M} (i) an object \overline{act} of the class *AtomicReceive*; (ii) an object \overline{bef} of the class *Before*; (iii) a *BeforeAction* link between \overline{act} and the object \overline{bef} ; (iv) a *NormConstraintAssignment* link between \overline{n} and \overline{bef} ; and (v) a *RessourceAssignment* link between \overline{act} and the \overline{mess} object which name is equal to $acon$.
- For each *conditional constraint* attribute $ccon$ of *Norm* n of M , must be inserted in \overline{M} (i) an object \overline{v} of the class *Value* if $ccon$ has a value v defined as its initial value; (ii) an object \overline{if} of the class *If*; (iii) a *NormConstraintAssignment* link between \overline{if} and the object \overline{n} ; (iv) a *ConditionalOperand* link between \overline{n} , the object of the type defined in the stereotype of $ccon$ which name is equal to $ccon$ and the object which name is equal to the initial value of $ccon$; and (v) the value of the type of $ccon$ as the value of the *operator* attribute of the object \overline{if} .
E.g.: For each *conditional constraint* attribute $ccon$ of *Norm* n of M that is a “conditionalAttribute” and has a value v defined as its initial value, insert in \overline{M} (i) an object \overline{v} of the class *Value*; (ii) an object \overline{if} of the class *If*; (iii) a *NormConstraintAssignment* link between \overline{if} and \overline{n} ; (iv) a

ConditionalOperand link between \bar{n} , \bar{v} and the object \bar{a} which name is equal to *ccon*; and (v) the value of the type of *ccon* as the value of the *operator* attribute of the object \bar{if} .

- For each *date constraint* attribute *dcon* of *Norm n* of *M*, must be inserted in \bar{M} (i) a *norm constraint* object of the type defined in the stereotype of *dcon*; (ii) an object \bar{d} of the class *Date*; (iii) the correct link between \bar{d} and the *norm constraint* object; and (iv) a *NormConstraintAssignment* link between \bar{n} and the *norm constraint* object.

E.g.: For each *date constraint* attribute *dcon* of *Norm n* of *M* that is a “before”, insert in \bar{M} (i) a \bar{bef} object of the class *Before*; (ii) an object \bar{d} of the class *Date*; (iii) a *BeforeDate* link between \bar{d} and \bar{bef} ; and (iv) a *NormConstraintAssignment* link between \bar{n} and \bar{bef} .
- For each *Norm n¹* of *M* that states a reward, insert in \bar{M} (i) a new object \bar{rew} of the class *Reward*; (ii) a *SanctionAppliesNorm* link between \bar{n}^1 and \bar{rew} ; and (iii) a *SanctionOfNorm* link between \bar{n}^2 and \bar{rew} where n^2 comes from the *SanctionOfNorm* relationship of *M* between n^1 and n^2 .
- For each *Norm n¹* of *M* that states a punishment, insert in \bar{M} (i) a new object \bar{pun} of the class *Punishment*; (ii) a *SanctionAppliesNorm* link between \bar{n}^1 and \bar{pun} ; and (iii) a *SanctionOfNorm* link between \bar{n}^2 and \bar{pun} where n^2 comes from the *SanctionOfNorm* relationship of *M* between n^1 and n^2 .

