

In the Name of God

**The Implementation of
Dynamic Assignment of
Rights, Responsibilities and Sanctions
to External Agents in Normative Multiagent Systems**

Thesis submitted in accordance with the requirements of the University of
Liverpool for the degree of Doctor in Philosophy

By

FARNAZ DERAKHSHAN

October 2008

Abstract

The Implementation of Dynamic Assignment of Rights, Responsibilities and Sanctions to External Agents in Normative Multiagent Systems

Farnaz Derakhshan

Recently, the design and development of multiagent systems (MASs) has become increasingly concerned with the recognition that they will be used in a dynamic and open environment. In such environments, it is a very difficult and complicated task to anticipate all possible runtime situations at design time. Therefore, in order to respond to changes in this environment it is necessary to allow the system to provide dynamic responses at runtime. This thesis is concerned with one particular aspect of such responses. Our novel contribution is that we explicitly identify, clarify and address the problem of dynamic assignment of rights, responsibilities (R&Rs) and sanctions to external agents in normative MASs.

The background setting of this work deals with the topic of dynamism in normative MASs and attempts to address and combine some issues regarding dynamic resources in MASs and different types of norms in legal systems consisting of various types of legal modalities, including obligation, prohibition, permission and right; enforcement modalities, including punishment, reward and compensation; and all key elements of norms such as addressee, beneficiary, temporal notions, and preconditions.

Following this introduction, we propose two alternative methods for dynamic assignment of R&Rs and sanctions to external agents, and propose a formalism to represent a commonsense understanding of our solution. The first method is based on role hierarchies in MASs and the second method is based on conditional norms. Both methods have common features including reliance on the concept of role, using a normative Knowledge Base (KB) and sensitivity to runtime occurrences affecting the MAS. The significant differences of these two mechanisms lie in the definition of roles and normative KBs.

Furthermore, we consider aspects of implementation based on common features of the proposed methods, which we follow with a general implementation architecture for dynamic assignment of R&Rs and sanctions to external agents. Using this general architecture and guidelines, we present an agent-based auction application to demonstrate the practical feasibility of our approach and of our architecture.

The implementation of an agent auction that we present allows us to examine and compare the functionality of our two methods under various scenarios, including different runtime occurrences and various types of legal notions.

CONTENTS

ACKNOWLEDGEMENTS.....	10
CHAPTER 1.....	11
INTRODUCTION	11
1.1 AIMS AND OBJECTIVES	12
1.2 ACHIEVEMENTS AND CONTRIBUTION	13
1.3 THE STRUCTURE OF THE THESIS	15
CHAPTER 2.....	20
BACKGROUND.....	20
2.1 INTRODUCTION	20
2.2 AGENTS AND MULTIAGENT SYSTEMS	21
2.2.1 Definition of an Agent	21
2.2.2 Definition of a Multi Agent System	25
2.3 AGENTS AND ROLES	27
2.3.1 The Key Role of Roles.....	29
2.4 NORMATIVE MULTIAGENT SYSTEMS	31
2.5 SUMMARY	34
CHAPTER 3.....	36
RIGHTS AND RESPONSIBILITIES.....	36
3.1 INTRODUCTION	36
3.2 CLASSIFICATIONS FOR NORM TYPES	38
3.3 REGULATIVE NORMS.....	42

3.3.1	The key elements of regulative norms.....	43
3.3.2	The Key Issues on Legal Modality.....	45
3.3.2.1	Definitions of Obligation, Permission and Prohibition.....	46
3.3.2.2	Different Meanings.....	48
3.3.2.3	Minimizing Legal Modalities.....	50
3.3.2.4	Examples of Minimizing Legal Modalities.....	52
3.3.2.5	Legal Modality in Our Work.....	53
3.4	NORM ENFORCEMENT.....	54
3.4.1	The Necessity of Enforcing Norms.....	54
3.4.2	Check Norms.....	56
3.4.3	Reaction Norms.....	56
3.4.4	Examples of Enforcement Norms.....	60
3.4.5	Enforcement-Norm Elements.....	61
3.4.6	Norm Enforcement versus Norm Regimentation.....	62
3.4.7	Enforcing the Enforcement Norms.....	63
3.5	FORMALIZATION OF THE NORMS.....	64
3.6	SUMMARY.....	69
CHAPTER 4 DYNAMIC ISSUES IN NORMATIVE MAS.....		70
4.1	INTRODUCTION.....	70
4.2	DYNAMIC ISSUES IN MAS.....	71
4.2.1	The Source of Dynamism.....	71
4.2.2	The Source of Changes.....	72
4.3	DYNAMIC ASSIGNMENTS.....	75
4.3.1	Dynamic Assignment of Roles to Agents.....	76
4.3.2	Dynamic Assignment of R&Rs to Agents in Normative MAS.....	77

4.3.2.1	Tri-level Structure for MAS.....	80
4.3.3	Dynamic Assignment of Sanctions to Agents.....	84
4.4	PROTOCOL-BASED VERSUS RULE-BASED NORMS.....	87
4.5	SUMMARY	89
CHAPTER 5		92
METHODS FOR DYNAMIC ASSIGNMENT		92
OF R&RS TO EXTERNAL AGENTS.....		92
5.1	INTRODUCTION	92
5.2	METHODS	93
5.2.1	Method 1- Using Role Hierarchies.....	96
5.2.2	Method 2: Using Conditional Norms	101
5.3	COMMON FEATURES OF METHODS	103
5.4	THE DIFFERENCES OF THE TWO METHODS.....	105
5.5	FORMAL REPRESENTATION	106
5.5.1	Agents.....	107
5.5.2	Roles	107
5.5.3	Actions of Agents.....	109
5.5.4	Environmental Events.....	110
5.5.5	Runtime Occurrences.....	112
5.5.6	Formal Syntax of Norms and Enforcement Norms.....	113
5.5.7	Formal Syntax of Normative Commands.....	116
5.5.8	Formal Syntax of Conditions of Norms	118
5.5.9	Normative Rules.....	122
5.5.10	Normative Knowledge Base	123

5.5.11	The Set of Current Activated R&Rs of agents	124
5.5.12	The Instantaneous Rights/ Responsibilities of Agent.....	125
5.5.13	The Function of Assignment of R&Rs and Sanctions to Agents.....	127
5.6	EXAMPLES OF R&R ASSIGNMENT	129
5.6.1	An Example for Agent set (Ag).....	129
5.6.2	An Example for the function of Current Role (CR).....	130
5.6.3	An Example for Normative Knowledge Base (NKB)	130
5.6.1	An Example for the Set of existing R&Rs (<i>S</i>).....	132
5.6.2	Examples of Runtime Occurrences (RO).....	132
5.6.2.1	Occurrence 1 (New agent joins)	134
5.6.2.2	Occurrence 2 (An Action).....	135
5.6.2.3	Occurrence 3 (An Event)	135
5.6.2.4	Occurrence 4 (An Env. Parameter changes).....	136
5.6.2.5	Occurrence 5 (A Deadline)	137
5.7	SUMMARY	138
CHAPTER 6.....		139
IMPLEMENTATION ISSUES.....		139
6.1	INTRODUCTION	139
6.2	SIMILARITIES VS. DIFFERENCES IN IMPLEMENTATION OF METHODS	140
6.3	ROLE DEFINITION	141
6.3.1	Role Definition in Method 1	142
6.3.2	Role Definition in Method 2	143
6.4	DESIGNING THE NORMATIVE KNOWLEDGE BASE	144
6.4.1	The Type of the Normative KB	145
6.4.2	The Descriptive Normative Language of KB.....	146

6.4.3	Creating the Normative Knowledge Base	147
6.5	GENERAL ARCHITECTURE (BASED ON COMMON FEATURES).....	148
6.5.1	Using Jess	149
6.5.2	Diagram	150
6.5.2.1	Entities	153
6.5.2.2	Communication Processes	158
6.6	THE DESCRIPTION OF THE PROCESS.....	161
6.6.1	How Dynamic Assignment Occurs	161
6.6.2	When a dynamic assignment occurs.....	164
6.6.3	Who assigns dynamic assignments.....	166
6.7	USING METHOD 1 OR METHOD 2	166
6.8	SUMMARY	168
CHAPTER 7		170
THE DESIGN OF A MIDDLEWARE TOOL		170
7.1	INTRODUCTION	170
7.2	ANALYSIS.....	171
7.2.1	The Functionality of the Tool	172
7.2.2	Inputs and Outputs of the Tool	173
7.3	DESIGN	175
7.3.1	Design of Tool Entities	175
7.3.1.1	Use case diagram.....	175
7.3.1.2	Activity Diagram.....	177
7.3.1.3	Class Diagram	179
7.3.2	Design of Normative Knowledge Base	182
7.3.2.1	Creating the Normative Knowledge Base	183
7.3.2.2	Domain-Related Rules	187

7.3.2.3	General Rules	192
7.3.3	Design of the Simulator of Event /Action Handler	195
7.3.4	Design of Timer	196
7.3.4.1	Time Management in R&R Allocator Tool	196
7.3.5	Design of User Interface	197
7.4	SUMMARY	197
CHAPTER 8.....		199
IMPLEMENTATION OF THE MIDDLEWARE TOOL.....		199
8.1	INTRODUCTION	199
8.2	IMPLEMENTATION.....	200
8.3	TESTING.....	201
8.3.1	Scenario	202
8.3.2	Method 1.....	204
8.3.3	Method 2.....	227
8.4	EVALUATION AND COMPARISON	246
8.5	SUMMARY	248
CHAPTER 9.....		249
DISCUSSION.....		249
9.1	INTRODUCTION	249
9.2	COMPARISON WITH RELATED WORK.....	250
9.2.1	Electronic Institutions	250
9.2.1.1	Definition of EIs.....	251
9.2.1.2	Tools for EIs.....	252
9.2.1.3	Applications Using EI.....	254
9.2.1.4	Evaluation of EI	256

9.2.1.5	Comparison with Our Work	258
9.2.2	Systems and Dynamic Assignment of Roles to Agents	259
9.2.2.1	Comparison with Our Work	261
9.2.3	Normative Framework for MASs	262
9.2.3.1	Comparison with Our Work	263
9.2.4	Beneficiaries, Rights and Compensation	263
9.3	SUMMARY	264
9.4	OUR CONTRIBUTIONS	267
9.4.1	Our Assumptions and Limitations	269
9.5	FUTURE WORK.....	272
BIBLIOGRAPHY		280
GLOSSARY		288
CHAPTER 10.....		297
APPENDICES.....		297
10.1	APPENDIX A: THE TABLE OF EVALUATION OF ROLE PROPERTIES.....	298
10.2	APPENDIX B: SILVA'S GRAMMAR.....	299
10.3	APPENDIX C: THE FUNCTIONS IN OUR ARCHITECTURE.....	300
10.4	APPENDIX D: MESSAGE SEQUENCE CHART	301
10.5	APPENDIX E: AUCTION NORMATIVE KB FOR METHOD 1	303
10.6	APPENDIX F: AUCTION NORMATIVE KB FOR METHOD 2	314
10.7	APPENDIX G: GENERAL RULES	322
10.8	APPENDIX H: TABLES FOR EXTRACTING GENERAL RULES	329
10.9	APPENDIX I: THE JAVA SOURCE CODE OF APPLICATION.....	338

Acknowledgements

Thanks to God, my creator, who helped me in the creation of this dissertation.

I would like to thank all people and organizations who helped me during my PhD time, I am most grateful for their assistance. Firstly, I would like to thank my sponsors, Iran's ministry of Science, Research and Technology, the University of Tabriz and Department of Computer Science at the Liverpool University for their financial support.

Secondly, I would like to thank my supervisors Dr Peter McBurney and Professor Trevor Bench-Capon for their supervision, valuable guidance and support. It has been a pleasure and an honour for me to have been supervised by these great supervisors.

I would like to thank my PhD examiners, Professor Mark d' Inverno and Dr. Katie Atkinson, for their constructive comments and attention to my work.

I express my heartfelt thanks to my spouse Dr. Maziar Asefi for his unfailing support and encouragement without whom I doubt whether I would have ever completed my research.

I also would like to thank my parents, Mr. Abolghasem Derakhshan and Mrs Fatemeh Khorasani and also my sisters Fariba and Zahra, and my brother, Hamid, for their love and support.

Last but no means least, I would like to thank all my friends and colleagues in the UK and in Iran, in particular, Mrs Azam Nemati, who was very kind and helpful in the demanding official process of my scholarship.

Chapter 1

Introduction

The title of this thesis is “*The Implementation of Dynamic Assignment of Rights, Responsibilities and Sanctions to External Agents in Normative Multiagent Systems*”. Basically this title specifies the area of research, which is normative multiagent systems (MAS). The title also shows that our research talks about dynamism, norms, the assignment of norms to agents and the implementation of such assignments.

Here an introduction to this thesis is presented. We briefly describe the context of this thesis, followed by the aims and objectives of this research. After that we

briefly summarize our achievements. Finally, we present an outline of the various chapters of this thesis.

1.1 Aims and Objectives

The context of this research is the design and development of normative multiagent systems (considering also that these will be *open* systems). In essence, we were trying to find a way to assign rights, responsibilities and sanctions to agents in these MASs at run-time. These systems are dynamic and external agents have autonomy to follow or violate the rules at runtime. The ability to assign rights and responsibilities and sanctions to external agents dynamically in such systems is important for several reasons:

First, currently methodologies for agent-oriented software design have assumed that roles, rights and responsibilities are assigned to agents at design-time, rather than at run-time. However, in dynamic environments it is a very difficult and complicated task to anticipate all possible runtime situations at design time, before runtime. Therefore, in order to respond to changes in the environment it is necessary to allow the system to provide the assignment of R&Rs to external agents dynamically at runtime.

Second, the ability of agents in the MAS to identify and punish undesirable behaviors themselves at run-time reduces the need for system designers to identify and exclude all such behaviors at design-time. Furthermore, identification and punishment of undesirable behaviors may be undertaken immediately when the behaviors happen.

From consideration of the importance of the topic of dynamic assignment of R&Rs and sanctions to external agents, a number of questions arose which we address in this PhD thesis. These research questions can be listed as follows:

- What is the concept of dynamic assignment of Rights and Responsibilities (R&Rs) and of sanctions to agents?
- Which factors may cause the corresponding R&Rs and sanctions of an agent to change at runtime?
- What methods can be proposed to undertake such assignments?
- How can the proposed methods be implemented?
- What is a practical example of such implementation?
- How can this work be extended in the future?

1.2 Achievements and Contribution

In this section, we describe what we have achieved while undertaking this research, and how we have addressed the aims and objectives and have answered our questions.

In this thesis, we start by presenting our new proposal for the dynamic assignment of R&Rs and sanctions to external agents in normative MAS. Then, in order to find a way to assign rights, responsibilities and sanctions in normative MAS to agents

dynamically, we specify two main aspects: first, the features of R&Rs and sanctions of roles in normative MAS which need to be stored in a normative knowledge base (KB); and second, the sources of dynamism in MAS which may lead to changes in R&Rs and to sanction assignments.

Next, we propose two novel methods for such assignments, along with a formal representation and examples of using the formalism for each method. Subsequently, we identify the key implementation issues and the general architecture for applying our proposed methods in real multi-agent systems. After that, we develop a practical middleware tool for providing dynamic assignment of R&Rs and sanctions to external agents using our two proposed methods, followed by an explanation of the development process for our middleware tool. We explain the development levels of our tool from analysis and design stages through to implementation. In addition, we test the functionality of this tool using an auction example, which we also present in this thesis. The design and implementation of the middleware tool, and its application to a realistic example (an auction system) provide us with a software prototype which can be used to evaluate our proposed approaches. In other words, the practical viability of our two proposed methods, and their respective strengths and weaknesses is assessed through the deployment in the prototype system we develop in this research. Moreover, the tool is not application-specific, but is generic, and so may be incorporated into any normative multi-agent system.

We also discuss related work and possible future work in this area of research.

Here we stress that one of our main research contributions is that we explicitly identify and address the problem of dynamic assignment of R&Rs and sanctions to

agents in MASs for the first time. In other words, the contribution of this PhD thesis is not only that we present a solution to the problem, but that we have identified, clarified and addressed the problem. This problem is one which has not been addressed explicitly before in the MAS literature, although some MAS methodologies do permit dynamic assignments of roles.

Identifying, clarifying and addressing the problem required us to consider the research literature on norms, obligations, permissions, etc, and to consider methodologies for agent-oriented software engineering; the relevant literature is considered in the early chapters of this thesis. Then we present our solution to the problem in the remaining chapters.

1.3 The Structure of the Thesis

Here we present a general overview of this work, along with a description of the main features and the structure of this thesis. We provide the title of chapters, the topic of their sections and a brief description of the contents of each chapter. The context of this thesis is chronologically structured as follows:

In Chapter 2, “*Background*”, we provide an introduction and background to the area of normative multiagent systems. We explain the basic ideas of this domain in the following sections: Basic Definitions (including Agents, Roles and Rights and Responsibilities), Multiagent Systems and Normative Multiagent Systems. Some of the relevant literature on agent-oriented software engineering is best considered in the light of our research proposals and our prototype application; accordingly, we postpone some of this literature discussion to Chapter 9.

In Chapter 3, “*Rights and Responsibilities*”, we define the concepts of rights, responsibilities and sanctions considering the normative viewpoint, in the way these concepts will be used throughout this thesis. Although we mostly have adopted definitions from the relevant literature, we have found it necessary to emphasize the concepts of beneficiary, rights and compensation (which can be claimed when a particular norm is not fulfilled); this emphasis is one of our contributions to this subject. In summary, this chapter covers the following topics: Classification for Norm Types, Regulative Norms, Norm Enforcement, and Formalization of the Norms.

In Chapter 4, “*Dynamic Issues in Normative MAS*”, we explain dynamic issues in normative multiagent systems. Chapter 4 includes the following sections: Dynamic Issues in MAS, Dynamic Assignments, and Protocol-based versus Rule-based Norms. In this chapter, first, we discuss the main sources of dynamic environments in MAS. Then we explain three types of dynamic assignments as responses to the dynamic environment of normative MAS. These responses include: dynamic assignment of roles to agents; dynamic assignment of R&Rs to agents; and dynamic assignment of sanctions to agents. The second and third of these responses are novel proposals first advanced in this thesis. In the last section of this chapter, we present a complementary discussion about protocol-based norms versus rule-based norms which will be used in the following chapter.

As a result, in Chapters 2, 3 and 4, we explain the idea of dynamic assignments of R&R and sanctions from a basic level and we explain how dynamic sources in MAS may frequently lead to changes in the R&R and sanctions of external agents. After that, in the following chapters of this thesis, we propose a formalism of our

methods, an example implementation, and a middleware tool for dynamic assignment of R&Rs and sanctions to external agents.

Accordingly in Chapter 5, “*Methods for Dynamic Assignment of R&Rs to External agents*”, we propose two methods for dynamic assignment of R&Rs and sanctions to external agents. This chapter includes the following sections: Methods, Common Features of Methods, Differences of Methods, Formal representations, and Samples of R&R Assignment.

For more clarification, here we present further explanations of our methods. Both proposed methods are rule-based (not protocol-based). The first method is based on *hierarchical roles* which can be applied in those MASs which have a hierarchical structure. The second method is based on *conditional norms*. In this method, there are not large numbers of distinct roles in the system, so changing the roles of agents does not happen frequently and thus each agent is assigned a role which is fairly stable. So dynamic sources provide the pre-conditions that should exist for assignment of the related rights or/and responsibilities to an agent.

Here, we also emphasize that our methods provide dynamic assignment of sanctions as well as of rights and responsibilities. This means that these methods are able to detect norm violations (or enactments) immediately after occurrence and enforce the punishment (or reward) defined by the system legislator in the rule base. After presenting our methods, a formal representation is defined for dynamic assignment of R&Rs to agents, followed by examples of using the formal representation in Chapter 5.

In Chapter 6, “*Implementation Issues*”, we discuss aspects of implementation for the dynamic assignment of R&Rs and sanctions to agents followed by a general architecture for both methods. In this chapter, we mention the different and common aspects for implementation of our two proposed methods. Then, we conclude that the most significant difference is in the definition of the normative knowledge base; this means that rules are defined differently in Method 1 and Method 2. But the sources of system dynamism are assumed to be the same for both methods.

In Chapter 7, “*The Design of a Middleware Tool*”, there are two main sections: Analysis and Design. As we developed a practical middleware tool using our proposed methods, we describe the analysis and design stages of the software development of this tool.

One of the important design tasks for this tool was the design of the normative knowledge base (KB). The contents of the norms are usually different from one domain application to another. In addition, as already mentioned, the KBs for Method 1 and Method 2 are designed differently. However, we present guidelines for a designer seeking to create the KB in each method.

Furthermore, this tool contains a built-in normative KB (rule base) consisting of general rules which will be joined to the designer’s rule base during the runtime. These general rules provide the necessary definitions and templates which are common for the KB of both methods. Defining such built-in rules in the system assists a system designer to define the norms in a normative MAS.

In Chapter 8, “*Implementation of the middleware tool*”, we explain the implementation of our tools based on the proposed analysis and design in the previous chapter. Then we present our implementation of this tool with some snapshots and the relevant descriptions. We demonstrate that the application works by means of a full example of an auction system. We test our application by developing dynamic assignment using both of our proposed methods. The tool created and described in Chapter 8 is a software prototype which demonstrates the practical viability of the novel methods for dynamic assignments we proposed in Chapter 5.

Our developed tool can be connected to any normative MAS to provide dynamic assignment of R&Rs and sanctions to external agents in the system. In other words, our tool has not been designed and developed for a specific application domain, but is generic. As a result, users can simply design their own rule base according to the proposed template for designing rule bases.

Finally, in chapter 9, “*Discussion*”, we include a discussion of related work to evaluate and highlight the advantages of our work relative to the existing literature. As mentioned above, we discuss some of the relevant agent-oriented software engineering literature here, rather than in Chapter 2, because the systems considered are best understood in comparison with our own proposals. We end this chapter and the thesis with a discussion of possible future research in this area.

We also include a number of appendices at the end of this dissertation including tables, charts and normative knowledge bases.

Chapter 2

Background

2.1 Introduction

This chapter provides an introduction and background to the research in this thesis. As the title of thesis shows, the area of this research is based on multiagent systems. For this reason, here a brief description of the basic concepts in this area is provided.

Firstly, we give a general description of the concepts of agents, roles, rights and responsibilities (R&Rs), and also explain the relationships of these concepts. The aim is to show how the concepts of roles, of rights and of responsibilities (derived from a model of organizations) are used in multiagent systems.

Secondly, as this work concentrates on a specific type of multiagent system called a “*Normative Multiagent System*”, the definitions of normative multiagent system will also be presented.

Also relevant to our research is the research literature on agent-oriented software engineering (AOSE), and the various methodologies and frameworks which have been proposed for this engineering activity. We have decided to include a discussion of this literature in Chapter 9, rather than in Chapter 2, in order that we can undertake a detailed comparison of each major AOSE methodology and framework with our own proposed methods. However, our own methods are only presented and explained in the main chapters of this thesis, so the detailed comparison needs to wait till the final chapter where our methods are assessed.

2.2 Agents and Multiagent Systems

In this section, the basic definition of agents and their main characteristics will be described, followed by the definition and discussion of important issues in multiagent systems.

2.2.1 Definition of an Agent

For some decades, the concept of object - as a software entity characterized with attributes and behaviors – along with object-oriented programming, has offered a valuable abstraction for modeling and designing computer systems and implementing computer software. Recently, however, the concept of agents has achieved some success, because this concept provides a still higher level of abstraction.

The definition of agent we use in this thesis is that of Wooldridge [82]:

*“An **agent** is a computer system that is situated in some environment, and that is capable of autonomous action in the environment in order to meet its design objectives.”*(page 15)

This definition clearly shows the main features of the agent. One of these features, “*situated in some environment*”, means that an agent has relationships with its environment, and can interact with and affect the environment. The other fundamental feature of agents is that they are “*capable of autonomous action*”, which indicates their ability to decide for themselves, to undertake flexible and effective operation in the environment to reach to their “*design objectives*”.

As an example of a very simple agent, Wooldridge [82] presented a control system such as thermostat with a sensor for detecting the temperature of a room. This sensor is situated within the room (as its environment) and produces one of the following two output actions to respond to the environment’s temperature: *Temperature is too cold* → *heating on* and *Temperature is OK* → *heating off*.

The classification of Agents: In addition to the above definition which provides the basic description for understanding the notion of agent, there are other definitions which characterize additional properties of agents. In [51], two distinct views of agents were presented: the **weak or intelligent notion** of agents and the **strong or intentional notion** of agents.

Intelligent agents, besides autonomy (mentioned in the basic definition of agent), are characterized with three additional properties [82]:

- **Reactiveness:** Intelligent agents can monitor their environment, and effectively respond to changes that occur in it, in order to satisfy their design objectives.
- **Proactiveness:** Intelligent agents can direct their behavior towards achievement of their goals over longer periods of time in order to satisfy their design objectives.
- **Social ability:** Intelligent agents, operating in dynamic and open environments, have the ability to interact and communicate with many other agents in order to satisfy their design objectives.

Although the weak or intelligent notion of agents generally has been accepted as the key feature of all agents, alternative characterizations with additional properties have also been provided. These include: learning ability, mobility, rationality and many other possible features of agents [51] which are considered as features of a strong notion of agency.

These strong or intentional notions of agents are often based on control architectures, comprising mental components such as beliefs, desires and motivations [51]. Moreover, they are typically characterized as approximate positions along certain dimensions, rather than being defined precisely. In the other words, more specific labels for agents - which describe further characteristics - are related to a particular application domain or capability of an agent. For instance, the typical agent for use in computer game applications may differ from that in electronic commerce applications.

Events and Actions: Among additional key concepts necessary for developing agent systems, one can mention *event* and *action*. An event is a significant

occurrence or change in the agent's environment or internally within the agent itself, to which the agent should respond by some means. As mentioned before, agents are reactive, and so events are very important when designing agent systems since they identify important changes to which an agent needs to react.

An action is what an agent does, which is the ability of the agent to affect its environment. Abstractly, an agent receives information about *events* and their effects from its environment, then *it somehow selects an action* to perform and finally *it performs the action*. By “*somehow*”, we can understand that an agent can autonomously choose a plan, since a proactive agent has defined goals and a collection of plans to realize these goals.

There are different types of actions, for instance, dialogical actions and non-dialogical actions which we briefly mention here.

Dialogical versus Non-Dialogical Actions: Dialogical actions are those actions which provide the exchange of messages between agents while non-dialogical actions are not related to other interactions between agents. Non-dialogical actions are related to tasks executed by agents, for example, their access to resources or their commitment to the performing of roles [71].

As an example, in an electronic auction, the auctioneer's notification for ending the auction is a dialogical action, because agents such as a buyer and a seller receive a message informing them about the new status of the auction. Some buyers activities such as logging into the system, or placing a bid are non-dialogical actions, since the execution of these actions is independent from sending and receiving messages between agents (here between buyer agent and other agents).

However, some actions can be defined as either dialogical or a non-dialogical, depending on the method of modeling the problem [71].

2.2.2 Definition of a Multi Agent System

After defining the basic concept of agent, we now define the concept of multiagent systems. Then the main concepts and concerns in multiagent systems area are discussed.

Referring to Wooldridge [82], a multiagent system (MAS) is defined as follows:

“A multiagent system is one that consists of a number of agents, which interact with one another, typically by exchanging messages through some computer network infrastructure.”(Page 3)

Interaction of Agents: In a typical definition of a multi-agent system, the definition of possible agent interactions is a basic feature. To facilitate successful interaction, similar to the human relationships, agents usually require the ability to cooperate, coordinate and negotiate with each other. An important issue related to cooperation is **reaching agreements** [82] in a society of autonomous agents which can be performed by **negotiation** and **argumentation**.

Negotiation can be governed by a particular mechanism called a protocol. The protocol defines the rules of interactions between agents [82]. Different types of protocols have been defined, for instance, game-theoretic protocols [39].

In **argumentation**, agents exchange messages with some goal or goals in mind. For example, one agent may try to convince another agent of the truth of some proposition [82] or the agents may be seeking to agree on a course of actions [6].

Communication is another important topic in agent computer systems. While it is usual to use communication through shared data structures, communication is treated differently in the agent community, as agents can neither force other agents to perform some action, nor to communicate, as these other agents are themselves autonomous. In fact, they perform communicative actions to attempt to influence other agents appropriately ([82], page 164). A number of communication languages influenced by the well-known speech act theory [69] have been developed specifically for agent communication such as KQML [43] and FIPA Agent Communications Language (ACL)[25].

Distributed systems versus MAS: Another issue we mention here is the main distinction between traditional distributed systems and multiagent systems; referring to [82], the differences can be summarized as follows:

First, in multiagent system, individual agents may have different goals, so they may not share any common goals. Thus, agents must act strategically in order to achieve the result they most prefer.

Second, agents are assumed to be acting autonomously; therefore, they have to make decisions at *runtime* rather than having all decisions made for them at design time. Consequently, agents must be capable to coordinate their activities *dynamically* and cooperate with others.

In comparison, coordination and cooperation in traditional distributed or concurrent systems are typically hardwired at design time so that the software components will achieve their assigned tasks.

Open multiagent systems: While traditionally agent based systems dealt with well-behaved entities in reliable infrastructures and simple domains, open systems are characterized by unknown components which can change over time, and which may be self-interested human and software agents developed by diverse parties [41]. Therefore, open multi-agent systems are considered as distributed systems comprising (possibly) large and varying populations of agents with different behaviors engaged in competitive or co-operative interactions (agent cooperation is not fixed at design time but may emerge at run time).

2.3 Agents and Roles

In multiagent systems, the concept of role is frequently used and, indeed, roles are a very important idea in MAS. Therefore, in this section, the notion of role and its main relevant concepts are discussed.

Basically, the concept of roles in multiagent systems originates from real roles in human organizations. In fact, using the metaphor of organizations is a very fundamental and useful pattern for developing multiagent systems.

In a human organization, there are some predefined and specified roles which throughout the organization's lifetime different individuals might occupy.

For instance, a supermarket has roles such as *store manager*, *sales manager* and *sales assistant*, and these roles are instantiated with different actual individuals at different times. During the supermarket's existence there should always be an individual who takes each role. The role assigned to an individual may be changed after a while, such as when a person receives a promotion from *sales assistant* to *sales supervisor*. Furthermore, different individuals may be assigned to the same role (in a supermarket, there are usually several *sales assistants*) and also different roles can be assigned to one individual (a person may be both a *sales assistant* and a *customer service provider*). So there is not any insistence on one to one mapping between individuals (agents) and roles.

Rights and Responsibilities of Roles: Each role has its own set of rights and responsibilities, with two main features:

First, the set of rights and responsibilities assigned to each role is ***independent of those assigned to the other roles***; while roles have interrelations and contribute towards the collective objectives of the multiagent system. For example, in supermarket, the role of *store manager* would usually have a set of rights and responsibilities which are different from those of a *sale manager*.

Second, the rights and responsibilities of roles are usually ***predefined*** in multiagent systems and are obviously ***independent of the agent or individual who plays the role***. For instance, *sale assistant* is a role in the supermarket, which has several rights and responsibilities. These rights and responsibilities are the same if *Mari* is the agent who plays the role of *sales assistant* or if *Sarah* plays that role.

Internal and External Agents (Roles): Multiagent systems are often composed of two types of agents (roles); ***internal*** and ***external agents (roles)*** [11]. Internal

agents work on behalf of the MAS, whereas external agents are agents that join the MAS to use its facilities. The internal roles can only be played by internal (or staff) agents on behalf of the MAS. The *external roles* are played by external agents that want to join the MAS.

2.3.1 The Key Role of Roles

The rapid growth of the development of agent technologies and methodologies, on the one hand, and the complexity of the multiagent systems environments, on the other hand, has influenced research for tackling environmental matters of such systems. Patsakoulasic and Vouros, the authors of [62, 64], have assessed the environmental issues of multiagent systems. In those papers, they explained the importance of roles for the reduction of complexity, especially in dynamic and unpredictable environments with a high degree of interaction and distribution.

In [62, 64], environmental issues of multiagent systems are said to be categorized along three dimensions: first, the degree of interaction; second, the dynamics of the environment; and third, the degree of distribution. We summarize each of these aspects as follows:

High degree of interaction: Interaction results from the need for agents to resolve issues of limited or shared resources, interdependencies of agent tasks, and goals or tasks shared by a group of agents that require collective effort.

For reducing the degree of interaction between agents there are a number of responses including: imposing a specific organizational structure on team members; specifying which resources each agent can use; what information should

be communicated between agents; and what goals should be achieved by each agent.

Comparing these responses for reduction of the degree of interaction and role's features, roles provide an appropriate level of abstraction for the specification of these issues. In this way, coordination can be simplified and the degree of interaction can be reduced, or be efficiently controlled.

Environment dynamics: Agents need to plan effectively for achieving their shared goals. Agents' actions or environmental changes provide dynamic and unpredictable changing environments for agents while each agent's prediction and monitoring capabilities are usually limited. Therefore, to plan and act in such a dynamic environment, agents must reason about their intended activities in an abstract way. Such an abstraction can be provided by roles.

Roles aggregate intentions of agents, and specify necessary conditions for executing particular actions, and capture dependencies among the intended behaviors in a shared environment.

Distributivity: The accessible resources of agents are distributed among them, which may include knowledge about tasks to be performed, information about the environment and other agents, and other task-specific resources that are inherently distributed to subsets of agents.

As such distributivity complicates managing of agent's tasks and environment, it may be necessary for agents to form groups with shared objectives, cooperate to

have an integrated view of both the entire task-environment and the mental states of their collaborators.

Roles provide abstract specifications of distributed behavioral patterns and also can accomplish a specific objective. Precise specification of roles, of their relations and interdependencies enable agents to:

- a. reduce the amount of interaction necessary for effective group behavior by applying an organization structure
- b. cope with the dynamics of the task-environment by organizing the group and revising the existing organization structure based on its needs
- c. manage the distributivity of the task-environment by making decision on the assignment of the roles to agents

According to the above discussions, Partsakoulasic and Vouros, stated in [62, 64] five most important properties for roles including *Explicit Specification*, *Dynamic Assignment to Agents*, *Dynamics*, *Cardinality* and *Lifespan*. Then, considering these properties, they evaluated some agent-based methodologies and multiagent systems presenting the results of their evaluation in a table, which is given in Appendix A.

2.4 Normative Multiagent systems

Having introduced the concepts of agents, roles, and multiagent systems, we now introduce the concept of a normative multiagent system, since it is the main focus

of this research. A normative multiagent system is a kind of multiagent system combined with a normative system.

The idea of normative multiagent system is based on considering multiagent systems as a social society in which norms are essential and which also influence agent behaviors. While various works now address multiagent organization or regulated systems, Boella and his colleagues, in a series of work [7-9], specifically consider normative multiagent systems.

In [7, 9], the need for defining normative multiagent systems is discussed from two aspects. On the one hand, there are several social viewpoints on multiagent systems from the basic agent concepts such as coordination, organization and communication to an artificial model of human societies. On the other hand, in comparison with the use of norms as a key issue in human social systems, it seems norms may be necessary too for artificial agents in multiagent systems that collaborate with humans, or display human-like behaviors.

Using notions from human social theory in multiagent systems is now well established and even appears in the basic foundations of agent theory, in particular, the definitions of agent and intelligent agent. For instance, as mentioned earlier, an intelligent agent has three main characteristics [82]: first, *reactivity* as interaction with environment, second, *pro-activity* as taking goal-directed behavior, and third, *social ability* as the interaction with other agents and cooperation. Therefore, it is clear that these fundamental notions of agents are based on the elements of social science theories.

Since norms play a very important role in many social phenomena such as coordination, cooperation, there is an increasing interest in using norms in multiagent systems as well. Obviously, use of norms in multiagent systems initially requires the analysis of the role of norms in social systems. In [7, 9], the authors provide an overview of norm definition, classification, and the role of norms for social theory by citing several relevant publications. For example, they mention Therborn's work [75] for the classification of norms and offer several definitions for normative systems, which will be mentioned in Chapter 3.

Although there are several works and definitions for normative multiagent systems, we mention just two of them here.

First, Boella and Torre - the authors of [10] - have summarized the norm discussion of normative multiagent systems as follows:

“Normative multiagent systems study general and domain independent properties of norms. It builds on achievements in deontic logic, the logic of obligations and permissions, for the representation of norms as rules, the application of such rules, contrary-to-duty reasoning and the relation to permissions. However, it goes beyond logical relations among obligations and permissions by explaining the relation among social norms and obligations, relating regulative norms to constitutive norms, explaining the evolution of normative systems, and much more.” (page 8)

Second, according to Boella [7, 9], normative multiagent systems combine theories and frameworks of both normative systems and multiagent systems. He defined normative multiagent systems as the following [10] :

“A normative multiagent system is a multiagent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other hand the normative systems specify how and in which extent the agents can modify the norms.” (page 6)

This definition is a comprehensive definition that represents the two-way effects of both norms on agents and agents on the norms. So in such a system, not only do norms influence agents' behaviours but also agents can influence norms with a level of authority for norm modification.

In our research, we use Boella's definition of normative multiagent system. However, we do not address norm modification by agents in our normative multiagent system, and so we can consider that norms are permanent during the life time of the multiagent system.

2.5 Summary

In summary, in Chapter 2, we have provided the necessary background of this research. As this thesis concentrates on dynamic assignment of rights and responsibilities to external agents in normative multiagent systems, in this chapter, we described the concepts of agent, multiagent systems, rights and responsibilities, external agent and normative multiagent systems which we will be using throughout the thesis.

In the next chapter, we will describe rights and responsibilities in detail from the normative viewpoint.

Chapter 3

Rights and Responsibilities

3.1 Introduction

The aim of this chapter is to present the definitions and concepts of norms related to the rights and responsibilities of external agents in multiagent systems. Using such definitions provides the basis for defining a normative language which is essential for representing any MAS inspired normative framework.

In this chapter, we mostly have adopted definitions from the relevant literature, for which we provide references in the text as well. However, we have found it necessary to emphasize the concepts of beneficiary, rights and compensation

(which can be claimed when a particular norm is not fulfilled); this emphasis is one of our contributions to this subject.

The important issues of norms are described in four Sections of this chapter. Firstly in Section 3.2, two classifications for norm types from the literature are presented, the second of which groups norms into three main types: *regulative*, *constitutive* and *distributive*. One of these classifications forms the basis of our work.

Then in Section 3.3, the main elements of *regulative norms* will be described (again, from the literature) including addressee of the norm, the legal modality (Obligation, Prohibition, Permission and Right), the act, time and any pre-conditions. These norm elements constitute the basis of descriptive normative languages.

In Section 3.4, enforcement of norms will be mentioned. This Section explains that for the enforcement of norms in multi-agent systems a precise mechanism is required. Such a mechanism controls the normative system at runtime so that it operates in accordance with defined regulations. The enforcement mechanism defines extra regulations over the normative system called *distributive norms* or *enforcement norms* which include check norms for detecting violations and reaction norms for reacting against violations. These norms can also themselves be defined in normative formalization languages.

Finally, Section 3.5 presents a descriptive normative language which is basically taken from the literature but with some additional features that we have added. Using such a descriptive normative language, one can formulate all the norms of a normative system. We also use a descriptive normative language to formulate all

the regulations, rights and responsibilities of external agents and their relevant enforcement norms. The formalization of all rights and responsibilities of agents will be applied to create a static knowledge base which we will discuss in Section 5.5.

3.2 Classifications for Norm Types

Not all norms are the same, and various authors have considered ways to classify norms. We present two main classifications in the literature: that of Vázquez-Salceda and his colleagues [80], and that given by Therborn [75]. We explain these two classifications because they are used in related works and they are complete enough to be the foundation of our work. Because after description of these two classifications, we compare them by explaining their similarities and differences. Then, we mention the classification we will use in this PhD thesis which is essentially that of Therborn.

First Classification: Based on the work by Vázquez-Salceda and his colleagues [80] in human regulations, three main types of norms can be observed:

1. Norms that provide the definition of abstract terms. This kind of norm provides basic definitions of terms in the application domain of the given normative system. As an example, in a vehicle traffic domain, the following definition is of this type of norm:

“Road vehicles are cars, buses and trucks.”

2. Norms which refer to the definition of an abstract action using sub-actions (a plan), a procedure or a protocol. This kind of norm provides the essential definitions for actions in the domain of application. As an example, in the police domain:

*“A request for personal data or **police certificate** is acceptable after receipt of the payment of £10 on police account.”*

3. Norms which refer to obligations, permissions and prohibitions. In fact, this kind of norm constitutes the main part of the legal system which specifies all the actions that an agent should do, not do, or may do. For example, in the traffic domain,

“It is forbidden for drivers to drive over the speed limit.”

While the first and the second type of norm in this norm classification are used to specify the definitions and vocabularies of the given legal domain, the third type of norm is concerned with the rights and responsibilities of agents. This type obviously is the center of attention in norm discussions in multi-agent systems, and we also mostly talk about the third kind of norms in the rest of this document.

Second Classification: A second classification of norms has been presented in Therborn’s work [75], cited in [7]. Based on this categorization, norms are classified from a different viewpoint; there are three types of norms: *regulative norms*, *constitutive norms* and *distributive norms*:

1. **Regulative norms** are those which help to regulate existing actions of agents. As an example, driving is an action which traffic rules help to regulate. This type of action can be done ignoring the regulations as well as following them, but regulative norms are used to regulate actions which

could be performed in any case. Regulative norms describe obligations, permissions and prohibitions.

2. **Constitutive norms** are non-regulative norms which have a classificatory or definitional character. In Searle's work [69], cited in [7], these types of norms have been called *counts-as conditionals* with the formalization of "*X counts as Y in context C*". For example, the following statement defines a classification:

"Motorcycles count as vehicles in the transportation domain."

The other well-known example is chess in which the rules of the game constitute the activities of the game. Such activities are dependent on these norms, as opposed to the regulative norms, where activities are independent from the norms.

3. **Distributive norms** define how rewards, costs and punishments are assigned to the social system. The main contribution of this type of norm is in the enforcement of the norms; specifying the rewards for executing a legal action or the punishment after a violation. Later, in relation to norm enforcement, these issues will be discussed in the Section *Norm Enforcement*.

Comparison of these two classifications: Comparing these two classifications, we conclude that the first two types of norms in the first classification are constitutive norms in the second classification. Also, the third type in the first classification has been divided into regulative and distributive norms in Therborn's classification.

In the second classification, norms are classified based on how they function in human interactions and these three kinds of norms tend to show different degrees of force. For instance, distributive norms tend to present a stronger reaction against violations. Otherwise, a distributive norm like a sanction is often just an additional norm. For example, the following norm is a distributive norm in a library application domain:

“If a borrower is obliged to return the book, and s/he does not return the book, then the borrower must pay the fine.”

The other point is that based on the second classification, regulative and constitutive norms are related together [7]. In [7], this relationship has been explained. It says in most of regulative norms obligations, prohibitions and permissions are conditional such that their conditions could either directly refer to entities and facts of the real world or refer to legal concepts or those from a more abstract classification of the world.

Referring to legal or abstract concepts is more typical, because in this way, conditions can be more independent from the common-sense view. In addition, norms require precise, agreed, definitions of individual concepts which is not normally found in common-sense vocabulary. For instance, referring to money instead of paper sheets or using properties instead of houses and fields are examples of precise norm concepts instead of common-sense vocabularies.

The classification we use: The classification we use in this thesis is the second one. The reason is that Therborn’s classification gives distinct definitions for regulative and distributive norms (in order to emphasize the importance of the distributive norms and present stronger reaction against violations). And we also

attempt to focus on distributive norms (or enforcement norms) and the implementation of them in our work.

We use all norm types of the Therborn's classification including regulative, constitutive and distributive norms in our work. Our normative system contains regulative norms including obligations, permissions and prohibitions; we will describe regulative norms in detail in Section 3.3. In addition, we use distributive norms or enforcement norms composed of sanctions (such as punishments and rewards) as well. So we will describe distributive norms and the importance of them in Section 3.4. As constitutive norms include legal and domain-related concepts and vocabularies of the normative system, in our work we specify these domain-related definitions in the normative knowledge base of the system. For example, later in the normative knowledge base of our auction example, we will define a domain-related term as "*fastPayer*" which will be defined as "*the winner of the auction who pays within 10 minutes of the ending time of the auction*".

3.3 Regulative norms

As mentioned in the last section, regulative norms specify the norms containing obligations, permissions or prohibitions.

Here the main elements of regulative norms including addressee, beneficiary, legal modality, act, scope, time and condition are presented. Then some key issues and aspects of legal modality will be explained, followed by some examples.

3.3.1 The key elements of regulative norms

As mentioned earlier, regulative norms that refer to obligations, permissions and prohibitions are the major focus in norm discussions. Regulative norms have some key elements for describing norms. In the following, the key elements of the regulative norms are shown by means of examples. We have used several references, taken inspiration from their original definitions, and then adopted with our domain of work using examples. Most of these references are from the work by Kralingen, Visser, Bench-Capon and Herik in [44]. However, we put the reference citation(s) we mostly used besides each definition.

Addressee of the norm [44]: The addressee of the norm is the norm's subject that can be specified by the norm for an individual, an agent, the public or the system. In the other words, the addressee is the agent or person who does the act.

Beneficiary of the Norm [36]: The beneficiary is someone who benefits from the norm. The beneficiary of the norm is as important as the addressee of the norm.

For example, in the following norm:

“In an Auction, the Winner of an item is obliged to pay the Seller the price of the item.”

The Winner is the addressee and the Seller is the beneficiary.

The legal modality (deontic modality) [44] determines whether the norm is either an obligation (ought), a prohibition (not ought) or a permission (may). In addition

to these legal modalities we consider right as a separate legal modality. Rights can be considered as a kind of permission with more features which needs to be made explicit in the context. We will describe rights with more details in the next section.

As an example, the following example shows a prohibition on placing a bid by the seller.

“In an auction, Seller is forbidden to place a bid.”

The act [44] is what the addressee is commanded, prohibited or permitted to perform. In the above example, *placing a bid* is the act.

Scope [44] of the norm specifies where the action is commanded, prohibited or permitted. For instance, while abortion is forbidden in some countries, in other countries it is permitted. Although sometimes scope is very essential, we do not consider it in our work, because we have a single scope in our context.

Time [44, 80]: Most norms are affected by time in different ways and the norm should specify *“When must something be done or is forbidden?”*.

The notion of time in norms can be divided into start-time, deadlines (if passed, these give rise to violations) and time limits. Time parameters can be attached to a norm with functions of *after(t)*, *before(t)* and *between(t1,t2)*.

Some norms will be activated from a moment of time for ever, such as:

“Smoking will be banned in restaurants after April 2007.”

Some norms are active for a period of time and after that they will be deactivated. However, some norms are timeless which means this type of norms expresses an obligation, permission or prohibition all the time. For instance, the following is an example of a timeless norm.

“Drivers are obliged to follow the traffic regulations.”

Conditions [80] for norms specify that activation or deactivation of a norm is subject to some circumstances. In other words, if some pre-conditions hold, the conditional norm will be activated or deactivated. For example, the condition may be occurrence of an action, such as

“If Winner pays the price of item, Buyer is obliged to send the item.”

Temporal and Conditional norms: In most cases, conditional norms contain time notions as well. For instance, we can have conditional norms with deadlines where the start of the norm is defined by a deadline [80]. An example of this sort of norm is the following:

“If a driver is penalized and does not pay the fine in two weeks, s/he will be obliged to pay double charges.”

3.3.2 The Key Issues on Legal Modality

Since the legal modality is the major element of the norm, here we present the key issues of legal modalities.

3.3.2.1 Definitions of Obligation, Permission and Prohibition

An **obligation** is an action which should be performed by the addressee. So one can say the addressee “*is obliged to*”, “*ought*”, “*must*”, “*has the duty to*” or “*is responsible to*” do an action. If doing the action is not performed, the addressee may be subject to some punishment or forfeit some right. For example:

“Everybody is obliged to pay tax.”

A **prohibition** is an action which according to the law, should not be done by the addressee. In this case, one can say the addressee “*should not do*” an action or the action “*is banned*”/“*is forbidden*” for the addressee. Like an obligation, the addressee may be subject to some punishment or sanction if the norm is violated. For example:

“In an auction, seller is forbidden to place a bid.”

A **permission** is an action that addressee is allowed to do. In fact, the addressee “*is permitted*”, “*is allowed*” to do and allowed not to do an action. For example:

“Students are permitted to access the university library.”

Right [68], for description of right, we refer to the definition of right in [68]:

“The idea of a right cannot build on the basis of obligation and permission alone. Such notions embed a teleological perspective, namely a focus on the purposes or interests. ... Only when such a proposition is concerned with the interests of certain individuals, we can view it as conferring rights upon these individuals.”(page 107)

With a right, if the action does not happen, the beneficiary will lose something and s/he can complain to some agent in authority for compensation. The following example shows the difference between a permission and a right in a private car park:

Norm 1: "A parking-permit holder is permitted to park his/her vehicle in the parking area."

Norm 2: "A parking-permit holder has a right to pass the parking bars using the code."

Norm 1 indicates a permission, but if the parking-permit holder does not park his car in the parking area, he will not lose anything and also if for any reason (such as not having a free space) he cannot park no complaint is acceptable. But based on Norm 2, if this person cannot pass the bar, he will lose his right and he can complain to the parking manager for compensation if somebody or something prevents him to enter the car park area.

In addition to the above classification of norms and its elements, legal modalities have more precise classifications in legal domain as well. In [68], Sartor considered normative positions in modality discussions as well identifying different subtypes for modalities; for example, *obligative right* or *absolute Right Obligation*. However, we do not consider these subtypes in our context to avoid complexity unnecessary for our purposes.

Examples: By way of illustration, here a couple of examples are included with their norm elements are specified.

“Students are permitted to access the university library.”

The addressees of the above norm are individuals, the legal modality is permission, and the act is library access.

- *“Everybody is obliged to pay tax.”*

This is a norm example directed to the public and describes an obligation for the act of payment.

- *“In an auction, seller is forbidden to place a bid.”*

This is another example which shows a prohibition against placing a bid by the seller.

3.3.2.2 Different Meanings

It is important how a norm is described, since the same norms can lead to different rights and responsibilities being assigned to agents. To clarify, here an example is presented which shows how a set of general norms can be expressed in two ways with two different meanings.

For example, the general norms say:

“The winner of the auction is obliged to pay the price of the item.”

“Auctioneer has the right to receive commission fee.”

“Seller has the right to receive the payment.”

“The commission fee is 10% of the payment.”

These general norms could be realized in several ways. First, suppose that in the real system the following occurrences happened:

“The payment is £100.”

“Sarah is seller.”

“Ali is auctioneer.”

“David is winner.”

The mentioned general norms can be realised in at least two ways:

1. Winner pays to the seller and seller pays the auctioneer. So in this case the following rights and responsibilities will arise:

“David is obliged to pay Sarah £100.

Sarah has the right to receive £100 from David.

Sarah is obliged to pay £10 to Ali.

Ali has the right to receive £10 from Sarah.”

2. Winner pays the auctioneer and auctioneer pays the seller after deducting a commission fee. So in this case the following rights and responsibilities will arise:

“David is obliged to pay £100 to Ali.

Ali has the right to get £100 from David.

Ali is obliged to pay £90 to Sarah.

Sarah has the right to receive £90 from Ali.”

Both of these procedures will realise the norms. If, however, the norm is violated, different people will be liable. In the first case, Ali will collect his commission from Sarah, while in the second case he must get it from David. Which is preferred will depend on the context but an agreed procedure is required if responsibility of violation is to be assigned.

So it can be seen that different interpretations of a general rule can result in different definitions for the detailed norm description, and then by different rights and responsibilities at runtime of the system. To sum up, the way of defining a norm is important for interpreting the norm and its implementation.

3.3.2.3 Minimizing Legal Modalities

Legal modalities including obligations, prohibitions and permissions can be minimized. Sartor [68] described how one can minimize deontic modalities. He mentioned:

“Our minimal deontic logic can be limited to the following definitions and axioms:

Being prohibited to perform an action means being obliged not to do it:

Forb A ≡ Obl NON A.

Being permitted to perform an action means not being forbidden to do it:

Perm A ≡ NON Forb A.

*Being obliged to perform an action entails being permitted to perform it:
IF Obl A THEN Perm A.*

*Being both obliged to perform action A and obliged to perform action B
entails being obliged to perform both actions: **IF (Obl A AND Obl B)
THEN Obl (A AND B).**”(Page 136)*

The first statement ($\text{Forb } A \equiv \text{Obl NON } A$) shows that prohibition can be defined with obligation and negation. The second statement also shows that a permission action can be defined using negation and forbidden. As a result, these two statements show that the different cases of legal modalities can be defined with one legal modality (such as obligation) and negation.

The following table reproduced from [68] shows an example of different cases of deontic notions. This table represents some equivalences in deontic notions. For example, it shows the first case of the above minimizing modalities $\text{Forb } A \equiv \text{Obl NON } A$ which says “in France, the rule of *women are forbidden to wear the veil* is equal to *women are obliged to not wear veil*”.

country	wearing the veil (V)	not wearing the veil (NON V)
France	<i>Forb V</i>	<i>Obl NON V</i>
Iran	<i>Obl V</i>	<i>Forb NON V</i>
UK	<i>Perm V</i>	<i>Perm NON V</i>

Table 1-Complete Deontic Qualifications reproduced from [68]

From the above discussions for minimizing deontic modalities, it is clear that the definition of permissions is convertible to obligations and also obligations can be converted to prohibitions using negation. As a result, one could conclude that all deontic modalities of norms can be defined using only one primitive modality (say obligation) and negation.

3.3.2.4 Examples of Minimizing Legal Modalities

As we mostly use auction example in this thesis, here, we illustrate some examples of minimizing the rules in an auction system which shows how deontic modalities can be converted together.

- **Converting Permission to Prohibition or Obligation**

A permission norm says:

“Buyer is permitted to place a bid.”

This norm can be defined as:

“Buyer is not forbidden to place a bid” or “Buyer is not obliged to not place a bid.”

- **Converting Obligation to Prohibition**

An obligation norm says:

“If Buyer wins the auction, s/he is obliged to pay the price of the item.”

This norm can be defined as:

“If buyer wins the auction, it is forbidden for her/him to do not pay the price of the item.”

3.3.2.5 Legal Modality in Our Work

Although we say that using negation and a primitive modality is enough for defining all deontic modalities, we still use all legal modalities including obligations, prohibitions, permissions and also rights. We do so for three reasons.

First of all, using all three modalities simplifies the definition and understanding of the norms. That is why most normative systems use all legal modalities.

Secondly, from the implementation viewpoint, we use separate notions for obligations, prohibitions and permissions and rights because one of our objectives of developing an application for dynamic assignment of R&Rs to external agents is *to inform agents of their rights and responsibilities* at runtime. We do so even if the implementation of some deontic notions such as permission seems unnecessary in many systems, because in our application we desire to inform the agents when a permission norm will be activated for them. The benefit of this approach is faster user responses to the system and consequently faster dialogs.

Thirdly, norms may often have different attitudes towards violation: prohibitions lead to commissive violations, obligations to omissive violations. Also styles of norm design get rid of different defaults: *everything is permitted unless forbidden* or *everything is forbidden unless permitted*. In general, norm systems are underspecified, and so the default can be important.

As mentioned in Section 3.3.2.1, there are more precise classifications in the legal domain, but we will mostly focus on the implementation of norms in multiagent systems. Thus, we do not consider more different subtypes of legal modalities in more detail.

3.4 Norm Enforcement

Although the primitive declaration of norms is fundamental in normative systems, the enforcement of norms is another important issue in norm discussions. The reason is that the implementation of normative systems needs to consider special mechanisms for executing norms; thus, taking into account techniques for norm enforcement is inevitable in normative multiagent systems.

In this section, firstly the necessity of enforcement norms is discussed, and then the solutions for norm enforcement are explained.

3.4.1 The Necessity of Enforcing Norms

In all normative systems, there is a set of substantive norms which describe the society's desired behavior which we have completely explained so far. However, after specifying substantive norms in a normative system it is necessary to control the system so that it operates according to these norms. Generally speaking one could say that enforcing norms is essential because of the following problems:

1. There always exists the possibility of violations. Violations are illegal actions or states that may occur. With respect to the legal modalities, *violation* can occur in the following cases:

- An obligation is not fulfilled by the end of the period of obligation.
- A prohibition (forbidden) activity occurs in the duration of prohibition.

Note that permissions are never violated by the addressee of the norm. However, it is possible that other factors prevent the addressee from exercising its permission. For example, if an auctioneer prevents a member from logging-in to the system then the member's permission to login has been violated.

2. Some norms have a beneficiary in addition to an addressee. In these cases, when a violation occurs by addressee, the beneficiary loses some rights. Therefore, the beneficiary may be eligible to *claim for compensation* from an agent in authority.
3. In some case, the legislator of the normative system allows some *rewards* for encouraging agents to act according to the law.

Because of the possibility of these problems, a normative system should be controlled somehow. In order to control the operation in accordance with the norms, and detect and manage compliance, violation, compensation, and reward, normative systems usually have enforcement mechanisms. Enforcement mechanisms comprise two phases: first *detection* of one of the above cases; and second, accomplishing predefined *reactions* (based on the regulator's specification).

Enforcement mechanisms define extra regulations over the normative system called *distributive norms* or *enforcement norms*.

Enforcement norms consist of a set of norms regulating checks and reactions on violations of other norms. Therefore, there are two types of enforcement norms: *check norms* and *reaction norms*.

We now explain these two types of norms.

3.4.2 Check Norms

Based on the above discussion, enforcement norms requires that first of all violations are detected. In addition, enforcement norms should be defined so that the normative system is always aware of eligible compensation cases, in order to execute the compensation when a beneficiary makes an application. The enforcement norms should also be defined in such a way to detects reward cases, if any.

Check norms specify the operationalization of the norms of the normative system for detection of any violation, compensation or reward cases. Different systems have different mechanisms to do these checks. Some of them have random checks to detect violation, compensation or reward cases, or some of them check the system based on a schedule [33]. Therefore, check norms determine who and when the system has to be checked for detecting violation, compensation and reward cases. Regarding who should do such detections we will discuss in Section 3.4.7.