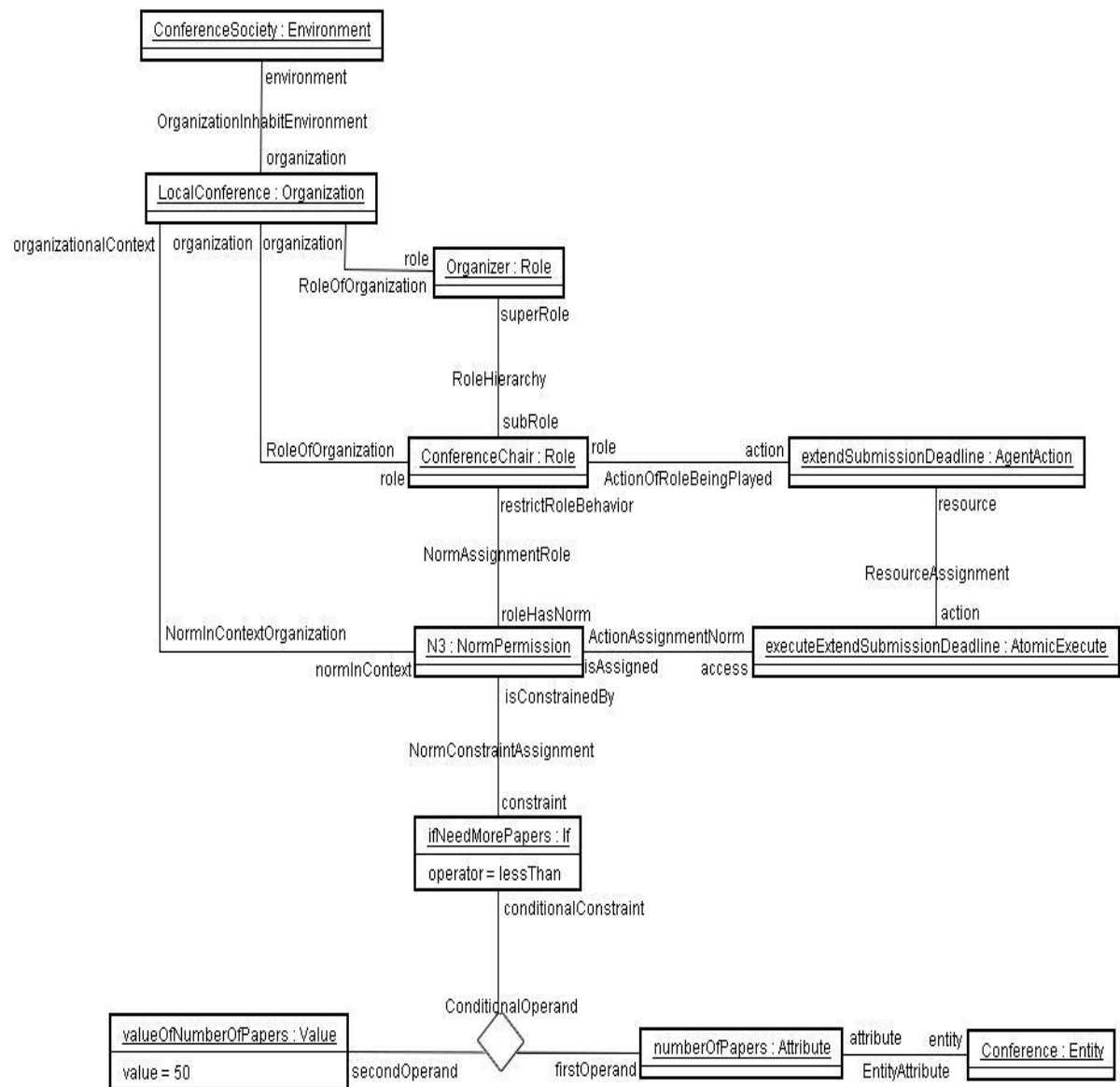


Figure G.2 N3

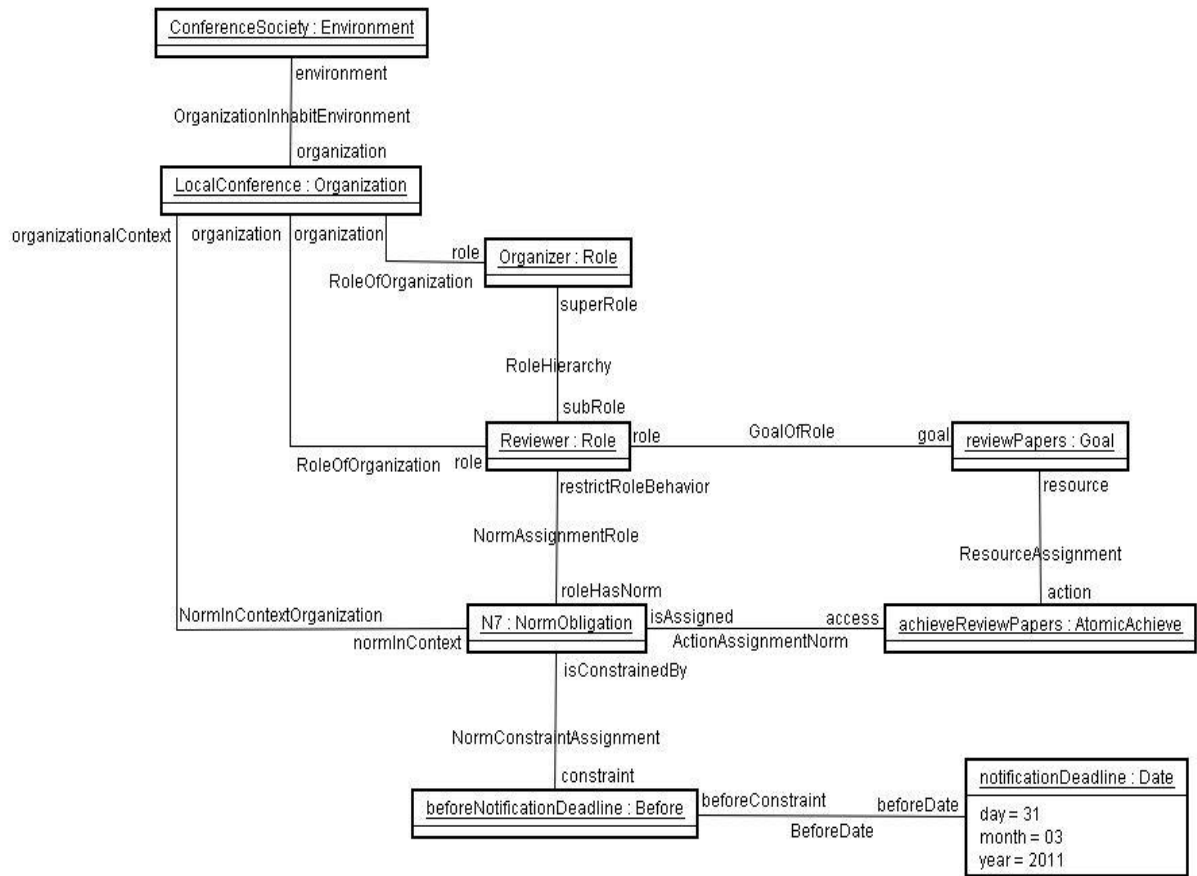


Figure G.5 N7

APPENDIX H: Conflict Cases

In the following we illustrate all the NormML conflict cases implemented by the check for conflicts operations. The check for conflicts of NormML was presented before in Section 3.4.

Context

Let's consider the conflict cases of the *context* analysis. It is important to check for conflicts: (i) if the norms are defined in the same context; (ii) if one norm is defined in the context of an environment, and the other in the context of an organization that inhabits such an environment; and (iii) if one norm is defined in the context of an organization and the other in the same hierarchy of organizations.

The context of a norm determines the scope where the norm is applied, i.e., the scope where the agents must fulfill the norm. According to the NormML metamodel (see Figure 3.10), a norm can be defined in the context of an environment or in the context of an organization. Organizations inhabit environments and might be composed of sub-organizations. Thus, let's consider the set of possible context relations between two norms.

Possible cases	First norm context	Second norm context
(a)	An environment <i>X</i>	An environment <i>X</i>
(b)	An environment <i>X</i>	An environment <i>Y</i>
(c)	An organization <i>a</i>	An organization <i>a</i>
(d)	An organization <i>a</i>	An organization <i>b</i>
(e)	An organization <i>a</i>	An organization <i>c</i> that is sub-organization of the organization <i>a</i>
(f)	An organization <i>a</i>	An organization <i>c</i> that is sub-organization of the organization <i>b</i>
(g)	An environment <i>X</i>	An organization <i>a</i> that inhabits the environment <i>X</i>
(h)	An environment <i>X</i>	An organization <i>a</i> that inhabits the environment <i>Y</i>

Table H.1 Possible context relations between two norms

When analyzing the contexts of two norms we need to observe if their contexts are related, i.e. if the scope of their application intersects. If the contexts of the norms

are not related, there is no need to keep looking for conflicts because the norms defined in not related contexts are not related to each other, and thus cannot conflict.