3.4.3 Reaction Norms

We said that, one stage of the norm enforcement is detection of abnormal behavior against the main norms in the normative systems by check norms. The next stage to complete norm enforcement is to define a plan of action to respond to the actions

of agents relevant to norms. Such a plan would be a *punishment* when a violation occurs or a *reward* when a norm is retained or compensation when occurrence of a violation causes some loss of rights of a beneficiary.

In the following we provide more descriptions of punishment, reward and compensation which are inspired by the citation in front of their bullet title.

Punishment [75, 80]: Punishments are actions to punish the violator when a norm violation occurs. In the other words, after detecting the norm violation, punishment norms define what the responses to the violation are. For instance, additional obligations or loss of permissions may be a kind of punishment.

Reward [75]: Although the normative system mostly investigates detection and management of violation, considering rewards for whom acts aligned with regulations would also be very useful to encourage agents comply with the law. Rewards are supplied when the norms executed and no violation of such norms has occurred. It is often used for prompt compliance. In some cases, reward is for encouragement of people for undertaking a permitted action. For example, customers are permitted to fill survey forms, but, in order to encourage them to fill the forms rewards might be offered to provide them with an incentive.

In this case, the normative system contains some rewards for acting some of the legal actions. For example, additional permissions or entitlements may be a kind of reward.

Compensation [36]: When an obligation or prohibition is violated, a punishment will be applied on the violator (or addressee). But the other problem is if the norm

has a beneficiary, the beneficiary still has the right to receive something. In other words, if the addressee of the norm does not perform the command (obligation or prohibition) or if addressee of the norm violates, the beneficiary lose his/her rights.

In this case, law may consider a right for beneficiary to claim which means the normative system usually anticipates some services for the beneficiary, if s/he claims for compensation. Such facilities might be providing the whole right of beneficiary or a part of that or any other form of compensation.

For illustration, we refer to the example on Section 3.3.2.2. Recall that in an auction system the status of the system is as follows:

“The payment is £100.”, “Sarah is Seller.”, “Ali is auctioneer.” and “David is winner.”

Suppose that David does not give the payment. Although David did not pay, in the first case, Sarah still is obliged to pay £10 to Ali and in the second case, Ali is still obliged to pay £90. Furthermore, in both cases Sarah and Ali can ask for compensation from the auction manager.

The topic of beneficiary, right and compensation is one of our main concerns which we will emphasize in our implementation as well.

To sum up, compensations will be given when a violation occurs, then an agent – who is the beneficiary of the norm - loses his rights and he can apply for compensation against that violation.

Based on the above definition we can distinguish compensation from the reward. Compensation will be given as a result of a norm violation, but reward will be given to an agent because of compliance with the norm and no violation. Compensation differs from punishment in that the beneficiary of the norm receives something. For example, compare restitution with imprisonment for theft: in the first case the victim is given something, whereas in the second he is not.

It is good to mention that it depends on the legislator of the normative system to define punishments, rewards or compensations; in some cases, systems have not seen any need for a sanction mechanism.

In Figure 1, different cases of enforcement of a norm have been presented. This figure shows that the enforcement mechanism might detect a violation or no violation for a norm during runtime of the normative system. When no violation occurs the system may or may not foresee a reward for observing the norm. When a violation has been detected, the reaction may be a punishment or compensation (if there is a beneficiary who claims compensation). However, as the figure shows there may exist some cases that the legislator has not foreseen any reward, punishment or compensation.

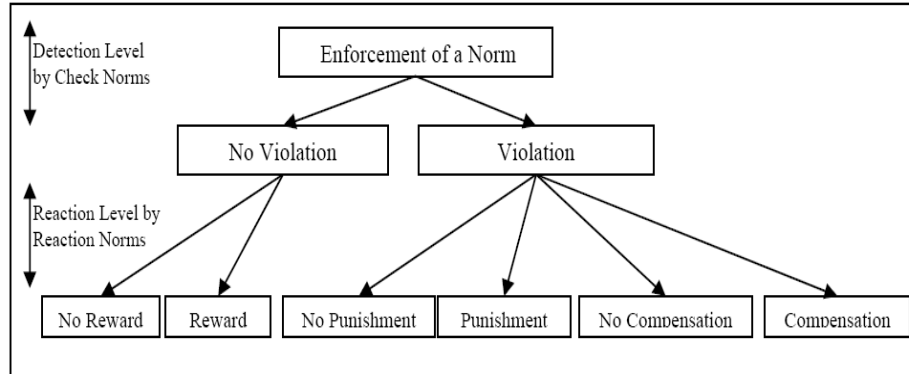


Figure 1-Different Cases for enforcement of a Norm

3.4.4 Examples of Enforcement Norms

To clarify, we present some examples to illustrate punishment, compensation and reward. The first example shows enforcement norms for a violation case.

“Norm: Winner is obliged to pay the item’s price in one day.

Check norm: The auction manager should perform random checks of the payments status every day.

Reaction norm: If a winner has not paid by the deadline, then winner will be fined accordingly.”

The second example shows a violation followed by a compensation claim.

“Norm1: Seller is obliged to send the item to the buyer after buyer paid up to 7 days.

***Norm2:** Buyer has the right to receive the item after the payment up to 7 days.*

***Check norm:** The auction manager should randomly check the status of sending the item every day.*

***Reaction norm1:** If a seller has not sent the item by the deadline, then buyer has the right to claim for compensation.*

***Reaction norm2:** If a buyer makes a claim for compensation and s/he has the right to compensation, compensation should be applied for the buyer. ”*

The third example shows the norm enforcement for a reward case. *There is not any basic norm for reward cases*

*“**Check norm:** Every day, the auction manager should randomly check the payment times of buyers to find those buyers paid in 5 minutes after the auction.*

***Reaction norm:** If winner pays the payments in 5 minutes (the deadline is 1 hour) after ending time of the auction, then winner should get double positive feedback as a reward. ”*

3.4.5 Enforcement-Norm Elements

In Section 3.3.1, key elements of the substantive norms are specified which are necessary for formalizing descriptive normative language. Similarly enforcement norms have key elements which should be formalized into the descriptive normative language as well. These elements consist of the notions of ***punishment***, ***compensation*** and ***reward*** which were described earlier in Section 3.4.3.

3.4.6 Norm Enforcement versus Norm Regimentation

Here it is necessary to mention a distinction in the methods for executing norms. From the implementation viewpoint there exist two ways for executing norms in multiagent systems: *Norm Regimentation* and *Norm Enforcement*.

Norm Regimentation [34] is an obvious way in which the fulfilment of the norms of a normative MAS can be implemented by making the violation of the norms impossible, so that norm compliance is inevitable. Regimentation guarantees the fulfilment of the norms in a multiagent system. As an example, in e-commerce, when shopping on the web, your goods are not delivered before giving approval for using the credit card number to pay those goods. So no customer can violate in this case.

However, for instance, Grossi has argued in [34], if no violation can happen, if nothing can go wrong, it does not make sense any more to talk about norms at all. From the agent point of view, a regimented norm is just a fact can be implemented by protocols; furthermore, the autonomy of agents is robustly limited. In addition, in a multi-agent system where the agents are programmed by different design teams, it is not possible to definitively verify that the programming code of an agent satisfies particular conditions, so norm regimentation is ultimately impossible in open agent systems.

Furthermore, providing the autonomy of violation to agents is desirable as well in some cases in normative systems. For example in traffic regulation, passing the red light is forbidden for drivers, but in an emergency case may be necessary and violation is actually desirable.

Norm Enforcement includes reactions that the normative multiagent specifies for responses to a violation of its norms. Therefore, enforcement presupposes the possibility of violation [34].

It has already been mentioned that the enforcement of norms in normative multiagent systems requires a mechanism for recognizing the occurrences of violations of the norms and subsequent responses to these violations. As mentioned before, this check-react mechanism is provided by means of additional norms. Regulation on car insurance is a typical example in this sense: car insurance is impossible to be regimented, it should be checked and possible violations detected. Once the detection takes place, specific reactions are also specified and made obligatory.

3.4.7 Enforcing the Enforcement Norms

So far we have said that to establish the norms of any normative multiagent system one needs additional norms called “Distributive Norms” or “Enforcement Norms”. As mentioned, these norms decide the appropriate reactions in cases involving punishments, rewards or compensation. Such norms are necessary for executing norms, otherwise behaving according the law and against to the law cannot be distinguished and violations are not detected.

Now we note that all of these norms should be enforced by an agent (or agents) in authority. In different human organizations, different roles have the duties of enforcing law, for example, police, and inspectors. In multiagent systems such roles are a special case of internal agents called “*Enforcers*” [17], which detect and enforce the norms, subject to norms defined in the system obliging them to do so.

All the responsibilities of an enforcer are defined in a way similar to the rights and responsibilities of other internal agents.

However, the question might then arise as to how to enforce these norms (defined as the responsibilities of enforcers) on enforcers themselves. Multiagent systems (similar to the real organizations) might include different levels of enforcers such that the actions of each level are always controlled by enforcer agents at a level above. But the enforcement chain cannot continue forever, and so must end somewhere. To achieve this, a *root enforcer* is defined such that one needs to have full trust in the root enforcers [17]. For more information about the management of the levels of enforcers we refer to [17], because in our MAS we do not consider levels of enforcers and just use one enforcer. An alternative is to use regimentation for the enforcers.

3.5 Formalization of the Norms

In order to implement a normative system, all norm definitions, classifications and enforcement issues need to be formulated in a descriptive normative language. Such a normative language makes it possible to describe norms based on the norm notions we have presented in this chapter. To date, several formalization methods have been proposed to formally describe norms. However, only a few of them have been implemented.

One of the descriptive normative languages that completes the former works has been presented by Silva in [71]. Appendix B shows a part of this grammar. In [71], the description of the normative language defines norms as the composition of:

- a *deontic* concept (obligation, prohibition or permission)
- punishment and reward
- an action/event and the relevant functions
- a temporal situation
- an *if* condition, when pertinent.

Although the above language provides the basic descriptive normative language, because of the lack of some essential normative notions such as *rights* and *compensations* we have extended the grammar of this normative language to support the notions of beneficiary, rights and compensation in multiagent systems. The following grammar shows our extended grammar.

```

<deontic_concept> ::=      'OBLIGED'      |      'FORBIDDEN'      |
'PERMISSION' | 'RIGHT'

<sanction> ::= <punishments> <rewards> <compensations>

                <punishments>
                | <rewards>
                | <compensations>

<compensations> ::= '(COMPENSATION: '<compensations>')'
| '(COMPENSATION: '<compensation>')' <compensations>
| '(COMPENSATION: IF' <if_condition> <compensation>')'
| '(COMPENSATION: IF' <if_condition> <compensation>')'
<compensations>

<compensation> ::= <authority>'COMPENSATIONS' <action>
| <authority>'COMPENSATIONS' <expression>
| <authority>'COMPENSATIONS PERMISSION '
<norm_description>

```

In addition to the notions we have just described for formalizing descriptive normative language, we need to extend temporal functions by adding a function denoted AT(t), to represent the occurrences of events. So the extended grammar of temporal functions would be as:

```

<temporal_situation> ::=
    BEFORE '(' <situation> ')'
  | AFTER '(' <situation> ')'
  | BETWEEN '(' <situation> ',' <situation> ')'
  | AT '(' <situation> ')'

```

Therefore, the normative language we use in this thesis includes the following notions: deontic concepts (characterizing obligation, prohibition, permission and right concept); punishment; compensation; reward; temporal functions; and if functions. In addition, we use action/event functions but we will say more about them and our additional event functions in Chapter 4 and Chapter 5, since these functions are more related to the implementation of norms.

Although when we explained norm classifications, regulative norms and the other definitions in this chapter, we specified the definitions we will use in our PhD thesis, here once more we clarify and emphasize norm-related definitions that we will use in our model. Because, all these norm definitions, classifications and enforcement issues will be formulated in the descriptive normative language that will be used in implementation of our methods for dynamic assignment of R&Rs to external agents.

As mentioned, we use Therborn's norm classification which classifies norms into regulative, constitutive and distributive norms. We use all of these types in our work, but we do not have a special definition for constitutive norms, because constitutive norms are defined for every application domain explicitly. We defined the main norm elements of regulative norms and also the main enforcement elements of distributive norms. In the following, we specify the main norm elements and the enforcement norm elements which will be used in this PhD thesis.

The **main norm elements** which we use in the normative language are as follows:

- Addressee of the norm: The person or agent who does the act.
- Beneficiary of the Norm: Someone who benefits from the norm.

- The legal modality: The Legal Modality determines whether the norm is an obligation, a prohibition, a permission or a right.
- The act: The act is what the addressee of the norm is commanded, prohibited or permitted to perform.
- Time: Time parameters can be attached to a norm with functions of after(t), before(t), between(t1,t2), and at(t).
- Conditions for norms specify that activation or deactivation of a norm is subject to some circumstances.

Therefore, from the list of norm elements, we only do not consider scope in our work, because we have a single scope in our context.

The **enforcement norm elements** which we use in the normative language are as follows:

- Punishment: Punishments are actions to punish the violator when a violation occurs.
- Compensation: Compensation is a service that the normative system anticipates for the beneficiary of the norm when an obligation or prohibition is violated and the beneficiary of the norm loses his/her rights.
- Reward: Rewards are services which a normative system may provide for agents whose acts align with regulations, to encourage agents to comply with the law.

3.6 Summary

This chapter has provided an overview of the normative concepts relevant for normative multiagent systems. As the aim of this research project is to provide dynamic assignment of rights and responsibilities to agents, this chapter explained all important issues relevant to rights and responsibilities of agents. We have taken these concepts from literature adopted and extended them.

This chapter started with the basic norm definitions such as norm classifications and the key elements of norms. Then it continued with the issues important for the execution of norms in multi-agent systems. In the last section, we described that in order to apply normative issues in multiagent systems, it is necessary to formulate all the mentioned norm elements into a descriptive normative language. Finally, we specified the language, building on the work of Silva [71], and the complementary extensions of that language- which we added to support the notions of right and compensation. This language will be used in the remainder of this dissertation.

Chapter 4

Dynamic Issues in Normative MAS

4.1 Introduction

So far we have provided essential background to the normative multiagent systems (in Chapter 2), followed by discussing important issues of rights and responsibilities in normative multiagent systems (Chapter 3). We now discuss the issue of providing dynamic assignment of R&Rs to agents in normative multiagent systems. In the next section of this chapter, Section 4.2, the main dynamic factors influencing assignments of rights and responsibilities are described. Then, in Section 4.3, different types of dynamic assignment including dynamic assignment of roles to agents, dynamic assignment of rights and responsibilities to agents, and dynamic assignment of sanctions to agents, are explained, along with illustrative

examples. In Section 4.4, the difference between protocol-based norms and rule-based norms will be explained to complete the discussion. Finally, in Section 4.5 a summary will be provided.

4.2 Dynamic Issues in MAS

One of the main characteristics of open multiagent systems is that their environment is dynamic. In a dynamic environment, there exist unpredictable processes operating on the system, and changes occurring beyond the control of the system. Accordingly such dynamism influences the whole system, including any process of assignment of rights and responsibilities of roles to the agents.

Therefore, the first step for handling dynamic assignments of R&Rs to agents in normative MAS would be recognition of the sources of dynamism and changes in open MAS; we discuss this issue in this section. Then, the next section will show the influence of these sources on the process of dynamic assignment of rights and responsibilities to agents.

4.2.1 The Source of Dynamism

In order to start the explanation of sources of dynamism, we first summarized the definition of open multiagent systems from the Introduction of the paper of Sierra and his colleagues in [70] as follows:

*...Open systems are characterized by unknown components which can **change over time**, and can be self-interested human and software agents developed by diverse parties. Therefore, **open multi-agent systems (MAS) are considered as distributed systems where (possibly) large, varying populations of agents with different behaviors cooperate**. Such agent cooperation is not fixed at design time but may emerge **at real time**....(Page 2)*

This description states that the number of agents connected to an open system is unpredictable and may change. Therefore, the population of agents is not fixed at design time, but only emerges and may change at run time. Consequently the population of agents is a dynamic factor in the environment of open multiagent systems. For example, in a session of an auction system, there may be six buyers at the beginning. After a few minutes perhaps four of them remain, two of them leave the session and one new buyer may join the session.

4.2.2 The Source of Changes

Runtime changes also influence a MAS. In a normative multiagent system, runtime changes may cause a rule from the MAS to be applied. So the performance of dynamic assignment of rights and responsibilities to agents needs to recognize sources of changes as well.

The major sources of changes which affect MAS are actions and environmental events. Here, along with describing the sources of changes, we use an example to illustrate how a normative MAS may be influenced by occurrence of changes.

The sources of changes in normative multiagent system can be categorized as follows:

1. **The action of agents:** The action of an agent is what an agent does, which is the ability of the agent to affect its environment. Therefore if an agent does an action, a change may have occurred in the MAS. For example, suppose that Mari is a Seller. Then the following change happens:

“Mari advertised a gold watch for Auction5 at 10:00.”

which shows that an agent (Mari) did an action (advertising). Following this action, some norm will be activated for this agent (Mari who plays the role of Seller) such as prohibition for placing a bid in Auction5.

2. **Environmental events:** A static environment remains unchanged except by the performance of actions by agents, but in dynamic environments there are also other processes – we call them *environmental events* – that change the state of the MAS in ways not in the control of the agents in the system. In the other words, environmental events are significant occurrences or changes in an agent’s environment that arise not from the action of agents, but which the agent should respond to in some way. Here we divide environmental events in three parts:

- **Action of other agents:** Sometimes the action of the other agents is the source of change in the status of an agent, compared with the first bullet which the action of the agent leads to change of its own status. For example:

“David placed a bid of £30 for the gold watch in Auction5 at 10:15.”

Suppose Ali is the auctioneer of this auction, then David's action (placing a bid) is an event that gives rise to an obligation for Ali (as the auctioneer) to validate the bid. It means that the action of another agent leads to a status change for Ali.

- **Parameter Changes:** The system may have some environmental variables which may change at runtime. For example, in auction system members feedbacks are parameters and are counted. So "*increasing the number of negative feedbacks*" can be a source of change. Suppose that after the latest negative feedback increment for an agent, the number of its negative feedback reaches 3 in total, and then given a particular auction rule, the agent account will be suspended.
- **Passage of time:** The notion of time is very important in normative multiagent systems and passing time can be a source of change as well. By passage of time and reaching to critical times of the system new rights or responsibilities may fire. These critical times can be divided into start-time, deadlines (if passed, these give rise to violations), and time limits. As mentioned in Section 2.2.2, time parameters can be attached to a norm with functions of $\text{after}(t)$, $\text{before}(t)$ and $\text{between}(t_1, t_2)$.

For example, suppose that "*Auction5 ends at 11:00*". And the auction regulation states "*Auctioneer is obliged to close and declare the winner of the auction*". Reaching this time then imposes a new obligation on the auctioneer to close the auction session and declare the winner.

- **Network Problems:** Network problems such as disconnections, low-speed and transmission delays in sending message can also be significant environmental events. For example, "*Due to an unpredictable*

disconnection, auctioneer did not receive the last bid of Mary.”. In this case, some rights or responsibilities may fire, if the normative system has foreseen supportive services. However, in this context, we do not consider any network problems and we assume that the underlying network operates without problems.

4.3 Dynamic Assignments

In the previous section the main sources of dynamism and change have been introduced. In order to respond to such changes and deal with this dynamic environment, an investigation of novel approaches and techniques is necessary.

For example, open MAS are characterized by unknown agents in which the population of agents can change over the time. Considering the organization structure in such a dynamic environment, agents join to the system dynamically, so their roles should be assigned to them dynamically as well. Dynamic assignment of roles to agents is an instance of organizational-based approach [64].

The research of this thesis also aims to present dynamic assignment of rights and responsibilities of roles to agents, as a new approach for improvement of the management of the normative multiagent systems at runtime.

Along with the objective of this research, in this section we precisely explain different types of dynamic assignments including: *dynamic assignment of roles to*

agents, dynamic assignment of rights and responsibilities to agents, and dynamic assignment of sanctions to agents.

4.3.1 Dynamic Assignment of Roles to Agents

The method of dynamic assignment of roles to agents is defined as a systematic way in which, taking account of conditions of roles, the capabilities of agents and the overall context of actions, roles are dynamically assigned to agents, by a group organizer or a management system of the MAS [64].

For example, in a supermarket, suppose *Ali* (as an agent) is a *sales assistant* (his current role) and he has achieved some new capabilities and experiences which are matched with the conditions of *department supervisor* (as a role). In the real time operation of the system, when the *manager* detects this match and assigns the role to *Ali*, a corresponding dynamic assignment of roles to the agents to reflect this change of status must occur.

In the auction example, when “*Mari logs into the system initially as a member*” she chooses to be a buyer in the auction session of Gold Watch. So the central management system gets her request, checks the auction session’s conditions (e.g. “*There is a minimum age limit of 18 for joining to this auction, because of the high price.*”) and provides a history check for Mari too. After passing the checks successfully, the role of buyer, and its accompanying rights and responsibilities, will be assigned to Mari by the management system.

Note that, here we use the word of *assignment* to state that dynamic assignment is a *management* task to assign roles to agents as required by events and actions. There

is another dynamic way in which agents can themselves decide which roles should be employed for achieving specific goals. In the context of this thesis, however, the roles are assigned to agents by the management system, and agents do not choose their roles by themselves (although their assignment may be the result of one of their actions intended to lead to the assignment of that role). The idea of *dynamic assignment of roles to agents* has been previously presented and supported by some methodologies. In [64], the authors described dynamic assignment of roles to agents and the supporting methodologies, followed by an evaluation and a comparison table.

In Chapter 8, *Discussion*, we will discuss related work which includes the works on dynamic assignment of roles to agents and also compare them with our work.

4.3.2 Dynamic Assignment of R&Rs to Agents in Normative MAS

As we mentioned before, identifying and addressing the problem of dynamic assignment of R&Rs to agents for the first time is one of the main contributions of this PhD thesis. We also have presented this part of work in [16].

Recall that dynamic assignment of rights and responsibilities to agents is proposed to improve the management of normative multiagent systems. Normative multiagent systems are multiagent systems together with normative systems in which agents can decide individually whether to follow the represented norms in the normative system [7, 9].

The normative part of the normative MAS represents all norms that apply to agents. Such norms indicate the obligations, permissions, prohibitions, rights and norms related to sanctions (including check norms and reaction norms), as described in Chapter 2.

As in other multiagent systems, the concepts of *role* and *rights & responsibilities* can be used in the structure of normative MAS, so norms in normative MAS (which apply to agents) can be considered as rights and responsibilities of roles which are assigned to agents at runtime.

Therefore, when at some time-point during runtime operation of the system a role is assigned to an agent, all the norms related to that role can be assigned to that agent. For example, in an auction system, there may be a set of rights and responsibilities for the role of *Auctioneer*. Thus, as long as “*Ali plays the role of auctioneer*”, he should follow the whole set of norms related to the role of *Auctioneer*.

Although once the role of the agent has been allocated, all the rights and responsibilities of the agent are identified, some of these may be conditional on the state of the system, and our approach attempts to specify and assign the specific right or responsibility of an agent at each instant of runtime.

There are two main elements in such an assignment: *first*, the represented norms of the normative MAS, and *second*, the dynamic triggers including the actions of the agent or other environmental events.

From the normative viewpoint, as the rules of the normative system are conditional and time-related, a norm will be fired when the condition of the norm holds or an important time point is reached. From the MAS viewpoint, the sources of dynamism and change influence the environment.

As a result, the knowledge base of the normative system also contains all conditional rules (R&Rs of roles). When a change occurs in a normative MAS, a pre-condition of a norm may become satisfied, and the corresponding norm will then be fired. We have already defined the sources of dynamism and changes as: changing the population of agents; occurrence of an action; or an environmental event. Therefore, occurrence of any of these types of sources may cause the pre-condition(s) of a right or a responsibility to be satisfied, so that a dynamic assignment of R&Rs then takes place.

For example, suppose that the following norms are valid in our auction system:

***Norm1:** “The Auctioneer is obliged to reject lower bids, during the auction session.”*

***Norm2:** “During the auction session, if a lower bid is placed and Auctioneer did not reject it, Punishment_2 will be applied for Auctioneer.”*

According to Norm1, the obligation is activated and assigned to Auctioneer agent only during the auction session. Norm2 shows that if Auctioneer agent violates during the auction session, s/he will be punished. So if the condition of this norm is satisfied, the norm will be activated and assigned to Auctioneer.

As a result, the activation and deactivation of the above norm is subject to the conditions of time (during the auction), event (place a lower bid) and action (rejection of the bid). Thus it can be concluded that the activation and deactivation of *each specific norm* happens dynamically at runtime. So assigning *each activated norm* to *the relevant agent* will also be a dynamic task. In this work we aim to provide such assignment.

4.3.2.1 Tri-level Structure for MAS

We use a Tri-level structure for MAS consisting of agents, roles and R&Rs is demonstrated. In this structure, the first level includes all the agents who can join to the MAS, the middle level consists of all predefined roles in the MAS, and the third level includes all R&Rs of roles. The aim of using such a structure is to show firstly how roles can dynamically be allocated to agents, and secondly how a right or responsibility of a role can dynamically be assigned to an agent. The structure permits an elegant separation of the three components (agents, roles, R&Rs), so that duplication in system representation is avoided and so that dynamic re-assignments (of roles to agents or R&Rs to agents) are facilitated.

This structure is illustrated with an example. The following figures shows dynamic assignments of *roles to agents* and *R&Rs to agents*, in a MAS based on this tri-level structure of an MAS with agents, roles and rights and responsibilities.

Figure 2-A shows the initial status of an auction system, when members join to the system. At this stage, none of them play a specific role and also there is not any enforcement of the norms.

Figure 2-B shows the status of the system just after selecting the roles of the agents (based on agents actions). This assignment is a dynamic task, because the roles of agents are assigned at runtime. At design time, it is not specified which agent will play which role(s).

Figure 2-C shows the status of the system at the start time of the auction. The related norm of the Start Time is stated:

“Auctioneer is obliged to declare the start of the auction at the Start_Time.”

The above norm will be activated and assigned to *Mari* who is the *Auctioneer* of this auction session. Therefore, at the start time, there is an obligation assigned to *Mari* (as an external agent) which says:

“Mari is obliged to declare the start of the auction at the Start_Time.”

Suppose that *Mari* declared the start of the auction. **Figure 2-D** shows the other status of the system just after declaring the start of the auction by *Mari*. There are two related norms for this stage, as follows:

“Buyer is permitted to place a bid after starting the auction.”

“Seller is forbidden to place a bid during the auction.”

The above norms will be dynamically activated and assigned to the external agents who play the role of *Buyer* and *Seller*. As the figure shows two buyers (*Ali* and *David*) and one seller (*Sarah*) exist for this auction, so that the result of dynamic assignment would be:

“Ali is permitted to place a bid after the auction has started.”

“David is permitted to place a bid after the auction has started.”

“Sarah is forbidden to place a bid during the auction.”