The cases (a), (c), (e) and (g) of Table H.1 may result in conflicts because the scope of application of the two norms intersects: the cases (a) and (c) are covered by item (i); the case (g) is covered by item (ii); and the case (e) is covered by item (iii). The cases (b), (d), (f) and (h) cannot result in conflicts because the scope of the first norm will never intersect with the scope of the second norm, so the norms contexts are not related.

Involved Entities

Let's consider the conflict cases of the *involved entities* of the norms. The involved entities of a norm are the entities whose behavior is being restricted by the norm. As illustrated in Figure 3.4 of chapter XXXX, a norm in NormML can regulate the behavior of: an agent, a role (i.e. all agents that play a given role), a specific agent when it is playing a given role, an organization (i.e. all agents that play roles in an organization), an sub-organization when it is playing a role (i.e. all agents that play roles in an sub-organization while such sub-organization is playing a role in its super-organization) and, an environment (i.e. all agents that inhabit an environment).

Therefore, It is necessary to check for conflicts: (i) between norms applied to the same entity; (ii) between a norm defined to a role and a norm defined to an agent or a sub-organization that can play that role; (iii) between norms applied to different roles played by the same agent or sub-organization; (iv) between norms applied to roles in the same hierarchy of roles; (v) between the norms of an organization and norms of roles, agents and sub-organizations of this organization; and (vi) between the norms of an environment and norms of agents and organizations of this environment. Table H.2 illustrates the set relationships between the involved entities of two norms.

Possible cases	First norm involved entity	Second norm involved entity
(a)	An agent <i>ag1</i>	An agent <i>ag1</i>
(a')	An agent <i>ag1</i>	An agent <i>ag2</i>
(b)	A role <i>r1</i>	A role <i>r1</i>

Possible cases	First norm involved entity	Second norm involved entity
(b')	A role $r1$	A role $r2$
(c)	A role $r1$	A role $r3$ that is sub-role of the role $r1$
(c')	A role $r1$	A role $r3$ that is sub-role of the role $r2$
(d)	A role $r1$	An agent $ag1$ that can play the role $r1$
(d')	A role $r2$	An agent $ag1$ that cannot play the role $r2$
(e)	A role $r1$	A sub-organization $s1$ that can play the role $r1$
(e')	A role $r2$	A sub-organization $s1$ that cannot play the role $r2$
(f)	A role $r2$ that can be played by an agent $ag1$	A role $r3$ that can also be played by the agent $ag1$
(f')	A role $r2$ that can be played by an agent $ag1$	A role $r1$ that cannot be played by the agent $ag1$
(g)	A role $r2$ that can be played by a sub-organization $s1$	A role $r3$ that can also be played by the sub-organization $s1$
(g')	A role $r2$ that can be played by a sub-organization $s1$	A role $r1$ that cannot be played by the sub-organization $s1$
(h)	An agent $ag1$ while playing a given role $r1$	An agent $ag1$ while playing the same given role $r1$
(h')	An agent $ag1$ while playing a given role $r1$	An agent $ag1$ while playing a given role $r2$
(h'')	An agent $ag1$ while playing a given role $r1$	An agent $ag2$ while playing the same given role $r1$
(i)	A sub-organization $s1$ while playing a given role $r1$	A sub-organization $s1$ while playing the same given role $r1$
(i')	A sub-organization $s1$ while playing a given role $r1$	A sub-organization $s1$ while playing a given role $r2$
(i'')	A sub-organization $s1$ while playing a given role $r1$	A sub-organization $s2$ while playing the same given role $r1$
(j)	An organization $org1$	An organization $org1$
(j')	An organization $org1$	An organization $org2$
(k)	An organization $org1$	An agent $ag1$ of the organization $org1$
(k')	An organization $org1$	An agent $ag1$ of the organization $org2$
(l)	An organization $org1$	A role $r1$ of the organization $org1$
(l')	An organization $org1$	A role $r1$ of the organization $org2$
(m)	An organization $org1$	An organization $org3$ that is sub-organization of the organization $org1$
(m')	An organization $org1$	An organization $org3$ that is sub-organization of the organization $org2$
(n)	An environment $env1$	An environment $env1$
(n')	An environment $env1$	An environment $env2$
(o)	An environment $env1$	An agent $ag1$ that inhabits the environment $env1$
(o')	An environment $env1$	An agent $ag1$ that inhabits the environment $env2$
(p)	An environment $env1$	An organization $org1$ that inhabits the environment $env1$
(p')	An environment $env1$	An organization $org1$ that inhabits the environment $env2$

Table H.2 Possible involved entities relations between two norms

When analyzing the involved entities of two norms we need to observe if their involved entities are the same or if their involved entities are related, i.e. if there are any relations between entities whose behavior are being regulated by the norms.