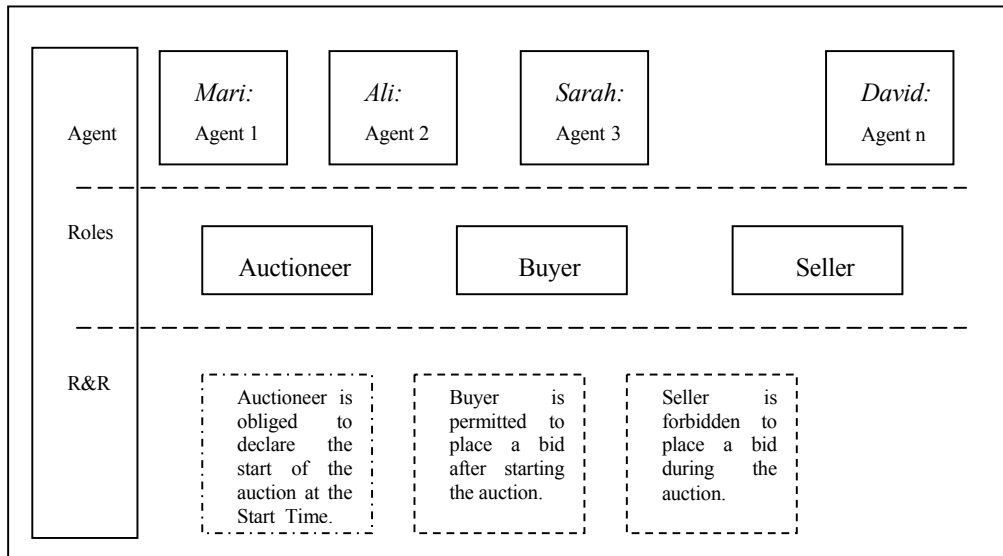


Figure 2-A-Time 1: Agents log into the system

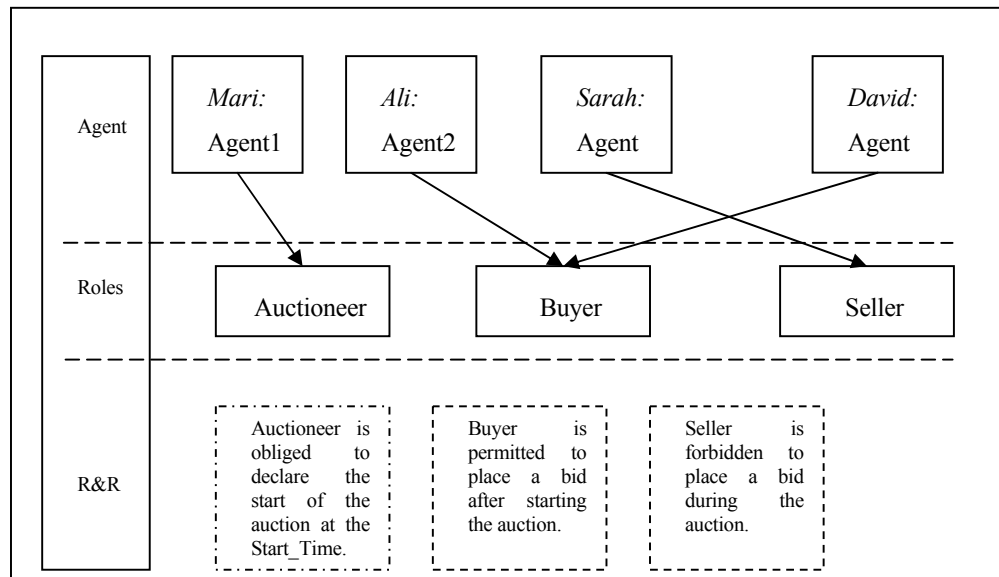


Figure 2-B-Time 2: Agents play particular roles

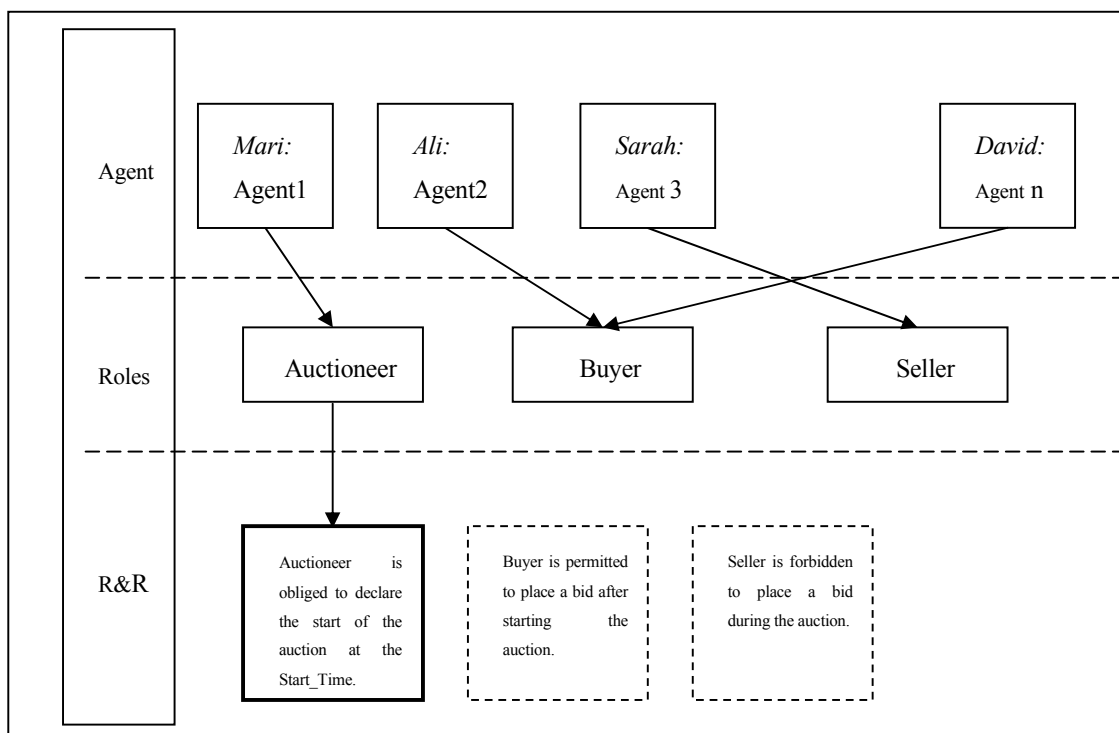


Figure 2-C -Time 3: At the start time, the highlighted norm is assigned to auctioneer (Mari).

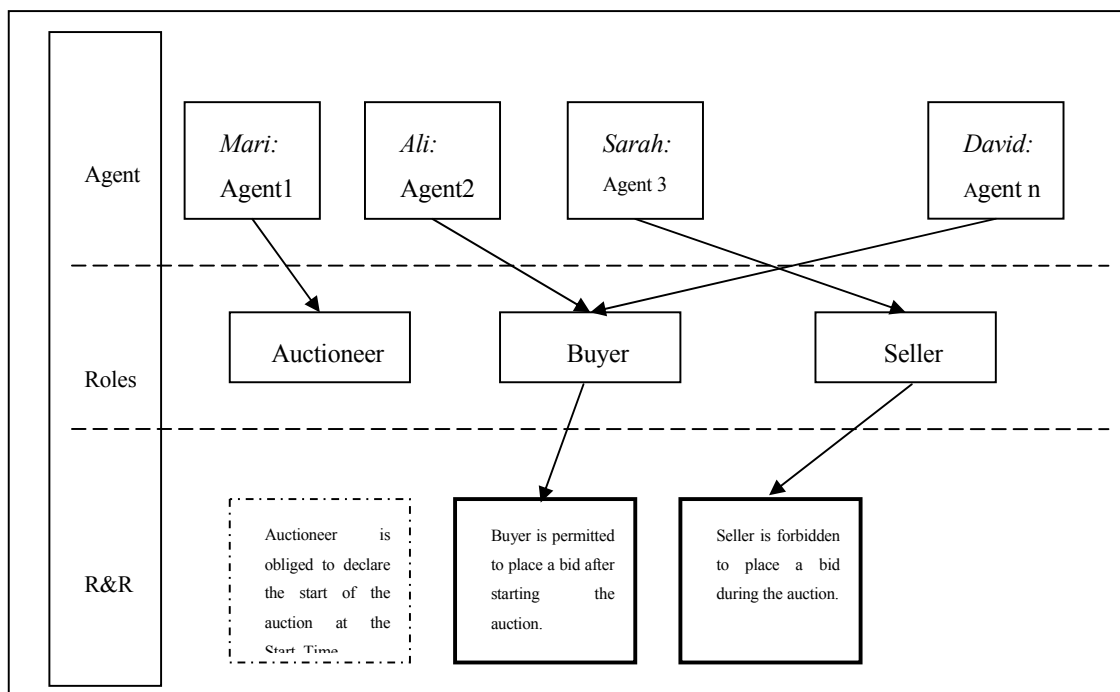


Figure 2-D -Time 4: After start of the auction, two norms for Buyer and Seller are activated.

4.3.3 Dynamic Assignment of Sanctions to Agents

As explained before, dynamic assignment of norms to agents is the objective of this work. Now this section emphasizes that such norms contain enforcement norms as well, which are rules for executing the norms. Thus dynamic assignment of enforcement norms is an important part of this work.

Enforcement norms are essential in any normative system because external agents with autonomy may violate the regulations of the system. Clearly, it is impossible to force external autonomous agents to do any action, and in particular, it is impossible to force an agent to comply with its responsibilities. Thus, a normative MAS needs to provide some responses to the violations of norms by external agents. As mentioned in Section 2.3, punishments, compensations or rewards are some responses to violations or enactment used for enforcing the norms.

However, before executing such responses, the system needs to check the compliance status to identify any cases of norm violation or norm enactment. For instance, for enforcing the punishment norms firstly the violation should be detected, and then as a response, the related sanction should be performed.

Consequently, the designer of the normative multiagent system should anticipate, first, some extra norms called *check norms* for detecting the enforcement cases (such as violation or norm enactment), second, methods for *responses* (punishments for the violations, compensation when a beneficiary claims for and rewards when a specific norm is fulfilled).

Regarding dynamic assignments of enforcement norms, the activation of check norms does not need to be communicated to external agents, as these sorts of norms are part of the internal operation of the system. However the activation or deactivation of response norms is desired to be notified to the external agents.

The responses of normative system are defined by the designer of the normative system or by the legislator. We categorized the responses defined by designer in two cases:

First, ***activating or deactivating some norms***: One way of providing responses against violations is to activate further norms or deactivate some other norms. Activating further norms means adding further obligations, permissions, prohibitions or rights. Deactivating some norms means removing norms that were previously activated.

For example, in the auction system “*David is a Buyer*”. And the following norm has been activated at the start time of the auction:

“David (Buyer) is permitted to place a bid, after Start_Time.”

Suppose that during the auction David violated a norm, and this violation which is detected using the check norms of the system. Then based on the normative system a punishment should be applied to him, such as:

“David is forbidden to place any bid.”

In addition to activation of the above punishment norm, the previous norm (permission of placing a bid) should be deactivated.

As a result, the dynamic assignment of a sanction in this example would be a prohibition norm is assigned to David. As this prohibition is opposite to David's permission for placing bid, a norm deactivation task should also be done.

Second, *doing some internal actions by internal agent*: The other way for providing response is to perform some internal actions. The agent which is responsible to provide appropriate responses for sanctions is called Enforcer. Enforcer has collaboration with internal agents of the normative MAS for executing responses. It forces internal agents to provide internal reactions which are appropriate actions against the violation or enactment.

Those internal actions can be designed by protocol-based norms (described in the next section).

For example, consider an auction system in which “*David is a Buyer*”. Suppose he puts in a wrong bid (e.g. a lower bid) which is a violation. Suppose that the system then detects the violation using check norms with the following enactment being related to this violation case of the auction system:

Enf_Norm1: “If buyer put a wrong bid, his negative feedback will increase.”

Enf_Norm2: “If buyer has three negative feedbacks, it is forbidden for buyer to put any other bid.”

So the following responses are applicable:

Enforcer should force the internal agent to increase David's negative feedbacks and also check the number of negative feedbacks. In fact, this task is an internal task which an environmental parameter is changed by enforcer. Then according to *Enf_Norm2*, the number of David's negative feedbacks should be checked. If it is three or more, then the enforcer should assign the sanction of prohibition for bidding to David.

Therefore, if David has three negative feedbacks, a dynamic assignment of a sanction will be placed to David.

It is necessary to mention that considering norm enforcement in normative system requires adding both check norms and reaction norms into the knowledge base of norms.

4.4 Protocol-based versus Rule-based norms

In a typical MAS based on organization structure, roles have rights and responsibilities which are implemented using protocols. In normative MAS, rights and responsibilities of roles can be considered as norms, but in addition to the protocol-based norms, normative MAS contains rule-based norms as well.

Primarily, norms in Normative MAS can be categorized in two types: *protocol-based* and *rule-based* [30]. Here we emphasize that the type of norms in “*dynamic assignment of rights and responsibilities*” are rule-base norms.

Protocol-based norms are related to all the necessary conventions for agent interactions. This type of norm establishes the permitted actions at each instant of time, considering the past actions of agents.

Protocol-based norms can be applied for both internal and external roles, such that using predefined protocols, internal roles communicate with one another and with external roles. These protocols are statically designed at design time. This fact means that the system designer defines all norms or regulations of agents in the format of protocols at design time. So at runtime agents simply follow the predefined dialogues of protocols, moving from one state to another.

A very well-known examples of such protocol-base norms is the performative structure in Electronic Institutions (EI) [56, 70]. The performative structure of the EI is a network of scenes which agents can move between according to the predefined scene protocols.

Rule-based norms are defined by a certain type of first-order formulae that set up a dependency relation between actions. These norms specify that under certain conditions, new commitments will be produced for agents to do some actions.

Rule-based norms are defined *just for external roles*. This limitation is *because* internal roles implement the protocol-based norms of the system, and do not have any autonomy to deviate from these system norms. The rule-based norms are *statically* defined in the knowledge base of the MAS, but the execution of the rule-based norms for external agents is a *dynamic* task which is executed at runtime. In the knowledge base of R&Rs of external roles, all obligations, permissions, prohibitions and rights of external roles are defined. At runtime, based on the

actions of the external roles and the regulations in the KB, the system detects that the action was acceptable or a violation occurs; if there was a violation, then a sanction should be executed.

As an example, there are some efforts in Electronic Institutions [21, 30, 56] to support rule-based norms, but it is not a complete support. For example, it just supports obligation via using *Governor* agent [2, 23].

To sum up, in the following when we talk about dynamic assignment of rights and responsibilities we specifically mean dynamic assignment of *rule-based norms* to *external agents*. Because internal roles deal with protocol-based norms which are defined statically at design time, dynamic assignment does not make sense for internal agents.

4.5 Summary

After providing the fundamental background to the normative multiagent system in Chapter 1 and Chapter 2, in this chapter we focused on the more detailed aspects related to providing dynamic assignment of R&Rs to agents in normative multiagent systems.

First, in Section 4.2, we explained that a dynamic environment is one of the most important features of an open MAS, so dynamic issues in MAS along with the source of dynamism and changes have been explained. As such dynamism results in more complicated management of the MAS, in Section 4.3 dynamic assignments

were presented as a means to improve the performance of the system. We categorized dynamic assignments in normative MAS as three different types: dynamic assignment of roles to agents; dynamic assignment of rights and responsibilities to agents; and dynamic assignment of sanctions to agents. However, we just focus on dynamic assignment of rights and responsibilities to agents and dynamic assignment of sanctions, since dynamic assignment of roles to agents has previously been proposed in earlier research by others as cited in [64].

Finally, in Section 4.4 a complementary discussion about the difference between protocol-based norms and rule-based norms was presented. As a main difference between protocol-based and rule-based norms at runtime, agents *just follow* the predefined dialogues of protocols while they *decide* whether *to follow the rules*. In fact, agents have autonomy for acting according to the rules or against the rules. In this context, we focused on the rule-based type of rights and responsibilities (R&Rs) of roles.

To sum up, in this chapter we addressed the main theme and direction of this research. Realizing that agents with autonomy to follow or violate the rules in a normative MAS directed us to investigate an approach for executing of rules to impose rights and responsibilities to agent at each instant of time. In the other words, autonomous agents make an unpredictable and dynamic environment such that their rights and responsibilities are influenced by these dynamic activities and environment. Therefore, based on dynamic sources (including the actions of the agent or other environmental events) and the represented norms of the normative system, at each instant of time during system operation specific R&Rs can be allocated to agents. We aim to provide dynamic assignment of rights and responsibilities to agents which not only benefits from improvement in

management of the normative MAS but also would impose the regulation of the normative system and enforce sanctions immediately after occurrences of violations or norm enactments.

Chapter 5

Methods for Dynamic Assignment of R&Rs to External Agents

5.1 Introduction

In the previous chapters, the basic concepts of rights and responsibilities of roles in normative MAS have been explained, followed by a discussion of the sources of dynamism in MAS. We then showed how such sources of dynamism may lead to changes in the rights and/or responsibilities of external agents who play a role in the system. Now, in this chapter, knowing these issues we propose an approach for dynamic assignment of R&Rs and dynamic assignment of sanctions to external agents.

In Section 5.2, we propose two methods for dynamic assignment of R&Rs to external agents and dynamic assignment of sanctions to external agents. Then, in Section 5.3, we explain the common features of both methods, following by the a discussion of their differences in Section 5.4. After that, in Section 5.5, we present the formal representation of these assignment methods, followed by an illustrative application of the formal representation in Section 5.6. Finally, in Section 5.7, a summary of this chapter is provided.

5.2 Methods

The common sense of dynamic assignment of R&Rs to agents has been explained in Section 4.3.2. Recall that dynamic assignments of R&Rs and sanctions to external agents in a normative MAS are founded on two main elements: *a source of R&Rs of roles* and *a source of dynamism*.

The source of R&Rs of roles is a static knowledge base which stores all the norms of roles, including the rights and responsibilities of roles, and also all sanctions. We call such a knowledge base a *Normative Knowledge Base (KB)*. We have already identified (in Section 4.2) the sources of dynamism and changes including diversity in the agent population, the occurrence of actions and of events, changes in environmental parameters, and the achievement of important times. When a source of dynamism causes a change in the MAS we say that a *Runtime Occurrence* takes place.

As the result of a runtime occurrence, one (or more) norm(s) of the normative KB may become satisfied. Therefore, we state that every dynamic assignment of R&R takes place due to occurrence of some source or sources of dynamism at runtime or a runtime occurrence.

Based on the above general issues, we propose two methods for the task of dynamic assignment of R&R and sanctions to external agents. These methods have some similarities, along with differences. In the following sections we explain our methods and the similarities and differences between them.

Before explaining these methods, first we clarify why we have proposed two methods and why we apply this dynamic assignment on external agents. We are chiefly interested in norms which are attached to roles, and the relationships between them. If we had no roles, all rights and responsibilities would have to be expressed using conditional norms. The conditions in these norms would express two things: some would be intended to identify the role of the agent, and some would be intended to identify a particular situation in which the agent playing that role finds itself. For example we might have a rule such as:

An agent which (1) is a member of the institution, (2) has made a bid in a particular auction, (3) has had the bid accepted by the auctioneer, (4) has made the highest bid in that auction, (5) when the auction closes shall pay the sum bid to the auctioneer.

Now this rule could define a situation in which any agent could find itself, a conditional norm that makes no reference to roles. Alternatively we could have a role *member*, defined by condition (1), and conditions (2-5) would define the

situation in which it must pay. Or we could define a role *candidate bidder* by conditions (1-2) and (3-5) would define the situation which activates the norm. Or conditions (1-3) could define a role *bidder* and conditions (4) and (5) the conditions for the norm. Or conditions (1-4) could define *highest bidder* and (5) state when the norm applies. Finally all of conditions (1-5) could define the role *successful bidder*, and the norm would always apply to all agents playing this role.

Thus we have a trade off between how specific we make the roles, and the conditions needed to state when an agent in that role has a right or a duty. The two approaches we describe here represent two different ends of the spectrum: one approach uses quite general roles, requiring extensive use of conditional norms, whereas the other approach uses entirely specific roles, obviating the need for conditional norms. Of course, there can be intermediate positions, as indicated above, but by taking two ends of the spectrum we hope to be able to explore the differences between and the strengths and weakness of the two approaches.

We will discuss when each method should be used in Section 6.7.

With respect to the reason for applying these methods on external agents, we first recall the definition of internal and external agents. Currently, most of the existing implemented MAS such as Electronic Institutions comprise two types of agents, internal and external agents. Internal agents work on behalf of the MAS (which has the central control of the system) to provide facilities, while external agents join the MAS to use these facilities. Internal agents are designed at design time and the predefined protocols of the MAS can provide the regularity structure for them to follow the regulations of the MAS. However, external agents join at runtime and their behaviors are unexpected at design time; therefore these agents can decide to

follow or violate the regulations of the system. Internal agents have no freedom of choice regarding the system norms, while external agents do.

On the one hand, we aim to provide the implementation of this dynamic assignment to those agents who have autonomy to follow or violate the norms; on the other hand, the internal agents of the existing implemented MASs do not have autonomy to violate the norms. Consequently, we propose our methods for applying to the external agents, in order to consider autonomy for agents to decide follow or violate the norms. For example, in an electronic auction, the auction manager who is an internal agent follows the regulations of the system as defined at design time. But Mike as the external agent joins to the system at runtime and plays the role of buyer, this agent is autonomous to follow or violate the norms at runtime.

5.2.1 Method 1- Using Role Hierarchies

Our first method uses role hierarchies. This method is significantly based on the concept of role such that roles have a detailed hierarchy structure and several sub-roles. Each role is assumed to be a sub-role of another role, except for those roles at the highest level. The rights and responsibilities of roles are also defined more specifically for the sub-roles. As an example, the following role hierarchy can be illustrated for an auction system (Figure 3):

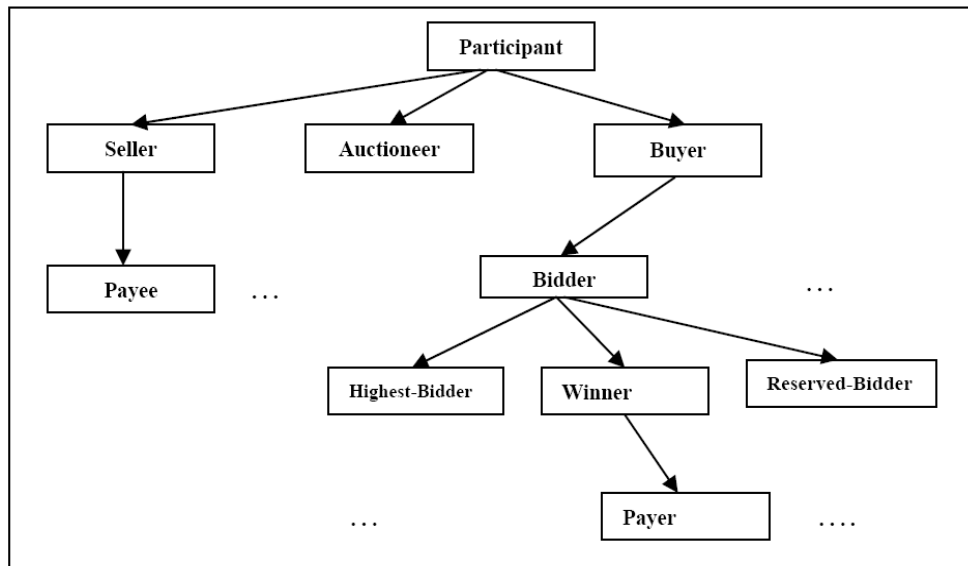


Figure 3-The role hierarchy for the auction system

Figure 3 shows the roles of external agents in an auction system structured in a hierarchy such that for every main role there are some sub-roles. For instance, *Buyer* is a main role which has sub-roles including *Bidder*, *Highest-Bidder*, *Winner* and *Reserved-Bidder*.

Each sub-role has a specific set of rights and responsibilities. For example, “*Winner is obliged to pay the price of the item.*” -- this obligation is just for the *Winner*, and no other roles have this obligation.

All R&Rs of roles and sub-roles are stored in the normative KB, as described in the last section. The normative KB of this method contains a set of constraints which note the corresponding normative commands of each specific role or sub-role. For example:

Norm 1: "Buyer is permitted to place a bid before ending time of the auction."

Norm 2: "Auctioneer is obliged to evaluate bids within 1 minute of placing bid."

Norm 3: "Winner is obliged to pay the price of the item within 1 hour of ending the auction."

Norm 4: "Payer has the right to get the item within 1 day of his payment."

In this method which is based on role hierarchy, the roles of agents will frequently change at runtime. After any runtime occurrences (such as occurrence of an action or environmental event), the role of external agents may change, typically by agents in a given role being assigned a more specific sub-role.

For example, in the beginning of the auction session, *Mari* logs in as a *Buyer*. Next, if she places a bid, her role will be changed to *Bidder*. Here placing a bid is an action changing the role of *Mari* from *Buyer* to *Bidder*. If there is no other *Bidder* in the auction round or her bid is the highest bid, the role of *Mari* is *Highest-Bidder*. However, once another higher bid is placed (more than *Mari*'s bid) *Mari* is no longer a *Highest-Bidder*. This change of role is because of the event in which some other person placed a bid.

So in this method the roles of agents change frequently. Such changes of roles are dynamic tasks and will affect the rights and responsibilities of agents. As a result, dynamic assignment of R&Rs and sanctions to agents using role hierarchies has the following steps:

A design-time task: The designer of the system provides the role hierarchy of the system.

A design-time task: For each sub-role, the specific set of rights and responsibilities are defined in the normative knowledge base. The knowledge base contains all rights and responsibilities of roles (including obligations, permissions, prohibitions or rights) and sanctions of roles (including punishments, rewards and compensations).

A runtime task: Dynamic assignment of sub-roles to agents occurs in response to system events. After any runtime occurrences, system detects whether the sub-role of agent changes or not (based on the agent's recent actions and the other environmental events).

A runtime task: Dynamic assignment of R&Rs of sub-roles to agent is the last phase of this method. This happens whenever the role of an agent changes, because each role independently has a separate set of rights and responsibilities. Therefore, in the case of any change in the role of an agent, the R&Rs of the recent role of the agent are assigned to that agent. In Chapter 5, we will explain this process from the implementation viewpoint in detail.

For dynamic assignment of sub-roles to agents, there need to be some regulations in the MAS which specify how and when the dynamic assignments of sub-roles to agents should be provided. Such regulations can be implemented by protocols instead of using rules for two reasons: first, sub-roles are internal concepts provided for the improvement of the system; second, we do not allow for any autonomy for external agents for selecting a sub-role for themselves, instead their roles are determined by their actions and the context, and so the roles of agents can

be assigned by the system. So the designer of the system can define protocols for dynamic assignment of sub-roles to agents at design time.

The following figure (Figure 4) is an example of state diagram and transitions of a protocol for the role hierarchy of auction system.

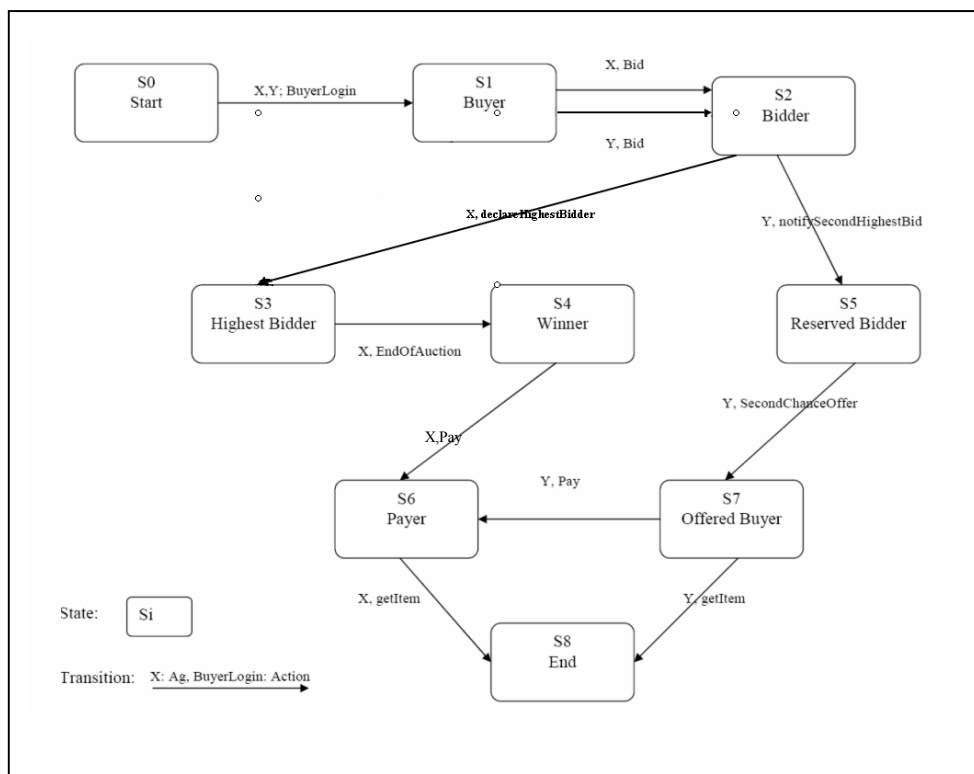


Figure 4 -An example of state/transition model for auction

In the last step, the assignment of R&Rs of sub-roles to agents is dynamically performed based on the rule-based norms defined in the normative KB. These norms consist of some obligations, permissions, prohibitions or rights. Assignment of these various modalities to external agents imposes some commitments for

them. So implementation of such norms is not provided by protocols, but by rules. All agents remain free to comply with or violate the norms associated with their role.

We have already described the differences of protocol-based and rule-based norms in Section 3.4. Recall that protocol-based norms are defined such that they anticipate all runtime events in design time and agents should follow the protocols. Anticipating all possible runtime actions of autonomous external agents at design time is a very demanding task (because they may violate or do not act in accordance the law). Therefore, in the last step, instead of using protocol-based norms, we use rule-based norms which specify that at runtime under certain conditions, new commitments will be produced for agents to do some actions.

For example, if Ali is logging into the auction system and he chooses to purchase something, then based on the protocols of the system the role of *Buyer* will be assigned to him (Step 1). Then based on the rule-based norms of the system, all R&Rs of *Buyers* will be assigned to Ali (Step 2).

5.2.2 Method 2: Using Conditional Norms

The second method is based on conditional norms. Conditional norms are defined in the static normative knowledge base. The knowledge base contains all rights and responsibilities of roles (including obligations, permissions, prohibitions or rights) and sanctions of roles (including punishments, rewards and compensations) also these norms (R&Rs of roles) are conditional. Therefore, the knowledge base is rule-based and contains statements of rules such as: *if A then B* ($A \rightarrow B$) where *A* is a set of pre-conditions and *B* is normative command(s).

Instead of defining several sub-roles as defined in Method 1, only the main roles are defined in this method. In Method 2, roles are more generic. For example, in the auction system the main roles of the system can be defined as *Seller*, *Buyer* and *Auctioneer*. So in this method, there are no sub-roles such as *Winner* (unlike Method 1) and the relevant R&Rs of *Winner* is defined for the generic role of *Buyer*. In a sense, the condition for a norm corresponds to the condition for being allocated a sub-role associated with that norm. For example:

“If buyer has the highest bid and the auction has been ended, then he is obliged to pay the price of the item.”

This method uses conditional norms. The condition(s) of norms shows the actions or events or status that are preconditions for the activation of a norm. So whenever runtime occurrences cause the satisfaction of the pre-conditions of a norm of a role, the corresponding norm will be fired and assigned to any agent which plays the role.

For dynamic assignment of R&Rs and sanctions to agents using conditional norms, the following steps should be implemented:

A design time task: the normative knowledge base should be defined by the system designer. This knowledge base contains all R&Rs and sanctions of roles, and all of these norms are conditional. Again we mention that roles in this method are very generic.

A runtime task: The system assigns a main role to each agent immediately after joining the system, according to the initial action of the agent (e.g., at the beginning, an agent may select to be a buyer). In this method, the number of

roles is limited and there are only a few specific actions that may lead to change the role of the agent. If an agent executes any of these actions which may lead to a change the role, the system assigns the new role to the agent.

A runtime task: Immediately after occurrence of any action or environmental event, if the condition(s) of a norm (related to a role) is satisfied, then the specific right(s) or responsibility(ies) of that role is/are assigned to the agent who plays the role. In Chapter 6, we will explain this process from the implementation viewpoint in detail.

5.3 Common Features of Methods

Here we mention the common features of the two proposed methods, because identifying these common features will help us to define the formal representation for our methods and subsequently assist with analysis, design and implementation of the methods.

The common features are as follows:

1. Both methods rely on the concept of role. In both methods, each external agent has a role at every moment of its lifetime. Roles are assigned to external agents by the administration of the system. The assignment of roles to agents is a dynamic task and is based on the actions that agents perform or on other runtime occurrences.

Here we recall from Section 4.3.1 notice that the method of dynamic assignment of roles to agents has already been proposed by other researchers [64]. Thus, we will not focus on this activity, and simply follow our methods for dynamic assignment of R&Rs and sanctions to external agents.

2. Both methods include a normative knowledge base. Such a knowledge base contains all the main norms and enforcement norms of the normative system, which we have called in this context *rights & responsibilities* and *sanctions*, respectively. In other words, this knowledge base contains all obligations, prohibitions, permissions and rights of roles as norms, and also all punishments, rewards and compensations as enforcement norms.

Therefore, this knowledge base is the main resource of norms in the normative MAS which are predefined by the legislator of the system and expected to be enforced by our methods. For the definition of the norms of the normative KB, a descriptive normative language is used, as mentioned in Section 3.5. We will discuss this issue in more detail in Section 5.5.10.

This knowledge base is static, as we consider there are no changes to its contents at runtime.

3. In both methods, the dynamic sources which may lead to dynamic assignment of R&Rs or dynamic assignment of sanctions are the same.

5.4 The Differences of the two Methods

Here we explain two main differences of our proposed methods: first the differences of them in the definition of roles; second, the differences of them in creating the normative KB.

1. In Method 1, the system uses a hierarchy of roles and several sub-roles, all of which must be pre-defined by designer at design time. After every runtime occurrence, the role of an agent may be changed. Subsequently, changing the role of the agent causes new R&Rs or new sanctions corresponding to be assigned to that agent, according with the agent's new role or sub-role. These sub-roles and the transitions to and from them are specified by the system designer using protocols.

However, in Method 2, the number of defined roles in the system is limited, and so the role of an agent is rarely changed, just by occurrence of a few predefined runtime occurrences. This method does not apply protocols and is just based on conditional norms defined in normative KB.

2. In Method 1, norms in normative KB have the format of a single constraint stating a commitment of a role, while in Method 2 norms are rule-based with the format of $LHS \rightarrow RHS$, where LHS is the condition(s) of the norms and RHS is the commitment. Therefore, after any runtime occurrence if the LHS of a norm is satisfied for an agent, then RHS will be executed, this RHS is a commitment (including R&Rs or sanctions) for the corresponding role of that agent.

5.5 Formal Representation

After describing the methods, we now provide a formal representation for dynamic assignment of rights and responsibilities to agents which is the foundation of the subsequent analysis, design and development of our proposed dynamic assignment methods. This formal representation provides a clear and precise description of what our implementation is supposed to do. For the definition of our formal representation, we use basic set theory, logic and function theory.

In this section, first of all, we list the assumptions. The initial assumptions of this formal representation are as follows:

Time: Time is assumed to be discrete, and infinite into a single (i.e., non-branching) future. We represent time points by the symbols of t , z , and u , and denote the set of these time points by $Time = \{ t, z, u, \dots \}$

Environmental Variables: There are many environmental variables in every MAS. These variables are propositions that describe the state of the environment. For example, *price* is a variable in auction system. We represent environmental variables by the symbols of v and w , and denote the set of environmental variables by $V = \{ v, w, \dots \}$.

Next, we define the basic concepts and notions of our formal representation, including agents, roles, runtime occurrences, and normative knowledge base. Then we define the main function of dynamic assignment of R&Rs and sanctions to agents, by formally representing all the components of this function.

5.5.1 Agents

Agents were described in Section 2.2.1. We give the formal definition of agents as follows: Let $\mathcal{A}g = \{a_1, a_2, \dots, a_n\}$ represent identifiers for a finite set of n agents. This set of agents includes both external agents (e.g., $Ali \in \mathcal{A}g$) and internal agents (e.g., $Enforcer \in \mathcal{A}g$).

Note that $\mathcal{A}g$ is a finite set. Because we are dealing with open multiagent systems, agents may join or leave the system at any time. For the purposes of our representation, we assume that all agents which join the system, are included in the set $\mathcal{A}g$. In other words, the size n of $\mathcal{A}g$ is sufficiently large to incorporate all the agents which may enter the system in any run.

5.5.2 Roles

We have identified the concept of role as an important notion in multiagent systems (in Section 2.3.1). Roles allow abstraction from the individuals or agents in a MAS. Considering the concept of role in MASs, it is obligatory for an agent to adopt at least one role in order to join the system. Thereafter, the behavior of an agent playing a given role should be in accordance to the R&Rs corresponding to that particular role. In addition, if an agent violates any norm applicable to the role it is playing, then the sanctions corresponding to that violation for particular role should also be enforced.

As the environment of the system is dynamic and agents may play several roles in their lifetime, the assignment of roles to agents is a dynamic task too, as described in Section 4.2.1. In this work, we have assumed that the MAS for which our methods are intended has the capability of providing dynamic assignment of roles to agents. Then, the result of any new assignment of role to agent will be used in our methods.

For formal definitions, we denote the roles and the function that assigns roles to agents, as follows:

$\mathcal{R} = \{r_1, r_2, \dots, r_m\}$ represents a finite set of m role identifiers.

$CR: \mathcal{A}g \times Time \rightarrow \mathcal{R}$, where $CR(a_i, t)$ is a function which assigns roles to agents at time t , for each agent a_i , such that $a_i \in \mathcal{A}g$. This function varies by time, because the assignment of roles to agents is dynamic and different roles can be assigned to agent at different times during runtime.

For example, suppose that *Ali* is an agent and a member of $\mathcal{A}g$ ($Ali \in \mathcal{A}g$), *Auctioneer* is a role and a member of \mathcal{R} ($Auctioneer \in \mathcal{R}$). If *Ali* at time t_1 has the role of *Auctioneer*, then $CR(Ali, t_1) = \{Auctioneer\}$.

Because our methods do not assign roles to agents dynamically (but rather R&Rs), we have not considered the dynamic assignment of roles to agents in the representation below. However, it would not be difficult to do this, using methods and notations similar to what we use in this chapter.

5.5.3 Actions of Agents

We defined the concept of action in Section 2.2.1. In addition, we described action of agents as a source of dynamism in Section 4.2.1. Here, we denote the notation of *Act* for a single action. This notation is used for the definition of actions which are commanded or permitted by specific roles. This notation of a single action is not the real action of agents which happens in runtime, but is an abstract representation of an action.

We denote a notation for *the set of actions of agents* as follows:

\mathcal{AC} is a set of actions which agents are capable of executing.

For example, “*paying the price of the item*” is an action of the agent playing the role of *payer*. This action is a member of \mathcal{AC} . “*Pay*” is *Act*, which is provided or commanded to provide by the role *payer*.

The details of the definition of \mathcal{AC} are dependent on the multiagent system intended to apply our methods for dynamic assignment of R&Rs and sanctions to agents. This is because different MASs have different domains and environments. Consequently, their parameters are different as well. Therefore, here we do not expand this notion, but leave it abstract.

5.5.4 Environmental Events

We defined the concept of event in Section 2.2.1. In addition we described environmental events of agents as a source of dynamism in Section 4.2.1. Recall that, there are different types of environmental events. First, some of these events are related to **the actions of other agents**. For example, if *Payer* pays the payment, then “*receiving the payment*” is an event from the *Payee* viewpoint. In other words, when the agent of *Payer* does the **action** of paying, this action is the **event** of receiving the payment from the payee viewpoint. (We suppose that the system works perfectly and there is not any fault in the system. So when the action of payment occurs, the event of receiving money also occurs. In other words, we assume determinancy of actions)

In this thesis, we distinguish the actions which an agent executes by itself and the events that come from the environment as the result of the actions of other agents. In this way, we emphasize that assignment of R&Rs and sanctions to agents occurs not only because of their own actions, but also because of actions of the other agents.

The second type of environmental events occurs when **environmental variables change**. In every MAS there are some environmental variables whose changes in values may influence the dynamic assignment of R&Rs and sanctions to agents. For example, in an auction system there may be an environmental variable called Negative Feedbacks. If the value of this variable changes, an environmental event occurs. For example, “*increasing the negative feedbacks of David (a buyer)*” is an example of this type of environmental event.

The third type of environmental events is due to **time alerts**. Certain specific and important times at runtime may affect the dynamic assignment of R&Rs and sanctions to agents. As an example, the ending time of auction is an important time such that reaching this time leads to the assignment of several R&Rs or sanctions to agents. For this reason, we provide a time stamp mechanism in order to notify these important times as an environmental event in the system.

We note that although in our earlier categorization of dynamic resources we considered the entry and departure of the agents as something distinct from events, here we consider this source of dynamism in the event category. When an agent joins to the system or leaves the system, an event happens to the system.

We denote the environmental events as follows:

\mathcal{EV} is a finite set of environmental events.

The details of the definition of \mathcal{EV} are dependent on the multiagent system intended to apply our methods for dynamic assignment of R&Rs and sanctions to agents. This is because different MASs have diverse domains and environments. Thus, their parameters are different as well. So, here we do not expand this notion, but as with the actions of agents, we leave it abstract.

5.5.5 Runtime Occurrences

As described in Section 4.2, runtime occurrences include changes to the population of agents, the actions of agents and environmental events including the actions of other agents (which we simply call events), environmental parameter changes, and achievement of important times. These sources of runtime dynamism in a MAS may influence the assignment of rights and responsibilities to agents. Indeed, we assume that any alteration in R&Rs of any agent of the system is due to a runtime occurrence, and only this.

In our formal representation of dynamic assignment of R&Rs to agents, runtime occurrences are very important parameters. Therefore, whenever a runtime occurrence happens, it should be detected and recorded. This means that after occurrence of any action, event, parameter change, or achievement of an important time (such as a deadline), such a new change should be detected and recorded.

We denote a runtime occurrence as a finite set \mathcal{RO} such that

$$\mathcal{RO} = \{o \mid o \in \mathcal{AC} \cup \mathcal{EV}\}$$

where \mathcal{AC} is the set of actions and \mathcal{EV} is the set of environmental events.

We can also define the runtime occurrences that have occurred up to time t , as follows:

$$\mathcal{F}\text{-}\mathcal{RO}: \text{Time} \rightarrow \mathcal{RO}$$

or $F_{\mathcal{RO}}(t) = ro; \quad t \in \mathit{Time}, \quad ro \in \mathcal{RO}$

where $F_{\mathcal{RO}}$ is a function gives the runtime occurrence happened at time t .

For example, suppose that “10:00” is a member of Time (“10:00” $\in \mathit{Time}$) and “*Auction started*” is a runtime occurrence and member of \mathcal{RO} (“*Auction started*” $\in \mathcal{RO}$). Therefore, $F_{\mathcal{RO}}(10:00) = \text{“*Auction started*”}$ states that “*the runtime occurrence of time 10:00 is “Auction started”*”

5.5.6 Formal Syntax of Norms and Enforcement Norms

In Section 3.5, we proposed a normative language in which the following notions are considered in the normative languages:

- deontic concept: obligation, prohibition, permission and rights
- sanction concepts: punishment, compensation or rewards
- temporal aspects : Before (t), After(t), Between(t1,t2), At(t)

Based on the descriptive normative language of Section 3.5, now we define the following syntax as a formal representation for norms and enforcement norms in our methods.

Norm: To show norms with legal modalities we use the following syntax:

$$Norm = DeoMode (Act, r, V, Tmp(t,u))$$

where,

DeoMode is one of the four deontic modalities: obligation; permission; right; and forbidden. Thus, we denote *Deo Mode* as: $Deo Mode \in \{Obl, Prm, Right, Frb\}$,

Act is an action referenced by role *r*. In this definition, *Act* is the abstract denotation of the action, not the real action in runtime.

r is a role, $r \in \mathcal{R}$,

$V = \{v, w, \dots\}$ is a set of other environmental variables (such as the name of the auction or the amount of current bid), and

Tmp(t,u) is the value of a temporal function, where:

$$Tmp(t, t) = before(t) / after(t) / at(t) \quad ; \text{ if } t = u$$

or

$$Tmp(t, u) = between(t, u) \quad ; \text{ if } t < u$$

Example: As an example of the normative part of the rules:

“Obl (pay, Buyer, price, before (Te + 6 hrs))” states *“Buyer is obliged to pay the price in 6 hours from ending time of auction(Te)”*.

Enforcement Norm: To show enforcement norms we use the following syntax:

$$EnfNorm = EnfMode (r', ECode)$$

where,

EnfMode is one of the three deontic modalities: punishment; compensation; reward. Thus so we denote *EnfMode* as: $EnfMode \in \{Pnsh, Cmp, Rwr\}$,

r' is a role, $r' \in \mathcal{R}$, and

ECode is an Enforcement Code number ($ECode \in \mathbb{N}$) and refers to the sanction, compensation or reward that should be applied on the role. This Enforcement Code is used by internal agents of the MAS which are responsible for enforcing the norms. These codes are defined by designer of the system at design time. Therefore, if an internal agent (e.g., Enforcer) receives an enforcement code at runtime, it identifies the enforcement code and will act based on the instruction of the code.

For example:

“Pnsh(Buyer, 10)” states “Buyer is punished by the punish-code # 10.”

5.5.7 Formal Syntax of Normative Commands

In this section, we propose a notation for normative commands. For representing normative commands, we use the notation of *norm* and *enforcement norm*. However, first, we describe the notation of normative commands. There are two types of normative commands: starting with *command mode* and starting with *execute mode*, as follows:

- **Command Mode:** We use command modes to represent the status of the norm. These modes would be useful for implementation of the norms. We define five types of command mode as follows:

ToBeActivated: represents that the norm should be activated at a future time.

Activated: indicates that the norm is activated.

Deactivated: indicates that the activated norm is deactivated.

Fulfilled: indicates that the activated norm is fulfilled.

Violated: indicates that the activated norm is violated.

So we denote command mode as follows:

$$\text{CommandMode} \in \{\textit{ToBeActivated}, \textit{Activated}, \textit{Deactivated}, \textit{Fulfilled}, \textit{Violated}\}.$$

- **Execute Mode:** We use an execution mode in our formal representation in order to represent enforcement command for internal agents to run the enforcement norm. We denote the syntax of execute mode as follows:
 $ExecuteMode \in \{Execute\}$.

Identifying command mode and execute mode in our formal definition, we denote Normative Commands as follows:

$$\mathcal{NCm} = CommandMode (Norm) \vee ExecuteMode (EnfNorm)$$

Where

\mathcal{NCm} is Normative Commands Mode,

$CommandMode \in \{ToBeActivated, Activated, Deactivated, Fulfilled, Violated\}$,

$Norm = DeoMode (Act, r, v, Tmp(t,u))$,

$ExecuteMode \in \{Execute\}$, and

$EnfNorm = EnfMode (r', ECode)$.

For illustration, we present the following examples for normative commands:

Example 1: If the normative command said that “*It is activated that Auctioneer is obliged to declare the start of Auction_1 at 11:00.*”, then its formal representation would be as follows:

Activated (Obl (declareStart, Auctioneer, Auction_1, At (11:00)))

Example 2: If the normative command said that “*Punish-code #10 should be executed for Buyer.*”, then its formal representation would be as follows:

Execute (Pnsh(Buyer,10))”

5.5.8 Formal Syntax of Conditions of Norms

Some norms are conditional and rule-based. In conditional norms, if the conditions of norms are satisfied the norm will be fired. We define the condition of a norm as a conjunction of one or more constraints.

We detect that the conditions of norms - which may occur at runtime and lead to the satisfaction of a norm - can be of the following types:

1. **SGE** refers to system generated events (SGE). In runtime, if the MAS generates a SGE and records this SGE as a runtime occurrence, then the constraint of the norm which has this SGE as a condition will be satisfied. Each *SGE* is a tuple with the following syntax:

$$SGE = gEvent(Act, r, V, At(t))$$

where

Act is the abstract definition of act provided by the r role. Here we emphasize that Act is different from \mathcal{AC} (defined in Section 5.5.3). \mathcal{AC} is a set of actions which really occur in runtime of the MAS but Act denotes the abstract definition of an action in the definition of the condition of a norm.

r is a role, $r \in \mathcal{R}$

$V=\{v, w...\}$ is a set of environmental variables. For example, $price$ is a variable in an auction system.

and $At(t)$ denotes that the SGE occurs at time t .

For example:

“ $gEvent (pay, buyer, price, At(t_x))$ ” means “ *This event happened: buyer pays the price at t_x .* ” In this example, the action is pay , the role is $buyer$, the variable is $price$ and $At(t_x)$ indicates that the payment took place at time t_x .

The notation of “ $gEvent$ ” indicates that this tuple is related to a system generated event.

We note that the details (e.g. number of parameters) of the definition of system defined events (SGE) are dependent on the MAS. However, here we just consider common parameters which every system generated event (SGE) may have.

2. **IS** indicates that the value of variable of x is y . The appearance of this pair in the condition of a norm interprets as “if x has the value of y, \dots ”. For example:

“ $IS(x, Buyer)$ ” states that the value of x is $Buyer$. So as a condition constraint it means that “if x is a $Buyer, \dots$ ”.

3. **MT** is a constraint of simple arithmetic calculation or comparison. For example, $(m > 3)$ or $(y+1)$.

After identifying the parameters of SGE , IS and MT , we denote conditions of norms by the notation Cnd . Here, we define the syntax of conditions for conditional norms. A condition of a norm is a *conjunction of one or more constraints*. Each of these constraints denotes a single state of the system. So at runtime, when all of constraints are true, then the condition(s) of the norm is(are) satisfied, and subsequently the commitment of the norm is fired.

$$Cnd = C_1 \wedge \dots \wedge C_n, \quad C_i := SGE \mid IS \mid MT; \quad 1 \leq i \leq k; \quad k \in \mathbb{N}$$

where,

SGE is a system generated events. $SGE = gEvent(Act, r, V, At(t))$,

IS is a pair of (x,y) represents that the value of x is y , and

MT is a constraint of simple arithmetic calculation or comparison.

Example: Now, we illustrate an example of conditions of a norm. For example, suppose that the condition of a norm is: “*If a Member is logged into the auction and her/his Negative Feedback is less than 3, then ...*”. The condition of this norm is the conjunction of multiple constraints. In the following we first formulate each constraint and determine the data type of the constraint (*SGE* , *IS* or *MT*), then present the whole condition:

$C_1 = (x, \text{Member})$ $C_1 := IS$, it shows that if x is a *Member*

$C_2 = (a, \text{Auction})$ $C_2 := IS$, it shows that if a is an *Auction*

$C_3 = (NF(x) < 3)$ $C_3 := MT$, it shows that if the value of function $NF(x)$ is less than 3. Note, we suppose that $NF(x)$ is a predefined function in our system which takes the name of the agent and then results the number of its negative feedbacks.

$C_4 = (gEvent(login, x, a, At(t_1)))$ $C_4 := SGE$, this condition shows that if x logs in to a , at time t_1 . Recall that the time of occurrence of all events is recorded in our system.

After defining C_1 , C_2 , C_3 and C_4 , in the following we define the conjunction of them:

$$Cnd = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

$$= (x, \text{Member}) \wedge (a, \text{Auction}) \wedge (NF(x) < 3) \wedge (gEvent(login, x, a, At(t_1)))$$

5.5.9 Normative Rules

In Method 2, which is based on conditional norms, norms are defined as a rule (\mathcal{RL}) composed of two parts (conditions and normative Commands). \mathcal{RL} has the following notation:

\mathcal{RL} : $Cnd \Rightarrow NCm$ where Cnd states conditions and NCm states Normative Commands.

In the above sections, we defined the notations of conditions (Cnd) and normative commands (NCm). The notation of condition is only used in Method 2, while the notation of normative commands will be used in both Method 1 and Method 2.

Example: Now we present an example of a conditional norm. Suppose that in the auction domain a norm says that:

Norm 3: *“If a Buyer places a bid, the Auctioneer is obliged to evaluate the Buyer’s bid within 1 minute”.*

Based on our formalism, this norm would be represented as follows:

N 3: $(x, Buyer) \wedge (y, Auctioneer) \wedge (b, Bid) \wedge (gEvent(placeBid, x, b, At(tx)))$
 $\Rightarrow Activated(Obl (evaluateBid, y, b, between(tx, tx + 00:01))$

5.5.10 Normative Knowledge Base

The normative knowledge base is very important in the process of dynamic assignment of R&Rs and sanctions to external agents, because it is the sole source of norms of the normative MAS. In Section 3.3, we explained fundamental normative issues including norm categorizations, the key elements of the norms, and the necessity of norm enforcement. Particularly, in Section 3.5, we described how norms can be represented by a descriptive normative language.

Using that descriptive normative language all norms (or in this thesis, R&Rs and sanctions) of roles can be formally defined. Then norms should be stored in a knowledge base as a normative resource. Such a normative knowledge base is the other central parameter in the function of dynamic assignment of R&Rs and sanctions to agents.

We denote the knowledge base by $\mathcal{NK}\mathcal{B}$. $\mathcal{NK}\mathcal{B}$ contains a set of norms or (normative rules) of the normative system. In our context, the main elements of the $\mathcal{NK}\mathcal{B}$ are norms as formulated in Section 5.5.9.

Although both proposed methods contain a normative KB, they have different notations for defining norms in each case. In Method 1, which is based on a role hierarchy, norms are defined as a single constraint which is a normative command. In Method 2, which is based on conditional norms, norms are defined as a rule (\mathcal{RL}) composed of two parts (conditions and normative Commands).

It is good to mention two points here: first, we assume that the normative knowledge base is static during the runtime of the system. Second, the definition of

norms in the normative knowledge base indicates the rights and responsibilities (R&Rs) of roles. Indeed, the definition of these norms at design time specifies when R&Rs are assigned to roles. In runtime, the task of lookup R&Rs of roles is done by inference engine. However, we define a function denoted $F_{\mathcal{NK}\mathcal{B}}$, which produces the set of normative rules associated with role r_j , as follows:

$$F_{\mathcal{NK}\mathcal{B}}: \mathcal{R} \rightarrow \mathbb{P}(\mathcal{NK}\mathcal{B}) \quad , \quad F_{\mathcal{NK}\mathcal{B}}(r_j) \subseteq \mathbb{P}(\mathcal{NK}\mathcal{B}) ; r_j \in \mathcal{R}$$

where $\mathbb{P}(\mathcal{NK}\mathcal{B})$ is the power set of $\mathcal{NK}\mathcal{B}$, i.e. the set of all subsets of $\mathcal{NK}\mathcal{B}$.

5.5.11 The Set of Current Activated R&Rs of agents

In the process of dynamic assignment of R&Rs and sanctions to agents, the set of *current* activated R&Rs and sanctions of agents is the other main factor which may affect on a new assignment. For example, suppose that Ali is the winner of the auction and previously this norm has been assigned to him: “*It is activated that Ali (winner) is obliged to pay the price of the item before (Te+1hour)*”. If the runtime occurrence alerts the deadline of the payment (“*The current time is (Te+1hour)*”), then our system detects a violation and a punishment norm will be assigned to Ali. Therefore, the current set of activated R&Rs and sanctions keeps a history of assignment and may be effective for next assignment of R&Rs or sanctions.

Therefore, we define the symbol of \mathcal{S} for the set of all R&Rs and sanctions of the existing external agents of the system which have been activated and *are still in force at the current time*. The data type of the members of \mathcal{S} , are of norm or enforcement norm type which we denoted by the following notations:

$$Norm = DeoMode (Act, r, V, Tmp(t,u))$$

$$EnfNorm = EnfMode (r', ECode)$$

We define the function of \mathcal{F}_S to produce all the R&Rs and sanctions applied to ai which have been activated by, and are still active at time t .

$$\mathcal{F}_S: \mathcal{Ag} \times \mathcal{Time} \rightarrow \mathcal{S}, \quad \mathcal{F}_S(ai, t) \subseteq \mathcal{S}; \quad ai \in \mathcal{Ag}, \quad t \in \mathcal{Time}$$

Example: Suppose that Ali is an agent ($Ali \in \mathcal{Ag}$) and “11:05” is a time (“11:05” $\in \mathcal{Time}$). The following statement:

$$\mathcal{F}_S(Ali, “11:05”) = \{ Obl (declareEnd, Ali, Auction_1, At(11:00)) \}$$

shows that the assigned norm have been activated to Ali by time “11:05” is “*Obl (declareEnd, Ali, Auction_1, At(11:00))*”.

5.5.12 The Instantaneous Rights/ Responsibilities of Agent

In Section 4.3.2, we explained that when a role is assigned to an agent at runtime, generally all the rights and responsibilities related to that role are allocated to that agent. However, whenever a runtime occurrence takes place, one or more right(s),

responsibility(s) or sanction(s) of that specific role might be activated and assigned to the agent playing that role.

We let $\mathcal{RR}(ai, t)$ denote the instant rights and responsibilities of agent ai which is the set of instant rights and responsibilities and sanctions assigned to an agent ai at time t . The data type of the members of $\mathcal{RR}(ai, t)$ are of normative command type (command mode or execute mode) which we denoted in Section 5.5.7.

We emphasize that in this section we have defined the *symbol* of $\mathcal{RR}(ai, t)$ for the instant R&Rs of an agent at a specific time, but in the next section we will define the *function* that assigns such instant R&Rs and sanctions to agents. In other words, the symbol $\mathcal{RR}(ai, t)$ denotes the set of rights and responsibilities of agent ai at time t , and this set is the output of the function λ defined in the next section, when λ is evaluated with the inputs agent ai and time t .

Example: Suppose that David is an agent ($David \in \mathcal{Ag}$) and “11:00” is a time ($“11:00” \in \mathcal{Time}$). The following statement:

$$\mathcal{RR}(David, “11:00”) = \{Activated (Prm(selectBuyer, David, Auction_1, before(11:00)))\}$$

Shows that the set of instant R&Rs assigned to David at time 11:00 is $\{Activated (Prm(selectBuyer, David, Auction_1, before(11:00)))\}$.

5.5.13 The Function of Assignment of R&Rs and Sanctions to Agents

Here we define a *function* for assignment of R&Rs and sanctions to agents. This function takes various parameters as inputs and then outputs the assignment of R&Rs and sanctions to an agent.

We have already defined all the input parameters of the function including agents ($\mathcal{A}g$), the function of current roles ($\mathcal{C}\mathcal{R}$), runtime occurrences ($\mathcal{R}\mathcal{O}$), the normative knowledge base ($\mathcal{N}\mathcal{K}\mathcal{B}$), the set of current R&Rs and sanctions of agents (\mathcal{S}) and the instantaneous rights and responsibility of agents ($\mathcal{R}\mathcal{R}$). Inputting these parameters the function outputs the instantaneous rights, responsibilities or sanctions of an agent (mentioned in the $\mathcal{A}g$ parameter of the function at a particular point). In fact, this function uses the mentioned inputs to assign the R&Rs and sanctions of the role (currently agent plays) - defined in the normative knowledge base- to the agent based on the given runtime occurrence which is a parameter of the function.

We define a function λ which assigns rights and responsibilities (R&Rs) and sanctions to agent dynamically, as follows:

λ takes as inputs a time point $t \in \mathcal{T}ime$ ($\mathcal{T}ime$ is the set of time points) and an agent identifier, $a_i \in \mathcal{A}g$ ($\mathcal{A}g$ is the set of possible agents).

λ produces as output the rights, responsibilities and sanctions assigned by the MAS to agent a_i at time t . In other words,

$$\lambda : \mathcal{A}g \times \mathcal{T}ime \rightarrow \mathbb{P}(\mathcal{R}\mathcal{R})$$

where $\mathbb{P}(\mathcal{RR})$ is the power set of \mathcal{RR} (ie, the set of all subsets of \mathcal{RR}).