These relations are defined by the relationships *AgentPlayingRole*, *AgentOfOrganization*, *AgentInhabitsEnvironment*, *SubOrgPlayingRole*, *OrganizationComposition*, *RoleHierarchy*, *RoleOfOrganization* and *OrganizationInhabitEnvironment* of the NormML metamodel (see Figures 3.3 and 3.9 in chapter XXX). If the entities of the norms are not related, i.e., if they apply to entities that are not related to each other, the norms are not in conflict.

The cases (a), (b), (c), (d), (e), (f), (g), (h), (i), (j), (k), (l), (m), (n), (o) and (p) of Table H.2 may result in conflicts because the involved entities of the two norms are related to each other: the cases (a), (b), (h), (i), (j) and (n) are covered by item (i); the cases (d) and (e) are covered by item (ii); the cases (f) and (g) are covered by item (iii); the case (c) is covered by item (iv); the cases (k), (l) and (m) are covered by item (v) and; the cases (o) and (p) are covered by item (vi).

The cases (a'), (b'), (c'), (d'), (e'), (f'), (g'), (h'), (h''), (i'), (i''), (j'), (k'), (l'), (m'), (n'), (o') and (p') cannot result in conflicts because the involved entity of the first norm are not related to the involved entity of the second norm by any way.

Deontic Concept

In 1951, Georg Henrik von Wright published a pioneer plausible system of deontic logic (von Wright, 1951). His work was discussed and refined by various researchers, resulting in the so-called standard deontic logic (Føllesdal and Hilpinen, 1971).

The standard deontic logic has three operators O , P and F that represent respectively the *deontic concepts* of *obligation*, *permission* and *prohibition*. According to the standard deontic logic there is a deontic inconsistency when there is a $O(p)$ and a $F(p)$ or a $P(p)$ and a $F(p)$. Also each $O(p)$ implies in a $P(p)$. Another important point to consider about deontic logic is: if there is a $O(p)$ and a $O(\sim p)$ it is said that they are mutually contradictory obligations, and the same is valid to prohibitions.

With NormML it is possible to describe norms using the three deontic concepts: obligation, permission and prohibition (see Figure 3.3). Considering the conflict cases of the deontic concept analysis based on the standard deontic logic, two norms may be in conflict if: (i) one norm states a permission and another states a

prohibition; (ii) one norm states an obligation and another states a prohibition; and (iii) one norm states a permission and another one states an obligation in the period the permission is not activated; and (iv) both norms state an obligation or both norms state a prohibition to do opposite actions. Thus, let's consider the set of possible deontic concepts relations between two norms.

Possible cases	First norm deontic concept	Second norm deontic concept
(a)	Obligation	Obligation
(b)	Prohibition	Prohibition
(c)	Permission	Permission
(d)	Obligation	Prohibition
(e)	Obligation	Permission
(f)	Prohibition	Permission

Table H.3 Possible deontic concepts relations between two norms

The cases (a), (b), (d), (e) and (f) of Table H.3 may result in conflicts: the case (f) is covered by item (i); the case (d) is covered by item (ii); the case (e) is covered by item (iii); and the cases (a) and (b) are covered by item (iv). The case (c) is the only that cannot result in conflicts because a permission do not affect the influence of another permission.

Action

After the checking of the deontic concept, the next element to be examined is the *action* of the norms. Let's consider the three conflict cases of the *action* analysis. If one of the cases (d), (e) or (f) of the deontic concept analysis is true, it is important to check if the actions being regulated by the norms are of: (i) the same type on the same resource; or (ii) related types on the same or related resources (as defined in the dialect action hierarchy, see Appendix B).

If one of the cases (a) or (b) of the deontic concept analysis is true, so, it is important to check if the actions of the norms are (iii) semantically opposite and restrict the access of the same resource (see Appendix D to the complete list of semantically opposite actions). Actions of that kind are analyzed because they are mutually contradictory when they refer to the same resource.