$\lambda(a_i, t)$, the value of λ for a_i at time t , is $\mathcal{RR}(a_i, t)$ which is a subset of \mathcal{RR} , that specific subset which identifies the rights and responsibilities of agent a_i at time t .

In order for λ to produce that output, the function needs to call some intermediate functions; we list them as follows:

- $\mathcal{CR}(a_i, t)$, to produce the role r_j which agent a_i is performing at time t ,
- $\mathcal{F_NKB}(r_j)$, to produce the normative rules associated with role r_j contained in normative knowledge base,
- $\mathcal{F_RO}(t)$, to produce the runtime occurrences that have taken place at time t , and
- $\mathcal{F_S}(a_i, t)$, to produce all the R&Rs and sanctions applied to a_i which have been activated by, and are still in force at time t . Therefore, $\mathcal{F_S}(a_i, t) = \lambda(a_i, t-1)$. (Recall that time is discrete.)

In other words, the function λ looks up the role assignments and the normative knowledge base to determine what rights, responsibilities and sanctions are assigned to agent a_i at time t , and then modifies these in accordance with the history of runtime occurrences and the consequent invocation of specific R&Rs and sanctions.

In this representation, we have simply defined the abstract syntax of the function, and we do not define the actual implementation (e.g. the underlying lookup procedures). Such an implementation will not be difficult, but will depend on the architecture of the underlying MAS. We have not defined a formal semantics for this syntax, because of the complexity of doing so. We leave that for future work. We next provide an instantiation of this representation, in order to provide an example application.

5.6 Examples of R&R Assignment

To facilitate the understanding of the dynamic assignment of R&Rs to agents, here we provide a worker example. In this example, we first initialize parameters of the function λ and the intermediate functions including Ag , CR , RO , NKB and S , then apply λ function on these parameters to give the dynamic assignment of R&R to an agent. As the knowledge base is static during runtime, we define NKB at design time. Then we instantiate some external agents in the MAS and provide several runtime occurrences to observe how dynamic assignments of R&Rs can be undertaken for those agents.

5.6.1 An Example for Agent set (Ag)

We suppose that currently, there are two external agents in the system such that:

$$\mathcal{A}g = \{Ali, Mari\}$$

5.6.2 An Example for the function of Current Role (CR)

In our auction system, the following roles have been defined:

$$\mathcal{R} = \{Member, Buyer, Seller, Auctioneer\}$$

The roles of the existing external agents of the system at current time (tx) are

$$CR(Ali, tx) = Auctioneer \quad \text{and} \quad CR(Mari, tx) = Buyer$$

5.6.3 An Example for Normative Knowledge Base (NKB)

The static normative KB of our auction system contains all rights and responsibilities of roles. We create this KB based on Method 2 where norms are conditional and where there are some generic norms in the system. Consider the following subset of \mathcal{NKB} , repeated from the example in Chapter 3:

$$\mathcal{NKB}_1 = \{Rule1, Rule2, Rule3, Rule4, Rule5\} \quad \text{where, } \mathcal{NKB}_1 \subseteq \mathcal{NKB} \text{ and}$$

Rule1: "If a Member is logged into the system and her/his Negative Feedback is less than 3, the Member is permitted to choose to be a Buyer."

Rule2: "If a Member chooses to be a Buyer, Member is permitted to place a bid before the ending time of the auction."

Rule3: If a Buyer places a bid, Auctioneer is obliged to evaluate the Buyer's bid within 1 minute.

Rule4: If the number of Negative Feedbacks of a Buyer is more than three, the Buyer is forbidden to join future auctions for a month, and the permission of the Buyer for placing a bid will be deactivated.

Rule5: If the current time is ending time of the auction, the permission of the Buyers for placing a bid will be deactivated and the Auctioneer is obliged to declare the end of the auction.

Now we use our syntax defined in Section 5.5.10 for representing the norms of the above normative KB.

$\mathcal{NKB}_1 = \{ Rule1, Rule2, Rule3, Rule4, Rule5 \}$

where,

Rule1: $(x, Member) \wedge (Te, EndingTime) \wedge (NF(x) < 3) \wedge (a, Auction) \wedge (gEvent(login, x, a, At(t))) \Rightarrow Activated(Prm(selectBuyer, x, a, before(Te)))$

Rule2: $(x, Buyer) \wedge (Te, EndingTime) \wedge (b, Bid) \wedge (a, Auction) \wedge (gEvent(selectBuyer, x, a, At(t))) \Rightarrow Activated(Prm(placeBid, x, b, before(Te)))$

Rule3: $(x, Buyer) \wedge (y, Auctioneer) \wedge (b, Bid) \wedge (gEvent(placeBid, x, b, At(t))) \Rightarrow Activated(Obl(evaluateBid, y, b, between(t, t + 00:01)))$

Rule4: $(x, Buyer) \wedge (NF(x) > 3) \wedge (Tc, currentTime) \wedge (a, Auction) \Rightarrow Activated(Frb(login, x, after(Tc))) \wedge Deactivated(Prm(placeBid, x, a, before(Te)))$

Rule5: $(Tc, currentTime) \wedge (Te, EndingTime) \wedge (Tc=Te) \wedge (x, Buyer) \wedge (y, Auctioneer) \wedge (a, Auction) \Rightarrow Activated(Obl(declareEnd, y, a, At(Te))) \wedge Deactivate(Prm(placeBid, x, a, before(Te)))$

5.6.1 An Example for the Set of existing R&Rs (\mathcal{S})

We suppose that so far \mathcal{S} consists of the following assignments:

$$\mathcal{F}_S(Ali, "10:03") = \{ Obl(declareEnd, Ali, Auction_1, At(11:00)) \}$$

$$\mathcal{F}_S(Mari, "10:03") = \{ Prm(placeBid, Mari, Auction_1, before(11:00)) \}$$

5.6.2 Examples of Runtime Occurrences (RO)

Now in this section we provide different examples of runtime occurrences leading to dynamic assignment of R&Rs to agents. To do so we should apply the λ

function over the arguments of the function and get the result of the dynamic assignment of R&Rs and sanctions to agents.

In the previous section, we initialized all our parameters excepting \mathcal{RO} . We first specify the environmental variables of the system. These variables are those variables in the system which describes the specification of the auction system, such as the name of Auction, Ending Time or Current Time. Here we suppose these variables have the following values:

$$v' = \{(Auction_1, Auction), (11:00, EndingTime), (10:03, Current Time)\}$$

As mentioned in Section 5.5.3, due to the variety of syntax definitions for actions and events in MASs, we did not define a specific syntax for actions (\mathcal{AC}) and events (\mathcal{EV}) of runtime occurrences, but to be consistent with the conditions of the normative knowledge base here we use the notation of SGE and IS for formulating runtime occurrences.

Now we give the values of \mathcal{RO} in this section. We identify the following runtime occurrences. Each of the following runtime occurrences may change the initial values of the other parameters. That is why in our example we always update the parameters of the function.

5.6.2.1 Occurrence 1 (New agent joins)

Now suppose that the following runtime occurrence takes place. A new agent (David) enters the system: “*David logs into Auction_1 at 10:06.*” This event can be formulated as:

$$\mathcal{A}_g = \mathcal{A}_g \cup \{David\} = \{Ali, Mari, David\}$$

$$\mathcal{F}_{\mathcal{RO}}(10:06) = gEvent(\text{login}, David, Auction_1, At(10:06)) \in \mathcal{RO}$$

Currently David is a *Member*. So the result of \mathcal{CR} mapping function would be as follows: $\mathcal{CR}(David, “10:06”) = Member$.

So far \mathcal{S} has not changed.

As a result, the dynamic assignment is met by the following function:

$$\lambda(David, “10:06”) = \{Activated(Prm(selectBuyer, David, Auction_1, before(11:00)))\}$$

which means the following norm is assigned to David:

“*David is permitted to choose to be Buyer before 11:00.*”

5.6.2.2 Occurrence 2 (An Action)

Now suppose that the following runtime occurrence, an action by David, takes place: “*David has chosen to be Buyer in Auction_1 at 10:07.*” which is formulated as:

$$F_{\mathcal{RO}}(10:07) = gEvent(selectBuyer, David, Auction_1, At(10:07)) \in \mathcal{RO}$$

After Occurrence 2, David is a *Buyer*. So the result of \mathcal{CR} mapping function would be as follows: $\mathcal{CR}(David, “10:07”) = Buyer$.

As a result, the dynamic assignment is achieved by the following value of the function λ :

$$\lambda(David, “10:07”) = \{Activated(Prm(placeBid, David, b, before(11:00)))\}$$

which means the following norm is assigned to David:

“*David is permitted to place bid during the auction time.*”

5.6.2.3 Occurrence 3 (An Event)

Suppose that the following runtime occurrence which is an event (from the Ali’s viewpoint) takes place: “*David places a bid of £25 at 10:13.*” which is formulated as:

$$F_{\mathcal{RO}}(10:13) = gEvent(placeBid, David, 25, At(10:13)) \in \mathcal{RO}$$

As a result, the dynamic assignment can be met by the following function:

$$\lambda (Ali, "10:13") = \{Activated(Obl(evaluateBid, Ali, 25, between(10:13, 10:14)))\}$$

which means that the following norm is assigned to Ali:

"Ali is obliged to evaluate David's bid."

5.6.2.4 Occurrence 4 (An Env. Parameter changes)

Suppose that the following runtime occurrence which is amendment of a parameter (here Negative Feedback) takes place: *"Enforcer adds negative feedback for David in Auction_1 at 10:55."* which is formulated as:

$$\mathcal{F}_{\mathcal{RO}}(10:55) = gEvent(addNF, Enforcer, David, At(10:55)) \in \mathcal{RO}$$

As a result, the dynamic assignment is achieved by the following value of the function λ :

$$\lambda (David, "10:55") = \{Activated (Frb(login, David, after(10:55)) , Deactivated (Prm(placeBid, David, Auction_1, before(11:00)))\}$$

which states that the following the norm is assigned to David

"David is forbidden to join the next auctions after(10:55)."

And the following norm is deactivated for David:

“David is permitted to place a bid”.

5.6.2.5 Occurrence 5 (A Deadline)

Suppose that the following runtime occurrence takes place:

“reaching to the ending time of the auction”

this is formulated as:

$$\mathcal{F}_{\mathcal{RO}}(11:00) = \text{“(11:00, Current Time)”} \in \mathcal{RO}$$

As a result, the dynamic assignment is achieved by the following value of the function λ :

$$\lambda (Ali, \text{“11:00”}) = \{ \text{Activated(Obl (declareEnd, Ali, Auction_1, At(11:00))) , Deactivate(Prm(placeBid, Mari, Auction_1, before(11:00)))} \}$$

which states that the following the norm is assigned to Ali:

“Ali is obliged to declare the end of Auction_1.”

And the following norm is deactivated for Mari (as she is the only buyer at the moment and David has been removed from the system):

“Mari is permitted to place a bid for Auction_1 before (11:00)”.

5.7 Summary

In Chapter 4, we proposed and explained our concept of dynamic assignment of R&Rs and sanctions to external agents. We have now followed that explanation with a formal definition and specification of two alternative mechanisms by which such dynamic assignment can occur. After a brief introduction in Section 5.2 we proposed two methods for dynamic assignment of R&Rs and sanctions to agents. Method 1 is based on role hierarchies, while Method 2 is based on conditional norms.

These methods have several common features as follows: both methods rely on the concept of role; both use a normative knowledge base; and runtime occurrences are considered in both methods (as described in Section 5.3). However, the two methods are not identical and their differences are as follows: the definition of roles is different because Method 1 uses a role hierarchy; and the definition of a normative KB is different, because Method 2 is based on conditional norms (described in Section 5.4).

Formal representation is very important for clarity of analysis, design and implementation of the methods. So in Section 5.5, we provided a formal representation of the function of dynamic assignment of R&Rs and sanctions to external agents based on the common features of both methods. In summary, this chapter proposed and specified two alternative methods for dynamic assignment of rights and responsibilities to agents at runtime in normative multiagent systems. In the next chapter, we will discuss implementation of these two methods.

Chapter 6

Implementation Issues

for Dynamic Assignment of R&Rs to External Agents

6.1 Introduction

So far we have described the initial proposal of dynamic assignment of R&Rs and sanctions to agents (in Chapter 4) and then proposed two methods for such assignments, followed by presentation of the formal representation of these methods (in Chapter 5). Now in this chapter we will provide the general issues for implementation of dynamic assignments of R&Rs and sanctions along with the presentation of a general architecture as a solution for providing such assignments.

In this chapter, we compare and contrast our proposed methods for implementation. From this comparison, we conclude that the most significant differences are in the definition of roles and normative KB, which means norms are defined differently in Method 1 and Method 2. But dealing with runtime occurrences are the same in both methods.

Then, based on the common aspects of the two methods, we present a general architecture - represented diagrammatically - for dynamic assignment of R&Rs and sanctions to external agents. We describe all details of the diagram along with the description of the process of such assignments.

6.2 Similarities vs. Differences in Implementation of Methods

In this section, we will provide the general issues of implementation of our methods based on their similarities and differences. Recall from Section 5.2, we described two methods for dynamic assignment of R&Rs to external agents. Method 1 is based on role hierarchies and Method 2 is based on conditional norms. Now in this section we first explain the similarities of implementation of Method 1 and Method 2, then the differences between them.

Method 1 and Method 2 have some similarities as described in Section 5.3. Both of them are role-based and contain a normative knowledge base. All main norms and enforcement norms of the normative MAS are stored in a normative KB such that the objectives of these norms are the roles predefined by system designer. The

other similarity is that dynamic resources which results in runtime occurrences are the same in both methods, as explained in Section 5.3.

With respect to differences, although we mentioned that both methods basically use roles and a normative KB, the details of the definition of roles and normative KB are different in the two methods, as mentioned in Section 5.4.

As a result, based on these similarities we will present a general architecture for implementation of our approaches for dynamic assignment of R&Rs to external agents in Section 6.5. Using this architecture, it is possible to implement a stand alone tool over a MAS intended to facilitate such dynamic assignment.

Therefore, implementation issues for the definition of roles in both methods are described in Section 6.3, followed by the design issues for creating the normative knowledge base in Method 1 and Method 2. Then based on these common features we provide a general architecture for implementation of our methods in Section 6.5. This general architecture is presented using a diagram.

6.3 Role Definition

As we explained in the previous section, the implementation of roles is different in Method 1 and Method 2. In the following, we discuss on this issue.

6.3.1 Role Definition in Method 1

Method 1 is based on role hierarchies. In this method, there are several roles structured in a role hierarchy such that each role may divide into several sub-roles and each sub-role has a specific set of R&Rs and sanctions characteristic of itself.

Practically in a MAS in which this method is applied, when an agent enters the system it automatically gets the root role of the hierarchy tree, then based on its own actions or environmental events the role of agent is changed. External agents do not have any autonomy to change their own roles and this task – changing the role of external agents - is internally performed by the internal MAS administration.

Clearly, in such a method, there are regulations for adopting a new role for agents. Such regulations specify under what circumstances which role should be assigned to which agent. These regulations are defined in the format of protocols at design time. The diagram of these protocols is composed of the hierarchy structure of roles (roles are states of the diagram) and the transitions from state to state which represents under what circumstances, the role of an agent changes from one role to another role. These protocols are statically designed at design time. So at runtime agents simply follow the predefined dialogues of protocols, moving from one state to another, and thus potentially from one sub-role to another. In other words, the designer of system should predefine protocols at design time in order to provide the assignment of sub-roles to agents at runtime.

As we are focused more on the normative area of this research, we do not consider the task of defining protocols. We just make it clear that if the MAS - intended to use Method 1 - has not been developed yet, after the definition of role hierarchy,

the system designer should also design such protocols for dynamic assignment of roles to agents. Also, if the MAS has already been developed, in addition to having a role hierarchy structure, the system should also have an in-built technique of dynamic assignment of roles to agents.

6.3.2 Role Definition in Method 2

This Method is mostly based on conditional norms. Although Method 2 also uses roles, the number of defined roles is very limited and these roles are very generic in this method. Consequently, the designer of the system need define only a few main roles in this method.

In the case of using this method over a pre-developed MAS, if the task of dynamic assignment of roles to agents has already been incorporated into the MAS, it would definitely benefit the system, as there is less work needed to create this ability for the system designer.

Otherwise, if this pre-developed MAS has not incorporated the technique of dynamic assignment of roles to agents, this task can be implemented in two ways. First, protocols similar to the described routine in Section 6.3.1 can be used. The second solution is to define these regulations in the normative KB as conditional rules, since the assignment of roles to agents follows some regulations (mentioned in Section 6.3.1).

In this way, the condition of the rule (LHS) specifies under what circumstances the role of an agent changes, while the RHS of each rule contains two commands: a command to retract the previous role of agent and also a command to assert the

new role of the agent. As an example of such a rule implemented in the Jess rule language [27], we give the following:

```
(defrule ruleName "description" LHS =>RHS )
```

where LHS (Left-hand Side) contains the conditions of the norm and RHS (Right Hand Side) contains facts, rules or functions which is fired when the condition is satisfied.

6.4 Designing the Normative Knowledge base

The design of the normative knowledge base is very important in the process of implementation of our approach. This knowledge base is the source of all norms and regulations considered by the legislator for the external agents of the system.

These norms specify which role is obliged to/or prohibited from/or permitted to/or has the right to do which actions. In addition, the norms of the normative KB specify the responses of the normative MAS if the desired action has not been executed by the external agents who play a role. It means that the enforcement norms are also defined in normative KB.

In the following, first we outline two important issues for designing of the normative knowledge base common to both methods: The type of normative KB

and the descriptive normative language of KB. Then we will consider the implementation issues of such a KB which are different between both methods.

6.4.1 The Type of the Normative KB

In our general architecture, we do not limit the designer to use any special type of knowledge base for creating a normative knowledge base. The important concern is that this normative knowledge base is used by an inference engine (which will be explained in Section 6.5), so the type of KB should be compatible with that inference engine. If the type of knowledge base is not compatible with the inference engine, a translator should be used to translate the KB to the language of the inference engine. In our architecture we also anticipate such a translator or transformer (to be explained in Section 6.5).

The inference engine we use in our general architecture is Jess rule engine [27]. As a Jess rule base needs to be based on the Jess rule language, so a normative KB should be created directly in Jess rule language, or if created in any other language, it should be first translated to the Jess rule language. In the case of using other types of KB, after completing the creation of the knowledge base, a translator component can be applied to translate to the Jess rule base.

In Jess 7.0, a native XML rule language is also supported and Jess has its own declarative XML rule language called "JessML"[29].

As we said, we use Jess inference engine in our architecture, and for simplicity we use Jess rule base language for creating our normative KB to avoid translation stage. All examples in the following are also in Jess language.

6.4.2 The Descriptive Normative Language of KB

For describing norms, a normative knowledge base should be created based on a descriptive normative language. The descriptive normative language contains the primary elements of main norms and enforcement norms. We explained our descriptive normative language in detail, in Section 3.5, followed by a presentation of a formalism for such a language in Section 5.5.10.

The descriptive normative language that is used in the KB of Method 1 is the same as the KB of Method 2. We have already defined the formalism of that language specially the formalism of *normative commands* in Section 5.5.7.

The definition of templates is one of the main features of the Jess rule base language. Therefore, here, we also define a Jess template for “*norm*” and “*enforcement norm*” comprising all the above elements of our formalism. The slots of this template are based on the key elements of main norms and enforcement norms. In the following, we defined Jess templates for “*norm*” and “*enforcement norm*” in our normative KB:

```
(deftemplate norm (multislot status) (slot deoMode) (slot act) (slot
addressee) (slot benef) (multislot object) (slot timeMode) (multislot time))

(deftemplate enforcementNorm (slot status)(slot addressee)(slot EnfCode))
```

6.4.3 Creating the Normative Knowledge Base

The main entity in the normative knowledge base is norm. This means that the normative knowledge base consists of a set of norms including all obligations, prohibitions, permissions and rights of the roles and also a set of enforcement norms, including all punishment, compensation and rewards.

We categorize norms of the normative KB in two types: *domain-related rules* and *general rules*. Domain-related rules are the norms specifically defined for the application domain. These norms should be defined by the legislator or the system designer. The syntax of the domain-related norms is different in Method 1 and Method 2.

General rules are some norms which can be used in the normative KB of every model intended to use our methods. The definition of these general rules is one of the main features of our approach. We present additional rules to the normative knowledge base for providing a part of the task of dynamic assignment of R&Rs and sanctions to external agents. General rules include a set of necessary rules for execution of commands. These rules are general and are not specific to an application. In addition, these general rules can be used in the normative KB of both Method 1 and Method 2.

In Chapter 7, Section 7.3.2, we will present general guidelines for creating the domain-related part of the normative KB for both methods which can be used by the legislator. These guidelines contain the acceptable and generic patterns and templates for Method 1 and also for Method 2 along with examples.

We also will explain the differences of KB in methods by means of examples in Section 7.3.2 and will represent that norms in Method 1 are single constraints (explained in Section 5.2.1) and just contain the normative command (compared with norms in Method 2 which have a conditional part as well). Thus norms in Method 2 are rule based (explained in Section 5.2.2) and composed of two parts: condition(s) and normative command.

In Section 7.3.2, we will also provide general rules for both methods.

6.5 General Architecture (based on common features)

In this section we present our general architecture for implementation of our techniques for dynamic assignment of R&Rs and sanctions to agents. This architecture is created on the basis of common features and similarities of Method 1 and Method 2. In Section 6.2, we explained that both methods use roles and a normative knowledge base (but in different ways) and also both of them have the same resources for runtime occurrences.

This architecture along with the issues we have already mentioned for defining roles and normative KBs provides the complete picture for design and implementation of these techniques for such assignments. This implementation is independent of the design of MAS and can be implemented over the top of a pre-developed MAS to provide this facility.

This architecture needs to use a rule engine - we chose the Jess rule engine [27], so before describing our architecture we explain Jess.

6.5.1 Using Jess

For performing reasoning tasks in this architecture we use the Jess rule engine [27]. Jess is a java-based rule engine and its java APIs can be simply used in java applications as well. This rule base engine is used by a variety of users in many different application domains.

Similar to other typical rule-based systems, Jess has three main components [28]: a rule base (or Jess knowledge base), a fact base (or working memory) and an inference engine. The rule base contains all the norms the system knows. The contents of this rule base are stored in a format that the inference engine can work with. The fact base contains information which the inference engine will operate on. Whenever the inference engine is invoked, it has to decide what rules can be fired based on the rule base and the fact base; such that if the existing facts in fact base satisfy the conditions of rules in the rule base, those rules are fired. Once the inference engine decides what rules are to be fired, it has to execute the actions of those selected rules.

The role of Jess in our method can be explained as follows: our approach provides a *Jess rule base*, a *Jess fact base* and the *invocation command* for executing Jess inference engine. The *rule base* is supplied by our normative KB (explained in Section 6.4) and created by the legislator or the normative system designer. The contents of the *fact base* are the facts of runtime occurrences which are dynamic and frequently updated in runtime. In our approach, after occurrence of each change (followed by asserting the change to the fact base), the inference engine will be invoked by executing *run ()* command in order to perform a reasoning task.

After running Jess, Jess may return some new results; such as a list of recently fired norms. The recently fired norms are new facts which should be caught and analyzed in our approach.

6.5.2 Diagram

In this section, we describe the general architecture of our methods by using a diagram (Figure 5). We propose this architecture for the implementation of our methods in normative MASs. This general architecture can be implemented as a middleware tool over every MAS to which our methods are applied. The combination of the implementation of this architecture and creation of the normative KB (described in Section 0) provides an application for our methods, and demonstrates their feasibility.

Here, in order to provide a general overview of the process, we first describe the external components - linked to the central part of the architecture - and their connections to the main entities of our architecture. Then the entities and communication processes will be explained in full detail.

At the top of the figure, there is a large shaded box showing that the MAS has interactions with external agents from its top and also has interrelations with the main internal entities from the bottom. The figure clearly shows that external agents can interact with the MAS, but they do not have any direct contact with the main entities. We suppose that the MAS has the entity of Event/Action Handler which tracks the occurrence of all runtime events and actions of the MAS and sends this information to the internal entities of the diagram.

In the right-hand side of the figure, there is a normative knowledge base which is the main normative resource of this system. This KB is a static knowledge base which stores domain-related norms including all R&Rs and sanctions of roles and also general norms for executing norms. We state that it is a static knowledge base, because we assume that there is no change in the contents of the norms stored in KB at runtime.

As shown in bottom of the figure, this architecture uses Jess to perform reasoning tasks. As explained earlier, Jess comprises three main components including a rule base, a fact base and an inference engine. In this approach, the rule base is supplied by the normative KB (via norm translator), and the fact base is supplied by the internal entities.

The internal entities of this diagram have been shown by boxes labeled En. These entities are explained in detail in the next section. The dotted arrows of the diagram labeled D are data flows and the simple arrows labeled C are Communication Processes. Data flows and communication processes are described in Section 6.5.2.2.

We put the table of functions and message-sequence charts of this architecture in Appendix C and Appendix D for further information.

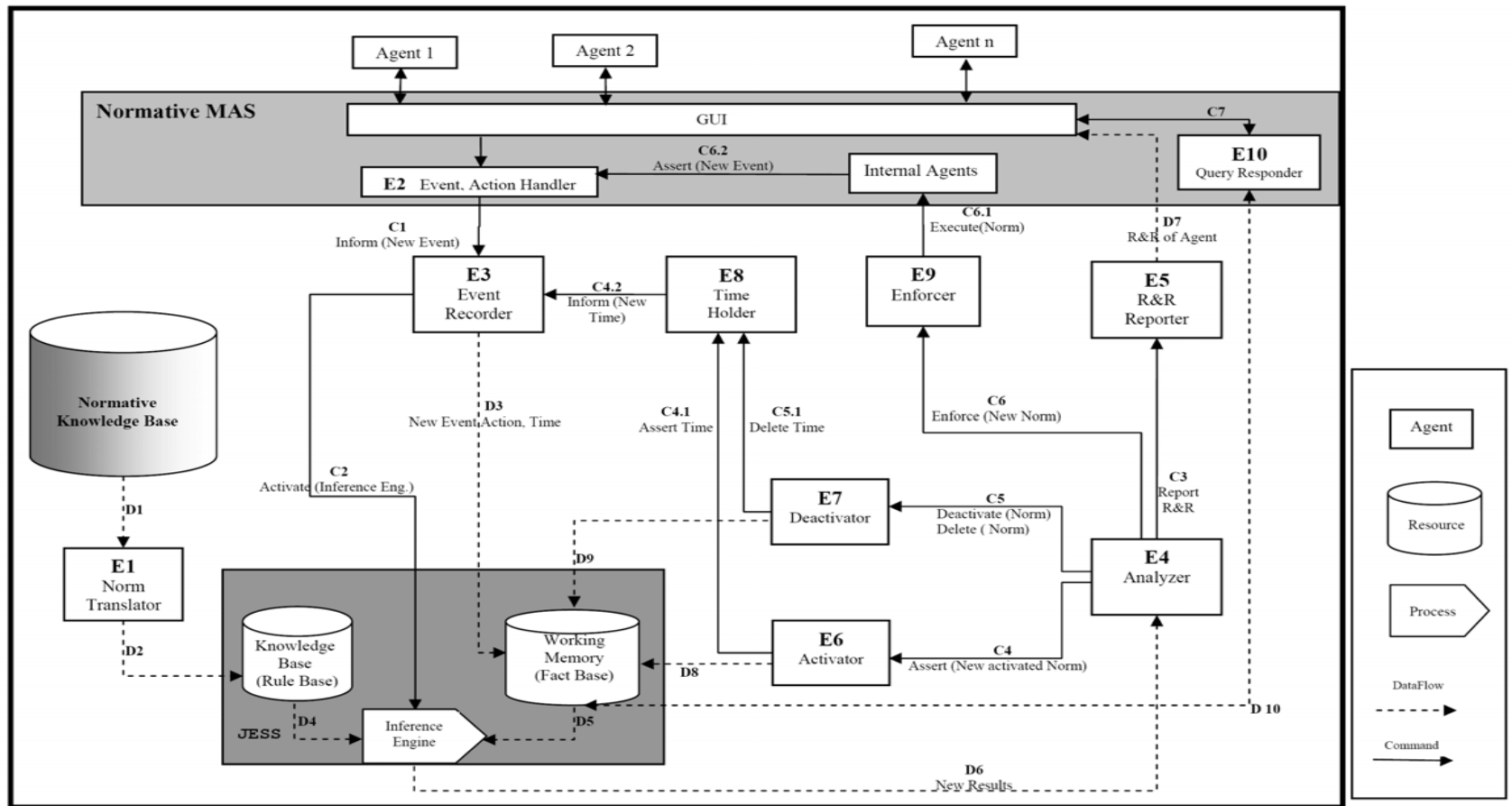


Figure 5-The Diagram of our General Architecture for Dynamic Assignment of R&Rs and Sanctions to Agents.

(D1, D2, D4 and D5 are arrows that transmit data from their source to their destination. D10 is a two-way arrow which connects Query Responder to Working Memory)

6.5.2.1 Entities

Now we explain the main entities of diagram as follows:

- E1. **Norm Translator:** As we use a Jess rule engine in this architecture, the contents of this knowledge base should be translated to the Jess language. The translation of the normative KB base to the Jess language is the responsibility of the Norm Translator, if the normative KB has been created in a different language from Jess language. The Norm Translator provides the contents of the Jess rule base from the normative KB. Therefore, if the language of KB is not compatible with Jess, we use a Norm Translator for exchanging the type of KB. D1 is the data control provides the input of the Norm Translator from normative KB, and D2 sends the output of Norm Translator which is the same KB in Jess language to Jess rule base.
- E2. **Event/Action Handler:** is responsible for catching *all runtime occurrences* such as the environmental events, the entry and exit of agents, the actions, reaching to an important time (e.g. deadlines) of the MAS and reports every new event to Event Recorder (E3) (More details on runtime occurrences have been described in 5.5.3) . This entity gets its inputs - including every event and action occurring in the system - from the GUI and the environment of MAS. Its output is transferred to Event Recorder (E3) via C1.
- E3. **Event Recorder:** is responsible for, first, asserting a new occurrence as a new fact to the Jess fact base; second, once the new occurrence is asserted as a fact, this entity is responsible for the invocation of the Jess inference engine. The inputs of this entity come from Event Handler (E2), Time Holder (E8) and Enforcer (E9) respectively via C1, C4.2 and C6.2. The inputs contain a command for asserting a new event/action, time or an internal parameter. After receiving inputs, this entity produces the Jess fact format of its input to assert this fact to Jess. The output of this entity is a data in an acceptable

format of Jess fact base transferred via D3. The other output, C2, is a command for activating Jess inference engine.

E4. Analyzer: is responsible firstly for collecting the result of Jess reasoning after each occurrence. Then this entity analyzes Jess results as the following (using the Status slot of fired norms or enforcement norms. Recall from Section 5.5.7, the status of norms including *ToBeActivated*, *Activated*, *Deactivated*, *Fulfilled*, and *Violated*):

- a. Based on the Jess result, if a new obligation, permission, prohibition or right has been fired, this new R&R transfers to the Reporter (E5) via C3.
- b. Based on the Jess results, the new fired rules have to assert to the fact base. Analyzer asks Activator (E6) via C4 to do that. (e.g. Jess results *Activate(Ali is permitted to place a bid.)*, then Analyzer asks Activator(E6) to activate this norm by putting the fired norm in fact base.)
- c. Based on the Jess results, if a modality *fulfilled* and needs to be *deactivated* or if the status of norm is violated and needs to be deactivated, then the norm should be retracted from the fact base. Analyzer asks Deactivator (E7) to do that via C5. (e.g. Previously “*Ali is permitted to place a bid*”, and this norm has been asserted to the fact base. However, due to next occurrences, currently, Jess results that “*Deactivate (Ali is permitted to place a bid)*”. Therefore, the previous fact has to be retracted from Jess fact base.)
- d. Based on Jess results, if an internal activity in MAS should be executed, then Analyzer should ask Enforcer (E9) via C6 to ask

the internal agents to do so. The internal activity can provide a change in the value of environmental variables (such as a feedback) or enforcing a punishment to (disconnecting the access of agent). As an example, suppose that the input of the Analyzer is a new fired norm says “*Execute (adding a negative feedback for Ali)*”. This is an internal activity which must be executed by Enforcer.

The input of this entity comes from the Inference engine via D6 which contains the data of the new results following the Jess reasoning. The output transfers to Reporter (E5), Activator (E6), Deactivator (E7) and Enforcer (E9) through C3, C4, C5 and C6 respectively.

E5. R&R Reporter: is responsible for displaying the new activated norms which have been assigned to the agents in the graphical user interface (GUI). This entity gets all newly fired norms and detects which norm is related to which agent to allocate its relevant R&Rs. The input of this entity is a command for reporting R&Rs, from Analyzer via C3, and the output of this entity is the data of assigned R&Rs or sanctions to agents via D7, which will be displayed in GUI.

- For example, if one of the newly fired norms is “*David is permitted to place a bid.*”, Reporter parses this phrase to find which agent this norm is related to. Then it assigns the norm to that external agent.

E6. Activator: is responsible for asserting the new activated norms to the Jess fact base. In addition, if this norm contains a time related parameter, Activator is responsible for sending a message to Time Holder (E8) to take note of that time and to notice it later on. The input of this entity is a command for asserting the activated norm via C4, and the output of this

entity is transferred via D8 and C4.1, respectively containing data to the fact base and a command for asserting the time notion of norm.

- For example, (e.g. David is the winner of the auction and the newly fired norm says "*David is obliged to pay item's price before(Te+1 hrs)* ", so the Activator send the time of (Te+1hrs) to Time Holder(E8).)

E7. Deactivator: is responsible for removing norms which have been fulfilled or which should be deactivated from fact base. In addition, if the norm had a time-related parameter which previously Time Holder (E8) got, now Deactivator should ask Time Holder (E8) to remove the norm from the list of important times. The input of this entity is a command for deactivating or deleting a norm via C5, and the outputs of this entity transfer via D9 and C5.1 which respectively shows an access to fact base (for removing the norm) and a command for deleting the time of deactivated norm from the list of important times.

- For example, in the previous example, if (Te+1hrs) has been recorded as an important time and "*David pays the price of the item on time*", then he is no longer obliged to pay. Therefore, Deactivator first removes this obligation, then asks Time Holder to delete (Te+1hrs) from the list of important times.

E8. Time Holder: Time Holder makes a list of all important times at runtime. Then whenever the current time reaches to each noted important time, Time Holder sends the current time to Event Recorder (E3) in order to assert the current time to the fact base. These important times are supplied by Activator (E6) from the time notion of activated norms. However, it is possible that Deactivator (E7) sends a message to say remove this time from the list of important times, for example, because the norm is fulfilled before

that important time. The inputs of this entity are commands for asserting or deleting a time from the list of important times, via C4.1 or C5.1.

- For example, an obligation should be fulfilled before (T1). Suppose this obligation is fulfilled sooner than this deadline(T1), Deactivator (E7) will send a message to Time Holder(E8) to remove T1 from the list of important times.

E9. Enforcer: is responsible for enforcing some internal actions. In fact, Enforcer interacts with internal agents and asks them to execute sanctions or rewards. Sometimes as a result of executing an enforcement norm, an environmental change occurs, reported by internal agents to Event/Action Handler (E2). Then Event/Action Handler (E2) sends a command to Event Recorder (E3) for the assertion of this new event. The inputs of this entity are a command for the execution of a norm via C6 and the output is one or more commands to the relevant Internal Agents via C.1.

E10. Query Responder: is responsible for answering the queries comes from GUI. This entity directly has relationship with the fact base. The input of this entity comes from GUI via C7. This entity has also access to the fact base via D10 to provide the queries answers. The output will also transfer via C7.

- *Note that we separate Reporter and Query Responder, as the responsibility of Reporter is just to represent the R&Rs of agents to GUI at each moment of time. So Reporter does not need to access to the fact base directly, as there are so much unrelated information in the fact base which Reporter does not need to access. But Query Responder needs to access the fact base for responding to various types of queries about norms.*

6.5.2.2 Communication Processes

In Figure 5, entities have communications between each other in two ways: by transferring data or by sending a control message. The data flow arrows have been labeled with D and the control arrows have been labeled with C. In the following, we briefly explain these communication processes.

- D1. This arrow shows that the normative KB is the resource for the Norm Translator (E1).
- D2. This arrow shows that the Norm Translator (E1) sends the result of translating norms to the Jess rule base.
- D3. This arrow shows that the data of new event, action or time (in the format of Jess fact base) transfers to the Jess fact base.
- D4. This arrow shows that the Jess inference engine uses and has access to the Jess rule base as a resource for reasoning.
- D5. This arrow shows that the Jess inference engine uses and has access to the Jess fact base as a resource for reasoning.
- D6. This arrow transfers the results of Jess reasoning to Analyzer (E4).
- D7. This arrow shows that the final results of assignment of rights and responsibilities of agents sends to the GUI, for presentation to the Agents.
- D8. This arrow shows that Activator (E6) has access to the Jess fact base to assert new activated norms.
- D9. This arrow shows that Deactivator (E7) has access to the Jess fact base to deactivate or delete norms from the fact base.

D10. This arrow shows the Query Reporter (E10) has access to the Jess fact base to respond to queries.

C1. This arrow shows that Event/Action Handler (E2) sends a message to the Event Recorder (E3) for asserting a new event.

C2. This arrow shows a message from Event Recorder (E3) to Inference Engine to invoke it.

C3. This arrow shows a message from Analyzer (E4) to R&R Reporter (E5) to report the R&Rs of agents arising from a newly activated norm.

C4. This arrow shows a message form Analyzer (E4) to Activator (E6) for the assertion of the newly activated norms to fact base.

C4.1. This arrow shows a message from Activator (E6) to Time Holder (E8) for adding an important time to the list of important times. Some norms contain time notions such as a deadline (e.g. an obligation should be performed before (Tx)). In such cases Activator (E6) detects the time notion and asks Time Holder (E8) to keep this time in its list.

C4.2. This arrow shows a message from Time Holder (E8) to Event Recorder (E3) to ask for the value of the current time to be recorded in the fact base. Whenever current time reaches one of the times in the list of important times, Time Holder (E8) detects it and asks Event Recorder (E3) for recording.

C5. This arrow shows a message form Analyzer (E4) to Deactivator (E7) for deactivating a norm or deleting a norm from the fact base.

C.5.1. This arrow shows a message from Deactivator (E7) to Time Holder (E8) for deletion of a time from the list of important times. In some cases a previous activated norm needs to be deactivated or deleted. Therefore, if that norm contains a time point (which previously has been added to the list of important times), now in the case of deactivation or deletion of that norm that time point should be removed from the list.

C6. This arrow shows a message from Analyzer (E4) to Enforcer (E9) for the execution of a norm.

C6.1. This arrow shows a message from Enforcer (E9) to the Internal Agents of the system for executing an enforcement norm (e.g. a punishment).

C6.2. This arrow shows a message from Enforcer (E9) to Event/Action Handler (E2) for the assertion a new event. In some cases when internal agents undertake some actions for execution of enforcement norms (e.g. increasing the negative feedback of a member due to his violation), internal environmental parameters change (e.g. the value of negative feedback increases) so this change should be reported to the fact base because it may influence subsequent reasoning tasks. (In this example, if the norm says *“If a member has 3 negative feedbacks, it is forbidden for him to log into the system next time”*, then the value of negative feedback should be updated in fact base).

C7. This arrow shows a two-way relation between GUI and Query Responder (E10) such that GUI sends a query to E10, then E10 will send the answer to GUI.

6.6 The Description of the Process

In the following, we provide a general description of the process of dynamic assignment of R&Rs and sanctions to external agents, based on the diagram of our proposed architecture. We show precisely in this diagram how such assignments occur, when, and by whom.

6.6.1 How Dynamic Assignment Occurs

Initially, this process needs a preparation stage before runtime. In this static stage, after identification or creation of system roles, the normative KB is created for those roles. Then if the language of this normative KB is different from the language of the Jess rule engine, Norm Translator (E1) should be applied on the normative KB to provide the compatible version of the rule base for Jess. Otherwise, the Jess rule base is directly supplied with the original normative KB.

In addition, suppose that all the connections between external agents and MAS, between the components of MAS and the entities of our tool, and between the entities of MAS and Jess rule engine have been set.

At runtime, agents enter and join the system, they interact with the GUI and GUI transfers their actions (as runtime occurrences) to Event/Action Handler (E2) for handling the events or the actions. Then Event/Action Handler (E2) sends a message via C1 to Event Recorder (E3) for informing that an event happens.

Next, Event Recorder (E3) accesses the Jess fact base via D3 to assert this event. Immediately after assertion, Event Recorder (E3) asks the Jess Inference Engine for reasoning via C3.

Then, the Jess Inference Engine reasons using the Jess fact base and the Jess rule base. After that, Analyzer (E4) gets the results of reasoning from the Jess Inference Engine via D6 to check which new norms have been fired by the recent occurrence. Jess results are a set of normative commands in the following template, as described in Section 6.4.2.

Next, Analyzer (E4) parses the above results (normative commands) and on the basis of the Status slot of each normative command, it makes decision for the next stage as follows:

1. If the normative command indicates an activation or deactivation, Analyzer (A4) sends the norm (or enforcement norm) to the Reporter (E5) via C3 for reporting the R&R.
 - If the normative command is an activation, Reporter (E5) parses the norm to specify which external agent this norm or enforcement norm is related to. Then, the norm (R&R) or enforcement norm (sanction) should be assigned to the recognized external agent. Therefore, Reporter (E5) provides the GUI format of the assigned norm and passing through D7 sends the data to display its specific frame in the GUI.
 - If the normative command is a deactivation, Reporter (E5) parses the norm to specify which external agent this norm or enforcement norm is related to. Then, the norm (R&R) or enforcement norm (sanction) should be removed from the specific frame of the external agent in the

GUI. Therefore, Reporter (E5) will access to the relevant frame via D7 to remove the norm or enforcement norm.

At this stage, the dynamic assignment of R&Rs or sanctions has been done for just one occurrence. However, for subsequent assignments of R&Rs to agents for all occurrences of the system, one needs to update the fact base (using Activator, Deactivator and Time Holder) and execution of internal enforcement actions (using Enforcer). Thus, the other responsibility of the Analyzer is the following based on the Jess reasoning results:

2. If the normative command indicates an activation, Analyzer (E4) sends a message via C4 to Activator (E6). Then, Activator parses the norm. If the norm contains a time notion, it will ask Time Holder (E8) to assert the time in its list of important times via C4.1. In addition, Activator (E6) will assert (update) the activated norm by accessing the Jess fact base via D8.
3. If the normative command indicates a deactivation, Analyzer (E4) sends a message via C5 to Deactivator (E7). Then Deactivator parses the norm. If the norm contains a time notion, it will ask Time Holder (E8) to delete the time from its list of important times via C5.1. In addition, Deactivator will remove (update) the deactivated norm by accessing the Jess fact base via D9.
4. If the normative command indicates an execution, Analyzer (E4) sends a message via C6 to Enforcer (E9). Then, Enforcer will send a message to Internal Agents to execute the enforcement norm via C6.1. This message contains the statement that the agent should be punished along with the related enforcement code. If the execution of the enforcement norm leads to a runtime occurrence (an environmental event or events), such an event

should be reported to Event/Action Handler (E2) for further action (repeating the cycle we described above).

There is another case which leads to a repetition of the above cycle. We have mentioned that Time Holder (E8) keep the important times in runtime. Then, when the current time reaches these times, Time Holder will inform Event Recorder (E3) via C4.2 that the current time is an important time. Therefore, this runtime occurrence - reaching an important time - will be asserted to the Jess fact base through Event Recorder (E3), and another process will be started.

In addition, if there exist queries from the external agents and it is desired to facilitate query responding, Query Responder (E10) has direct access to the Jess fact base via D10 for answering the queries. C7 is a two-way path for receiving queries from the GUI and responding to them via the GUI.

6.6.2 When a dynamic assignment occurs

On the following occasions, dynamic assignments may occur:

- **Entry and exit of agents:** On these occasions, an agent gets a new role (sub-role in Method 1). Thus, according to the rule base and fact base, dynamic assignment of R&Rs happens and the initial related norms of the new role will be fired for the agent.

For example, “*Mari joins the system as a buyer*”, then the primary R&Rs of buyers will be fired for her, such as “*Mari is permitted to place a bid before ending time of the auction*”.

- **Occurrence of an Action:** When an action occurs, according to the rule base and the fact base, dynamic assignment of R&Rs may happen.

For example, suppose that Mari is a buyer and Ali is an auctioneer in an auction, then this action (from Mari's view) happens: "*Mari places a bid.*", then a set of norms will be activated for Mari, such as "*Mari has the right to get respond for acceptations or rejection of her bid within 1 minute.*".

- **Occurrence of an Event:** When an event occurs, according to the rule base and the fact base, dynamic assignment of R&Rs may happen.

For example, suppose that Mari is a buyer and Ali is an auctioneer in an auction, then this event (from Mari's view) happens: "*Ali accepts the bid*". In this case, the previously activated norm for Mari should be deactivated or removed.

- **Reaching to an Important Time:** When the current time is an important time, such as achievement of a deadline, according to the rule base and the fact base, dynamic assignment of R&Rs may happen.

For example, suppose that Ali is the auctioneer of an auction and based on the KB

Norm 1: "Auctioneer is obliged to declare the ending-time of the auction."

Norm 2: "If the current time is ending time of the auction and Auctioneer did not declare the ending time, he will be punished by punishment_22."

If “*the current time is ending time*” of the auction and Ali did not declare ending of the auction, he will be punished according to Norm 2. Therefore, here a sanction is assigned to Ali as an external agent.

6.6.3 Who assigns dynamic assignments

As described in the diagram, all the entities of this approach contribute to the goal of providing dynamic assignment of rights and responsibilities to external agents. But specifically Reporter (E5) is the agent who decides which activated norm or enforcement norm is related to which external agent, and then asks the GUI to represent the result of dynamic assignment at every stage. Reporter also removes deactivated norms or enforcement norms from the GUI frames of external agents.

6.7 Using Method 1 or Method 2

After the presentation of our two methods and the general architecture for implementation of them, in this section we discuss how, when and which method should be used for dynamic assignment of R&Rs and sanctions to agents. Our proposed general architecture can be used for the implementation of both methods. The reason is that we have designed this general architecture based on the similarities of the two methods.

The other point is that we have designed this architecture independently of the type of MAS: this architecture can be applied either on *a MAS which is designed from scratch* or over *an existing MAS which is intended to facilitate dynamic assignment of R&Rs and sanctions to external agents*.

If the MAS intended to use one of our methods has not been designed yet, therefore, simply our general architecture can be considered in its analysis and

the design to be developed. Otherwise, if the MAS has already been developed, our general architecture can independently be developed and connected to the MAS as a standalone tool. This middleware tool collects its inputs (including the normative knowledge base, events/actions of the MAS and the system clock), then after processing the inputs, will dynamically output assignment of R&Rs and sanctions to external agents.

It is also useful to discuss which option is the most appropriate from our two methods for dynamic assignment of R&Rs and sanctions to external agents. Therefore, here we explain when Method 1 is more suitable than Method 2 and vice versa.

As we mentioned, the significant differences of these two mechanisms lie in the definition of roles and normative KBs. The usage of these two methods is also relevant to their differences.

If the existing MAS intended to facilitate dynamic assignment of R&Rs and sanctions to external agents has a very detailed and multilevel role structure, then, Method 1 is the better option to use. This is because, the definition of the norms in the normative knowledge base is more convenient in Method 1 and it is straightforward for the system designer to define the normative knowledge base. Norms in Method 1 are a simple constraint such as, “*Winner is obliged to pay the payment of the auction*”.

If the existing MAS intended to facilitate dynamic assignment of R&Rs and sanctions to external agents contains only a few roles, then Method 2 is the better option to use. In this case, the norms of the normative system are conditional and the normative knowledge base contains conditional norms. In Method 2, the designer of the normative system should precisely define the

conditions of the norms. Norms in Method 2 have two parts: *precondition(s)* followed by the *constraints*. For example, “*If the Buyer wins the auction, he is obliged to pay the payment of the auction*”.

In the case that the MAS has not been developed yet and has to be designed from the scratch, the designer can choose Method 1 or Method 2. Using Method 1 needs a precise role structure but needs less effort for the definition of the normative knowledge base. While using Method 2 needs only to define the main roles, the conditions of the norms of the normative knowledge base should be defined very precisely. The selection of each method depends on the designer choice, the prior infrastructure of the MAS, and the degree of sophistication of the normative system intended to be applied within the MAS.

6.8 Summary

In Chapter 4, we proposed our two methods for dynamic assignment of R&Rs and sanctions to external agents, followed by formal representation of these methods. Next in this chapter, we discussed the main implementation issues of these methods.

At the beginning, in Section 6.2, we discussed the similarities and differences of both methods from the implementation viewpoint and we concluded that the most important differences are in the definition of roles and the normative KB; this means that norms are defined differently in Method 1 and Method 2. Runtime occurrences are the same in both methods.

Consequently, in Section 6.3, we explained different ways of role definition in Method 1 and Method 2. In Section 6.4, we discussed the main issues for designing the normative KB in both methods, including the type of knowledge base, the normative descriptive language, and general issues for creating normative KB in our methods.

We proposed our general architecture through a diagram on the basis of the common features of the two methods in Section 6.5. We have described this diagram with details including entities and communication processed. Finally, in Section 6.6, this architecture diagram has been used to describe the end-to-end process of the dynamic, runtime assignment of R&Rs and sanctions to the external agents in a normative MAS.

Chapter 7

The Design of a Middleware Tool

for Dynamic Assignment of R&Rs to External Agents

7.1 Introduction

We discussed the main issues and presented an architecture for the implementation of our proposed methods for dynamic assignment of R&Rs and sanctions to external agents in Chapter 6. To demonstrate the practical feasibility of our approach and of our architecture, we have developed a middleware software tool to enable the provision of our dynamic assignment methods in normative MAS.

In this chapter, we explain the development stages of this tool including analysis and design through to implementation. This tool is not application-specific, but is generic, and so may be incorporated into any normative multi-agent system which can validate our proposed approach.

In Section 7.2, we explain the analysis stage along with the description of the functionality, inputs and outputs of this tool. Next, in Section 7.3, we describe the design of this tool including the design of tool's entities, its normative KB and the other main components, such as a timer and a user interface. We use a realistic auction example for testing this tool; therefore, in this section, we create the domain-related part of the normative KB for that auction example.

We will complete the discussion of development stages of this tool in Chapter 7, with an evaluation of it.

7.2 Analysis

We are motivated to develop an application called "*R&R Allocator*" as a middleware tool in order to demonstrate the applicability of our proposed methods for dynamic assignment of R&Rs and sanctions to external agents of any normative MAS. For such a development, we consider the main implementation issues discussed in the previous chapter.

In Chapter 5, we presented a general architecture based on the common features of our methods for implementation of such a tool. Therefore, here in this section on the basis of this architecture we develop an application.

At a general level, we considered two main issues for developing this application: first, this tool implements *both* of our proposed methods —Method 1 and Method 2. As mentioned in Section 6.2, these methods have both similarities and differences. Based on these similarities, the diagram of general architecture has been proposed which provides the main process of dynamic assignment, independent of choosing Method 1 or Method 2. The main difference between the two proposed methods is the creation of the normative KB which is static, being built at design time and provided to the tool as an input at runtime.

Secondly, we design an independent tool that can be used for any multiagent system for which normative features are intended to be incorporated. So this tool is designed as a middleware tool and it can be attached to an existing multiagent system to perform the capability of dynamic assignment of R&Rs and sanctions to external agents. In this way, appending the normative features over a multiagent system does not require any changes to the design, implementation or development of the MAS itself. The tool just takes some required inputs from the multiagent system and from the normative KB, and then returns the result to the MAS after each dynamic assignment. This feature gives the tool wide potential applicability.

We next present the analysis of the tool as follows: firstly, the main functionality of the tool based on the common features of the two methods is described and then, secondly, the inputs and outputs of the tool are specified.

7.2.1 The Functionality of the Tool

Here we explain the general functionality of the tool. This tool is implemented based on general architecture along with the creation of a normative KB based on either Method 1 or Method 2.

The creation of the normative KB is a design task and the normative KB should be created by the system designer at design time whether based on the format of Method 1 or Method 2. Then, this KB will be used as an input of the tool. Recall that this normative KB contains all the main norms and the enforcement norms. In addition to these norms, this tool must insert additional processes for providing the process of dynamic assignment of R&Rs. All of these norms are in the Jess language and should be substituted for the rule-base of Jess.

Then, at runtime, the main work of this tool is started. This tool is connected to a MAS and obtains all events and actions dynamically as they occur, such as entry and exit of external agents, the actions external agents undertake and any environmental events. For each of these runtime occurrences the following tasks are undertaken, in a continuous cycle:

- Our tool first asserts this event as a new fact in the Jess fact base, then, activates Jess to perform reasoning task. The Jess inference engine undertakes the reasoning task using the rule base and fact base.
- Next, this tool collects the results of Jess reasoning and analysis of these data. The result of this analysis would be a set of new assigned norms to internal agents of the MAS or a set of new assigned norms to external agents.

7.2.2 Inputs and Outputs of the Tool

Using the above description of the functionality of the middleware tool, identifying the inputs and outputs of the tool is very straightforward. One of the main inputs of the tool is the normative knowledge base including the main rules describing all obligations, prohibitions, permissions, and rights of agents along with the temporal functions and also enforcement of norms including

punishment, reward and compensation. This input is provided by normative designer or the legislator of the normative system.

The other inputs are the runtime occurrences which includes events, actions, and time. Runtime events and actions can be provided by the Event/Action Handler component of the multiagent system. For time inputs, the system needs timers for announcing the important times (detected inside the application); therefore, the application needs to use the clock of the system to provide such inputs.

This tool has two types of outputs: the first one is the result of dynamic assignment of R&Rs and sanctions to external agents of the MAS which at each instant of time presents what rights and/or responsibilities have recently been allocated to each agent.

The other output is the enforcement instructions for internal agents of the MAS. As described before, our rule base contains enforcement norms. When the tool executes norms of the system, the enforcement norms will be fired as well. These enforcement norms contain punishments, compensations, or rewards, all of which should be enforced by internal agents of the MAS over external agents. In our application, every punishment, compensation or reward norm has an instruction code for internal agents. Based on this code, the internal agent can execute the related instruction for punishment, compensation or reward.

7.3 Design

In this section, we present the design of the tool, including the entities of the tool, normative KB, timer and user interface.

7.3.1 Design of Tool Entities

After identification of the functionality of the tool and the inputs and outputs of that, we now present the design of the application based on the analysis just presented. Here the diagrams of the software engineering design, including a use case diagram, an activity diagram and a class diagram, are presented with detailed descriptions for each diagram.

These diagrams have been created with UML environment in Java NetBeans IDE 6.0 [54, 77].

7.3.1.1 Use case diagram

Figure 6 shows the basic use case diagram of the tool. This figure shows the resource of inputs of the system on the left-hand side, the general functionality of the system in the middle box, and the target of outputs of the system on the right-hand side. As shown, the inputs come from Normative Knowledge Base (designed by the legislator), Event/Action Handler (as a component in any multiagent system) and system clock.

The use cases inside the box show a general view of the tool's functionality. The use case of "*Provide Norm(R&Rs)*" gets the norms from Normative Knowledge Base. The use case of "*Provide Event/Action*" gets events and actions from Event/Action Handler in MAS. *Clock* is the resource provider of the use case of "*Provide Time*". Finally, the use case of "*Provide Assigned*"

R&Rs” performs the result of the process of dynamic assignment of R&Rs, and returns the outputs to the internal and external agents.

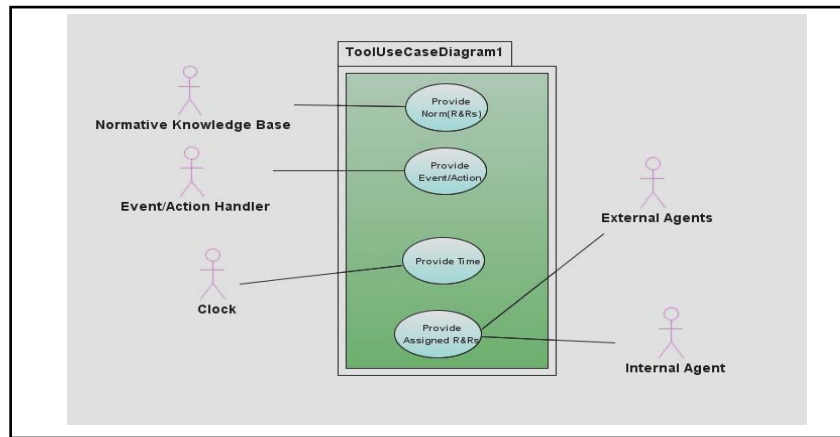


Figure 6- Use Case Diagram

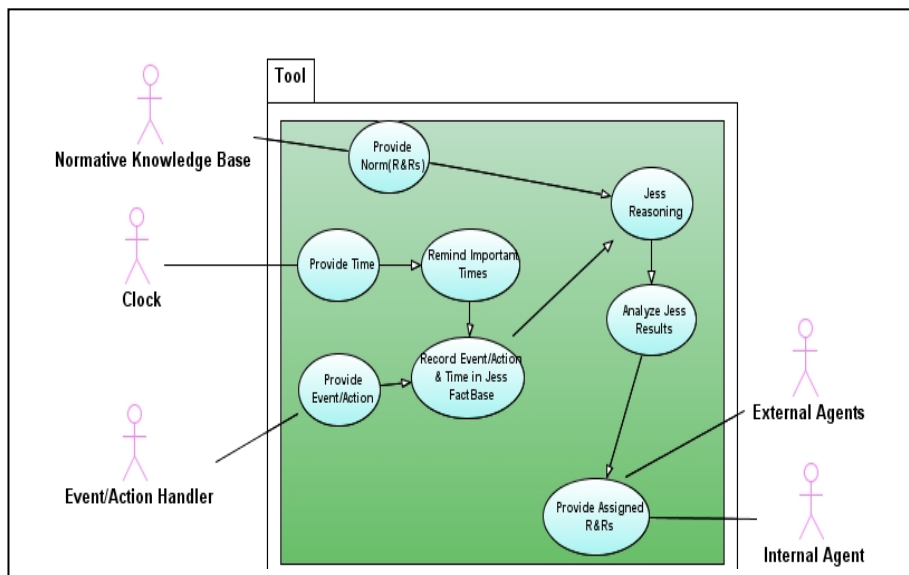


Figure 7-The Detailed Use Case Diagram

Figure 7 shows a detailed use case diagram. The inputs and outputs of the diagram are the same as for the basic use case diagram, but more use cases are displayed to present more features of the tool.