Thus, let's consider the set of possible actions relations between two norms based on the relations of the actions dialect of NormML.

Possible cases	First norm action	Second norm action
(a)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act1</i> on a resource <i>res1</i>
(b)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act1</i> on a resource <i>res2</i>
(c)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act2</i> related to the action <i>act1</i> on a resource <i>res1</i>
(d)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act2</i> related to the action <i>act1</i> on a resource <i>res2</i> related to the resource <i>res1</i>
(e)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act2</i> related to the action <i>act1</i> on a resource <i>res2</i> not related to the resource <i>res1</i>
(f)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act2</i> not related to the action <i>act1</i> on a resource <i>res1</i>
(g)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act2</i> semantically opposite to the action <i>act1</i> on a resource <i>res1</i>
(h)	An action <i>act1</i> on a resource <i>res1</i>	An action <i>act2</i> semantically opposite to the action <i>act1</i> on a resource <i>res2</i>

Table H.4 Possible actions relations between two norms

The cases (a), (c), (d) and (g) of Table H.4 may result in conflicts because the actions of the norms are related to each other according to the actions dialect of NormML: the case (a) is covered by item (i); the cases (c) and (d) are covered by item (ii); and the case (g) is covered by item (iii). The cases (b), (e), (f) and (h) cannot result in conflicts because the norms actions are not related to each other or because they apply to different resources.

Activation Constraints

Finally, let's reflect on the conflict cases of the *activation constraints* analysis. Two norms may be in conflict: (i) if one norm is not constrained by any period of time; (ii) if the periods established by actions and dates of the invariants Before, After, Between and If intersect; (iii) in case of two If conditions, if the values related to the same attribute or belief intersects (e.g.: $x > 10$ and $x = 15$); and (iv) in case of two If conditions, if the values related to the same goal are equal.

Norms have a period during while they are active, i.e., during while their restrictions must be fulfilled. To describe this activation period of a norm one can define constraints to it. Norms can be activated by one constraint or a set of constraints. NormML has four kinds of norm constraint: *before*, *after*, *between* and *if* (see Figure 3.7). The *before*, *after* and *between* constraints can be associated with actions or dates, and the *if* constraint can be associated with a date, a belief, a goal or an attribute. In Section 3.2.4 it is explained in details how each norm constraint of NormML can be used to describe the activation period of a norm. Thus, let's consider the set of possible activation constraints relations between two norms illustrated in Table H.5.

Possible cases	First norm activation constraint	Second norm activation constraint
(a)	None	None
(b)	None	A before, after, between or if constraint
(c)	A before constraint associated with an action <i>a</i>	A before constraint associated with an action <i>a</i>
(c')	A before constraint associated with an action <i>a</i>	A before constraint associated with an action <i>b</i>
(d)	An after constraint associated with an action <i>a</i>	An after constraint associated with an action <i>a</i>
(d')	An after constraint associated with an action <i>a</i>	An after constraint associated with an action <i>b</i>
(e)	A before constraint	An after constraint
(f)	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>
(f')	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	A between constraint associated with a before action <i>a</i> and an after action <i>c</i>
(f'')	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	A between constraint associated with a before action <i>c</i> and an after action <i>b</i>
(f''')	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	A between constraint associated with a before action <i>c</i> and an after action <i>d</i>
(g)	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	A before constraint associated with an action <i>a</i>
(g')	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	A before constraint associated with an action <i>b</i> or <i>c</i>
(h)	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	An after constraint associated with an action <i>b</i>
(h')	A between constraint associated with a before action <i>a</i> and an after action <i>b</i>	An after constraint associated with an action <i>a</i> or <i>c</i>
(i)	A before, after or between constraint associated with actions	A before, after or between constraint associated with dates
(j)	A before, after or between constraint associated with actions	An if constraint
(k)	A before, after, between or if constraint associated with dates	An if constraint associated with beliefs, attributes, goals or values
(l)	A before constraint associated with a date <i>a</i>	A before constraint associated with a date $b \leq$ date <i>a</i>
(l')	A before constraint associated with a date <i>a</i>	A before constraint associated with a date $b >$ date <i>a</i>
(m)	An after constraint associated with a	An after constraint associated with a date $b \geq$

Possible cases	First norm activation constraint	Second norm activation constraint
	date a	date a
(m')	An after constraint associated with a date a	An after constraint associated with a date $b < \text{date } a$
(n)	A before constraint associated with a date a	An after constraint associated with a date $b < \text{date } a$
(n')	A before constraint associated with a date a	An after constraint associated with a date $b \geq \text{date } a$
(o)	A between constraint associated with a before date a and an after date b	A between constraint associated with a before action c and an after action $d \leq a$
(o')	A between constraint associated with a before date a and an after date b	A between constraint associated with a before action c and an after action $d > a$
(p)	A between constraint associated with a before date a and an after date b	A before constraint associated with a date $c > b$
(p')	A between constraint associated with a before date a and an after date b	A before constraint associated with a date $c \leq b$
(q)	A between constraint associated with a before date a and an after date b	An after constraint associated with a date $c < a$
(q')	A between constraint associated with a before date a and an after date b	An after constraint associated with a date $c \geq a$
(r)	An if constraint associated with a date a	An if constraint associated with a date a
(r')	An if constraint associated with a date a	An if constraint associated with a date b
(s)	An if constraint associated with a date a	A before constraint associated with a date $b > a$
(s')	An if constraint associated with a date a	A before constraint associated with a date $b \leq a$
(t)	An if constraint associated with a date a	An after constraint associated with a date $b < a$
(t')	An if constraint associated with a date a	An after constraint associated with a date $b \geq a$
(u)	An if constraint associated with a date a	A between constraint associated with a before action b and an after action c , where $c < a < b$
(u')	An if constraint associated with a date a	A between constraint associated with a before action b and an after action c , where $c \leq a$ or $a \geq b$
(v)	An if constraint associated with an attribute a with a value v	An if constraint associated with an attribute a with a value v' that intersects v
(v')	An if constraint associated with an attribute a with a value v	An if constraint associated with an attribute a with a value v' that does not intersect v
(v'')	An if constraint associated with an attribute a	An if constraint associated with an attribute b
(x)	An if constraint associated with a belief a with a value v	An if constraint associated with a belief a with a value v' that intersects v
(x')	An if constraint associated with a belief a with a value v	An if constraint associated with a belief a with a value v' that does not intersect v
(x'')	An if constraint associated with a belief a	An if constraint associated with a belief b
(w)	An if constraint associated with a goal a with a value v	An if constraint associated with a goal a with a value $v' = v$
(w')	An if constraint associated with a goal a with a value v	An if constraint associated with a goal a with a value $v' \neq v$
(w'')	An if constraint associated with a goal a	An if constraint associated with a goal b
(y)	An if constraint associated with an attribute	An if constraint associated with a belief
(y')	An if constraint associated with an	An if constraint associated with a goal

Possible cases	First norm activation constraint	Second norm activation constraint
	attribute	
(z)	An if constraint associated with a belief	An if constraint associated with a goal

Table H.5 Possible activation constraints relations between two norms

The activation period of a norm corresponds to the period when the entities of the norm must fulfill the norm. When analyzing the activation constraints of two norms we need to observe if they can be active at the same time. If they cannot be active at the same time they cannot conflict because the entities of the norm will be able to fulfill the two norms separately.

The cases (a), (b), (c), (d), (f), (f'), (f''), (g), (h), (e), (m), (n), (o), (p), (q), (r), (s), (t), (u), (v), (x) and (w) of Table H.5 will result in conflicts because the two norms will be active at the same time interval: the cases (a) and (b) are covered by item (i); the cases (c), (d), (f), (f'), (f''), (g), (h), (e), (m), (n), (o), (p), (q), (r), (s), (t) and (u) are covered by item (ii); the cases (v) and (x) are covered by item (iii); and the case (w) is covered by item (iv).

The cases (l'), (m'), (n'), (o'), (p'), (q'), (r'), (s'), (t'), (u'), (v'), (x') and (w') cannot result in conflicts because the two norms will never be active at the same time. Our approach assumes that the cases (c'), (d'), (e), (f'''), (g'), (h'), (i), (j), (k), (v''), (x''), (w''), (y'), (y'') and (z) are also not in conflicts because these cases cannot be analyzed at design time.