The diagram shows that the unit of “*Provide Norm*” gets the Normative Knowledge Base and provides norms for “*Jess reasoning*”.

The unit of “*Provide time*” provides time from Clock to pass to “*Remind Important times*”. These important times are sent to “*Record Event/Action & Time in Jess Fact Base*”.

The unit of “*Provide Event/Action*” gets data from “Event/Action Handler” of multiagent system and sends to “*Record Event/Action & Time in Jess Fact Base*” and then to “*Jess Reasoning*”.

Then, the result of “*Jess Reasoning*” sends to “*Analyze Jess Results*” and after analysis it goes to “*Provide Assigned R&Rs*”. Finally the results will be reported to the internal and external agents.

7.3.1.2 Activity Diagram

In Figure 8, the Activity Diagram of the tool is presented which contains more details of the process of dynamic Assignment of R&Rs to the external agents.

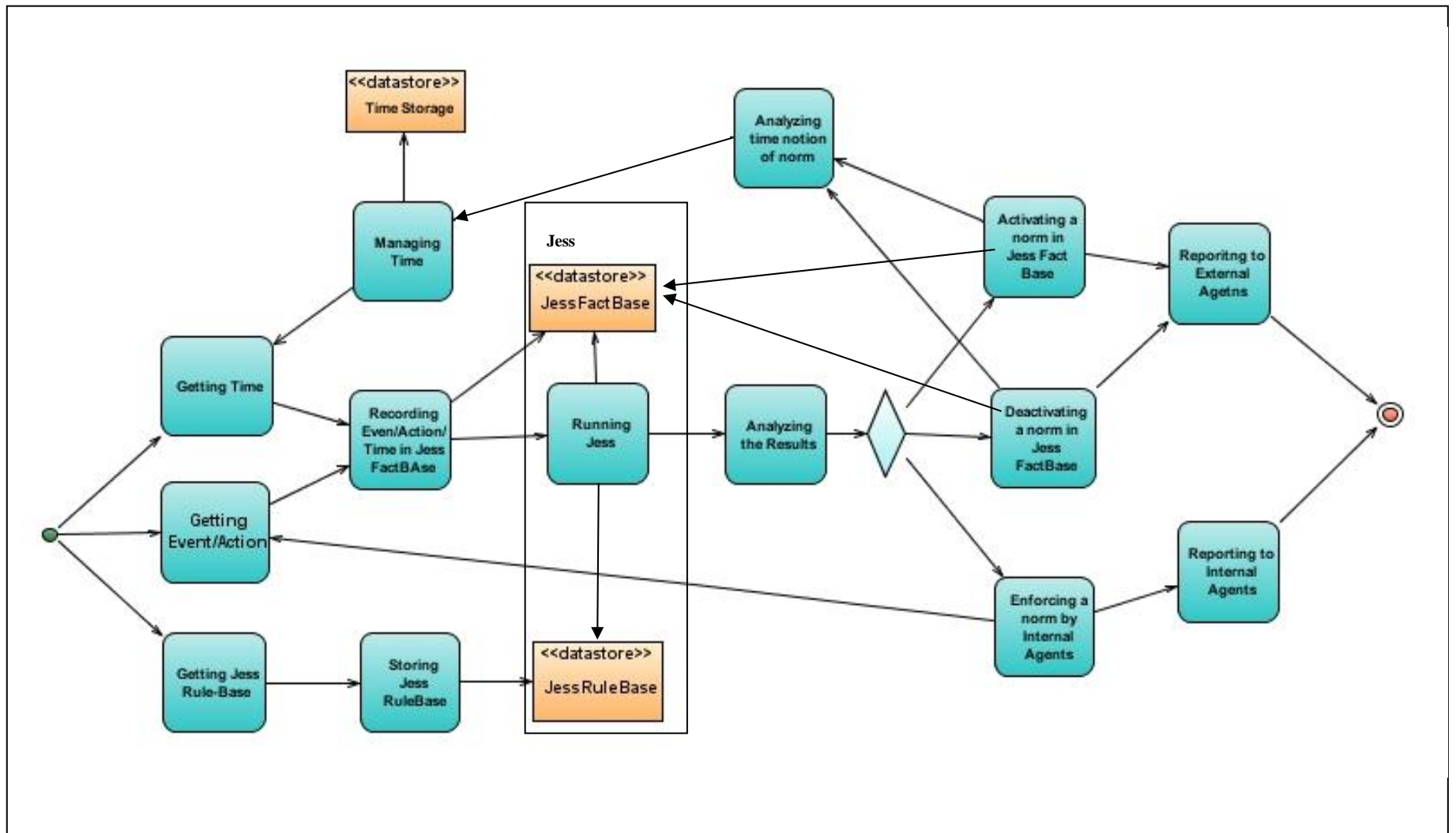


Figure 8-Activity Diagram of the tool

7.3.1.3 Class Diagram

We now present the class diagram of the tool, provided in Figure 9. This diagram is based on the activity diagram and is the basis for implementation of the tool. It presents a number of classes (with their attributes and methods), their relationships to one another, and the output of the system.

There are seven main classes, as follows:

InputSimulator: This class provides inputs of the tool. As mentioned, there are two types of inputs in this tool: normative KB and event/action. The file of normative KB (in Jess language) is the first input which this class substituted in Jess rule base using Set and Get operations. This class also simulates event/action inputs which in the real application should be provided by Event/Action Handler. As we want to concentrate on the main process of dynamic assignment of R&Rs to agents, here we just simulate the inputs we expect to get from event/action handler.

Clock: This class works based on the system clock. This class gets a time from NormAnalyzer, creates a timer for that, and then announces it to JessInteractionProvider when the system's clock shows the time.

JessInteractionProvider: This class can connect to Jess for reasoning and to provide the required results for NormAnalyzer.

NormAnalyzer: This class gets Jess Results after each Jess reasoning activity, then analyzes the results, and sends the analysis to NormReporter and Enforcer.

NormReporter: This class creates an ExternalAgentFrame for the existing external agents of the system. When the list of new assigned norms is sent to NormReporter, for each norm this class recognizes the ExternalAgentFrame related to the norm and then send the assigned norm to that frame.

ExternalAgentFrame: This class provides the user interface for the existing external agents of the MAS and reports dynamic assignment of R&Rs and sanctions to external agents.

Enforcer: This class is responsible for enforcing norms when a punishment should be executed, or compensation claims or a reward should be submitted. The class detects the instruction code of the enforcement task and reports this to internal agents for running the instruction. Sometimes running such instructions causes changes in environmental variables which are important and affect the normative part of the system. For this reason, Enforcer asserts such changes as new facts to the Jess fact base.

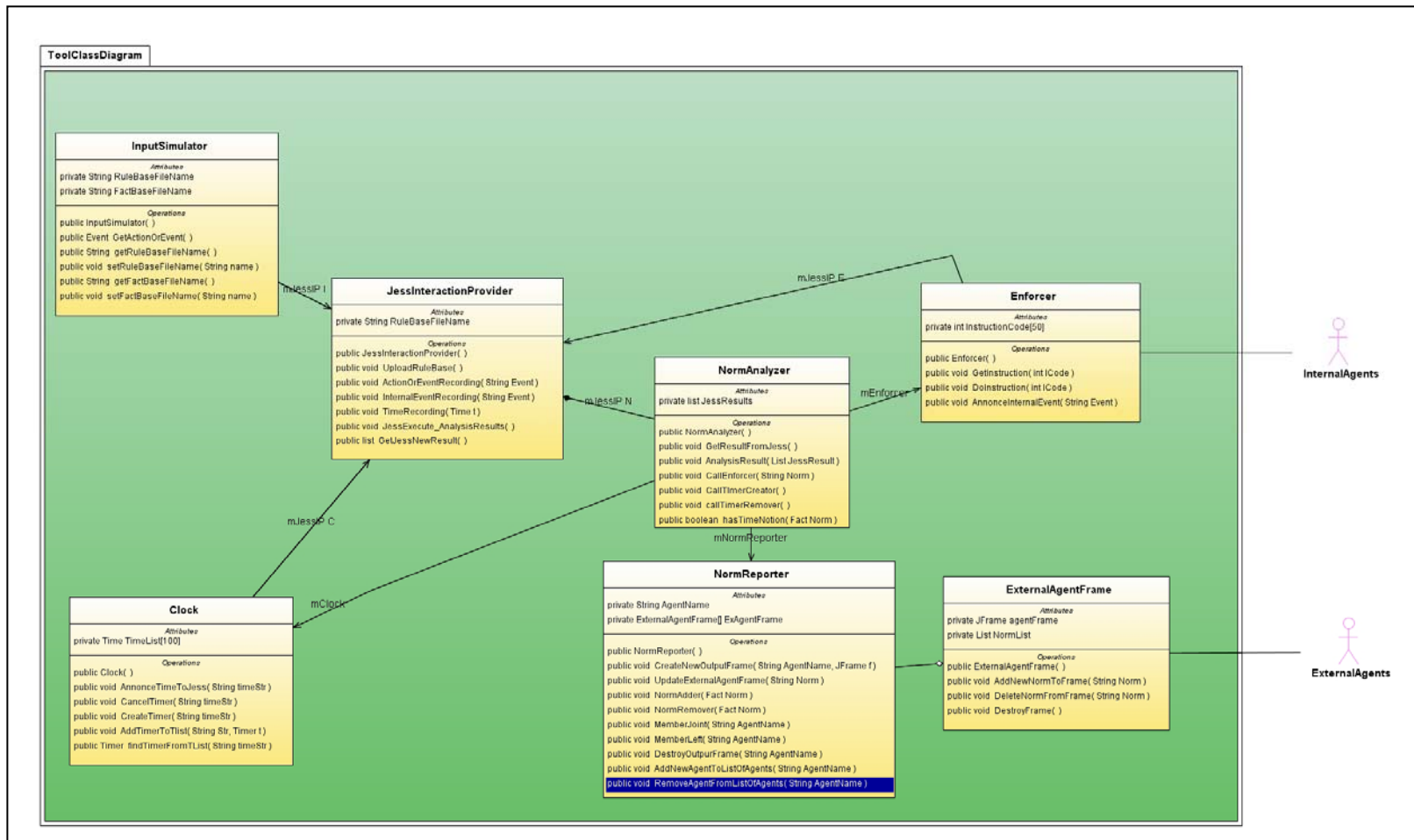


Figure 9- Class Diagram of the Tool

7.3.2 Design of Normative Knowledge Base

The normative knowledge base is a very important part of this tool. This knowledge base is the source of all norms and regulations considered by the legislator for the external agents of the system. These norms specify which external agent under what pre-condition are obliged to/or prohibited from/or permitted to/or have the right to do which actions. In addition, the norms of the normative knowledge base specify the reactions of the normative system if the desired action has not been executed by the external agent.

In the previous chapter, Section 6.4, we explained main issues for designing normative KB, including *the type of normative KB*, *the descriptive normative language of normative KB* and *the issues for creating normative KB*. Here we also briefly describe the type and language of the KB in this application, and then explain the creation of KB with more details.

- **The type of KB:** In this application, we make our normative knowledge base directly in Jess rule language for simplicity, though our proposed procedure for designing the KB is general and applicable for any other type of knowledge base as well. The Jess rule base has a specific syntax for describing templates, rules, facts, function definitions and execution commands. Therefore, the normative knowledge base should follow such syntax for defining the norms of the system. For example the definition of a rule in Jess language is as follows:

```
(defrule ruleName "description" LHS =>RHS )
```

where LHS (Left-hand Side) contains the conditions of the norm and RHS (Right Hand Side) contains facts, normative commands or functions which is fired when the condition is satisfied.

- **Descriptive normative language:** For describing normative commands (RHS in rule definition), we use the descriptive normative language which we have already defined in Section 3.5 and formalized in Section 5.5. The descriptive normative language contains the main elements of the norms and enforcement norms. These elements are represented in the Jess template defined for description of the norms. Here we also define a Jess template for “*norm*” and “*enforcement norm*” comprising all the mentioned norm elements. The description of a norm in Jess template in the knowledge base of this tool is as follows:

```
(deftemplate norm (multislot status) (slot deoMode) (slot act) (slot addressee) (slot
benef) (multislot object) (slot timeMode) (multislot time))

(deftemplate enforcementNorm (slot status)(slot addressee)(slot EnfCode))
```

Our tool application also recognizes the keywords we defined in the templates in order to analyze the results of Jess reasoning.

7.3.2.1 Creating the Normative Knowledge Base

In this section, we provide general guidelines for creating a normative knowledge base. In Section 6.4 we mentioned that the normative KB of this tool contains two types of norms: domain-related rules and general rules.

As we explained in Chapter 2, every full normative system has the following types of norms: **Main Norms**, which define the regulations of the system; **Check Norms**, which are rules for detection of violation, reward and compensation cases; and **Reaction Norms**, which are rules defining what punishment or reward should be enforced as the reaction of a violation or reward case.

Domain-related rules include all *main norms and reaction (or enforcement) norms* related to the domain of the application. These norms are created by the legislator or normative-system designer. General rules are additional rules which we generally define for all applications intended to use our methods. These rules include a set of rules which are necessary for the execution of the process of dynamic assignment. They also contain *check norms* for detection of violation, reward or compensation cases. This tool will automatically attach these rules to the domain-related part of the normative KB.

As this knowledge base is based on the Jess language, before describing domain-related rules and general rules, we provide an introduction to templates and functions in Jess language which can be used by the legislator to define norms. Then, we provide guidelines for creating the domain-related part of the knowledge base, followed by a presentation of the general rules we defined in this tool.

7.3.2.1.1 Defining Templates and Functions

Jess language facilitates the definition of templates and functions. So in this section we present templates and functions will be used in for the definition of norms of the tool's knowledge base.

There are two types of templates in this normative KB: first, templates for *general definitions* which can be used for every KB intended to use our tool. Second, templates for *specific domain definitions* which differ from one application domain to another. For example, in an auction system there are some specific templates relevant to the auction domain.

In the following, we define the *general templates* that are used in the creation of the normative KB of this tool. These templates include *AgentJoined* and *AgentLeft* that define the entry and exit of agents. Here we consider two slots for these templates, the name of the agent and the time it joins or leaves the MAS. We also consider a template for defining the *role* of the agents including the role title of the agent, the name of the agent, and the time that the role has been assigned to this agent. The other template we define is *event*. In the definition of this template the following slots are important for us: the act has happened, the actor of that event, the person this act is provided for, the time of event, in addition to a multislot for objects which we provided it for other factors that may be important for the definition of specific events. We also defined a template for *current time* which has a *value* slot.

The definition of the templates of norm and enforcement norms are in accordance with the formalism we provided in Section 5.5.10.

```
(deftemplate AgentJoint (slot agentName)(slot AtTime))
(deftemplate AgentLeft (slot agentName)(slot AtTime))
(deftemplate role (slot roleTitle)(slot agentName)(slot AtTime))
(deftemplate event (slot act)(slot actor)(slot forPerson)(multislot object)(slot AtTime))
(deftemplate currentTime (slot value))
(deftemplate norm (multislot status) (slot deoMode) (slot act) (slot addressee)(slot
benef) (multislot object) (slot timeMode) (multislot time))
;in norm definition benef stands for beneficiary or benefactory
(deftemplate enforcementNorm (slot status)(slot addressee)(slot EnfCode))
```

As mentioned, each knowledge base has a specific template, for defining *domain related definitions*. As we will assess our tool with an auction example, in the following, the specific domain-related templates of our auction example

have been provided. These templates will be used in the definition of the domain-related rules of the normative KB.

```
(deftemplate auctionStartTime (slot value))  
(deftemplate auctionValue (slot item)(slot auctionID)(slot price))  
(deftemplate feedback (slot actor)(slot value))
```

In order to provide arithmetic calculations or comparisons in the definition of norms, it is sometimes necessary to define *functions* in addition to defining templates. Here we present an example of function definition in our auction knowledge base example.

Based on our auction regulation, the start time of the auction for advertised item is one hour after advertisement time ($\text{startTime} = \text{advertisementTime} + 1 \text{ hours}$). The input of this function is a string in the standard format of UTC or Universal Time (e.g. "3 Jan 2008 09:14:20 GMT"). We use `java.util.Date` class for the definition of date as well.

```
(deffunction getStartTime (?x)(bind ?date (new java.util.Date))(bind ?longFrmt (call  
?date parse ?x))(bind ?lf (+ ?longFrmt 3600000))(bind ?endDateObj (new java.util.Date  
?lf))(bind ?strFrmt (call ?endDateObj toGMTString))(return ?strFrmt))
```

7.3.2.2 Domain-Related Rules

After defining Jess templates and functions for the knowledge base, the legislator or system designer can define the main rules of the system in a knowledge base using the defined templates and functions.

There are two ways that a legislator can select to create this part of KB: using a role hierarchy based on Method 1 or using conditional rules based on Method 2. For simplicity of our explanation, in the following we first provide the examples of Method 2, then for Method 1.

Method 2: In Method 2, for the definition of all norms, the basic format of Jess “*defrule*” (mentioned in Section 7.3.2), the above mentioned templates such as norm, and functions definition would be used. The Left-Hand-Side (LHS) part of the rule contains all conditions of norms which will activate them.

In the following, we provide several examples of norm definitions in the domain of an auction application on the basis of Method 2. The list of all domain-related norms based on Method 2 (including main norms and enforcement norms) of the example of auction system is presented in Appendix E.

For instance, the following rule shows an example of the main rules created by a legislator. This rule says “*If seller advertises an item, Seller is not allowed to place a Bid during the Auction Time (between Start Time and End Time)*”. For describing this norm, we use Jess templates (such as role, event and norm) and the function (getStartTime) which were defined in the previous sections. In the following, we provide the definition of this rule.

```

(defrule placingBidForbidden
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?event <-(event (act advertiseItem)(actor ?x)(object ?item ?auction ?price)(AtTime
    ?time))
  =>
  (bind ?startTime (getStartTime ?time))(assert (auctionStartTime (value ?startTime)))
  (assert (norm (status ToBeACTIVATED)(deoMode Frb) (act placeBid) (addressee ?x)
    (object ?item ?auction) (timeMode BETWEEN) (time (getStartTime ?time)
    (getEndTime ?startTime)) ) ) )

```

The legislator of a normative KB is also responsible for defining reaction norms of the normative system. We defined a template for reaction norms as *enforcementNorm*. Such reaction norms determine that the system has anticipated what punishment or compensation for the case of violations, or what reward has been considered, for enactment of the various norms.

In the following, we present an example of an enforcement norm which is defined by the legislator of the system. This rule says “*If the obligation of the act of placeHigherBid is violated, the addressee is punished by P1*”. P1 is an instruction code which should be run by internal agents of the multiagent system. For description of this norm we used the templates of norm and enforcementNorm.

```

(defrule wrongBidPunishment
  ?normFact <- (norm (status VIOLATED) (deoMode Obl)
    (act placeHigherBid)(addressee ?x))
  =>(assert (enforcementNorm (status PUNISHMENT) (addressee ?x) (EnfCode P1))) )

```

Reward norms are the other types of reaction norms which will be fired in cases where the legislator wishes to encourage agents to execute a permitted action, or to execute a task sooner than the agreed deadline, or for similar reasons. Therefore, reward norms are fired as the result of enacting norms at or before a specific time (specified by the legislator).

In our approach, an enactment norm is labeled as FULFILLED and the time of this fulfilment is also stored. So given these two aspects, the implementation of such norms is very straightforward. In fact, if legislator wants to define a reward norm, he needs only to verify the status of the desired norm and the time of fulfilment.

Here we provide an example of reward. The following norm says “*if a buyer pays before the deadline of fastPaymentTime, the reward R1 should be assigned to that buyer*”. Suppose that the legislator states that fastPaymentDue Time is 5 minutes after winning the auction, while the buyer has the right to pay until paymentDueTime (which is one hour after winning the auction). The purpose of this norm is to encourage buyers to pay quickly.

```
(defrule rewardForFast Payment
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))
  ?normFact <- (norm (status FULFILLED)(deoMode Obl)(act pay)
                  (timeMode AT)(time ?t))
  (test (< ? t (getFastPaymentDueTime ?startTime)))
  =>
  (assert (enforcementNorm (status REWARD) (addressee ?x) (EnfCode R1))) )
//where fastPaymentTime is 5 minutes after winning time.
```

Method 1: Similarly, Method 1 also uses the basic format of Jess “*defrule*” (mentioned in Section 7.3.2), the mentioned templates such as norm, and functions definition for the definition of all norms. As there are several roles and sub-roles in the system, the LHS part of the rule shows the role of the agent to whom the following normative command in the RHS applies. Recall that the assignment of roles to agents is a separate task which is done by protocols. Here we assume that this task is undertaken by MAS and our tool is informed the latest role assignments.

In the following, we provide similar examples of norm definitions in the domain of an auction application on the basis of Method 1.

For instance, the following rule shows an example of the main rules created by a legislator. This rule says “*Advertiser is not allowed to place a Bid during the Auction Time (between Start Time and End Time)*”. Advertiser is a sub-role of Seller who advertise an item for auction. In the following, we provided the definition of this rule.

```
(defrule placingBidForbidden
  ?roleFact <-( role (roleTitle Advertiser)(agentName ?x)(AtTime ?time))
  =>
  (bind ?startTime (getStartTime ?time))(assert (auctionStartTime (value
?startTime)))
  (assert (norm (status ToBeACTIVATED)(deoMode Frb) (act placeBid) (addressee
?x) (timeMode BETWEEN) (time (getStartTime ?time)
(getEndTime ?startTime))) ) )
```

As an example of enforcement norms, this rule says “*LowerBidder is punished by P1*”. LowerBidder is a sub-role of Bidder who put a bid lower than the current highest bid. P1 is an instruction code which should be run by internal

agents of the multiagent system. For description of this norm we used the templates of enforcementNorm.

```
(defrule wrongBidPunishment
  ?roleFact <-( role (roleTitle LowerBidder)(agentName ?x)(AtTime ?time))
=>
(assert (enforcementNorm (status PUNISHMENT) (addressee ?x) (EnfCode P1))) )
```

Here we provide the definition of our example of reward based on Method 1. The following norm says “*The reward R1 should be assigned to fastPayer*”. FastPayer is a sub-role of Winner who pays in 5 minutes time of winning the auction.

```
(defrule rewardForFast Payment
  ?roleFact <-( role (roleTitle fastPayer)(agentName ?x)(AtTime ?time))
=>
(assert (enforcementNorm (status REWARD) (addressee ?x) (EnfCode R1))) )
//where fastPaymentTime is 5 minutes after winning time.
```

7.3.2.3 General Rules

As mentioned, in addition to the norms created by the legislator of the system, the normative KB needs some general rules defined for the execution of norms based on the status of the norm, conditions of the norm, or time notions of the norm. These general rules are automatically added to the normative KB by our tool at system runtime.

These general rules are independent from the domain of application and also are independent from selection of either Method 1 or Method 2. These rules are conditional rules and are generated automatically by our tool. All general norms and violation detection norms have been listed in Appendix G. We formulated and defined all general norms based on tables we had produced and presented in Appendix H. These tables cover the various different cases of deontic modes and time notions.

We defined our general rules in two categories:

One of the main types of norms of general rules is those for *assisting the process of dynamic assignment of R&Rs and sanctions to external agents*. In such norms, the modification of the status of the norms is very important. Recall from Section 5.5.10, we defined the command mode of norms in the formalization of normative commands such that $CommandMode \in \{ToBeActivated, Activated, Deactivated, Fulfilled, Violated\}$. Then, in the template of norms, we considered a slot named “*status*” slot to represent this mode. The status of a norm is changed at runtime when such modifications of modes are provided by our general rules. The modification of modes can be summarized as follows:

- ToBeACTIVATED → ACTIVATED
- ACTIVATED → DEACTIVATED FULFILLED
- ACTIVATED → DEACTIVATED VIOLATED

The other types of norms in general rules of our middleware tool are *check norms*. In fact, these check norms detect when violations occur, and then label the norm with status VIOLATED. These norms also detect the reward or compensation cases.

In the following, we provide two examples of general rules. The first one is an illustration of general rules for changing the status of norms. The second example shows a general rule for detecting violation.

First Example: This norm detects when any obligation action is not fulfilled by the deadline. So if an obligation norm satisfies the status of the activated norm it should be deactivated and labeled as fulfilled. The description of this norm states that if an event occurs (such that *actor “y” does the act “x” at time “t”*) and this event matches an activated norm (which says *actor “y” is obliged to do act “x”*), it means that the norm is fulfilled and the status of the norm should be changed to DEACTIVATED mode and labeled as FULFILLED at the time immediately after occurrence of the event.

```
(defrule statusChangeForObligationNorm
  ?eventFact<-(event (act ?x)(actor ?y)(AtTime ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Obl)(act ?x)(addressee ?y))
=> (duplicate ?normFact (status DEACTIVATED FULFILLED)
      (timeMode AtTime) (time ?t) (retract ?normFact))
```

Here, we emphasize that in our general rules the status of all Activated norms is changed to Deactivated and labeled with Fulfilled or Violated. This task happens on two occasions: when the norm is executed so it will be labeled as Fulfilled (as defined in the above norm), or when a violation occurs in which case it will be labeled as Violated.

Second Example: The general rules of our middleware tool contain check norms as well. These check norms detect when violations occur, and then label the norm with status VIOLATED. Check norms are usually relevant to time.

For example, there is an activated norm stating, “*an obligation for doing an action before time tx*”; if the current time is “*tx*” and the norm is not fulfilled by the addressee of the norm, then a violation has occurred.

Such time-related rules mark norms as VIOLATED, if the current time is asserted to the fact base (showing that this time is one of the critical times) and there are still one (or more) activated norms in the fact base which have not been fulfilled by that critical time.

When the current time is asserted to the fact base as a fact, if there is any Activated norm related to this time in fact base, the left-hand side of the check norm is satisfied and check norm will be fired. In this case, the related fact should be marked as VIOLATED.

We again emphasize that enforcement norms (punishment or reward norm) are fired after detection of the violation or enactment of norm. In fact, by modification of the status of the norms, from activated to fulfilled or violated, the task of detection of violation or enactment is done. And the next stage

would be the activation of the enforcement norms as enforcement norms. The enforcement norms are defined by the legislator as described before.

In the following, we provide an example of check norms:

```
(defrule violatedObligationBefore

  ?CurTimeFact <- (currentTime (value ?t))

  ?normFact <- (norm (status ACTIVATED)(deoMode Obl)

                 (timeMode BEFORE)(time ?t))

=>

  (duplicate ?normFact (status DEACTIVATED VIOLATED)(timeMode AtTime)(time
  ?t))(retract ?normFact))
```

7.3.3 Design of the Simulator of Event /Action Handler

One of the main inputs of the system are events and actions which occur in the multiagent system at runtime. In order to focus on the techniques of dynamic assignment of R&Rs to agents, we have simply simulated these inputs, using the InputSimulator class. This class generates events/actions for the system as if they were runtime occurrences in the example auction system. These events/actions are passed to the tool with the format of “event” template defined in Section 7.3.2.1.1.

Recording the Time of Events: As we use a simulator to simulate occurrences of events and actions, when the occurred event/action is asserted to the Jess fact base, the time of occurrence is also added to the fact to show at what exact time

the event/action happened. Later, the $At(t)$ function will be described in order to represent the time of occurrence of the time.

7.3.4 Design of Timer

Time is another important issue in this tool. As mentioned before, norms are mostly time-related and they contain the notion of times using the temporal functions such as $before(t)$, $after(t)$ and $between(t1,t2)$ according to the grammar of the full normative language. In addition to these three functions there is another implicit time notion for the time of occurrences of event. As we use an event simulator, we add another function, namely $At(t)$, showing the time of event/action occurrence.

When a norm is activated, the values of time functions indicate the important times for the status of the norm. Such an important time may be a start time for activation of an obligation, prohibition, permission or right. Or it may be a deadline for an activated norm and after that time norm should be deactivated.

7.3.4.1 Time Management in R&R Allocator Tool

For recording of time and date and their management, we use a timestamping method. *“A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.”* [76]. Using a timestamp allows for easy comparison of different records and tracking progress over time.

The format of our timestamps is based on the IETF standard date syntax recognized by `java.util.Date` and also used by the Jess language for timestamps. The source of time for this standard is 12:00 AM on 1/Jan/1970 and contains both time and date.

In general, the process of time management in this tool has the following stages:

1. Events are recorded into the fact base the time of occurrences with a timestamp.
2. This timestamp is kept in the timer list of the application. The timer notices whenever the current system time matches the timestamp, when the current time (notified by timer), will be asserted to Jess fact base. The format of current time has been defined as “*currentTime*” template in Section 7.3.2.1.1.
3. The Jess inference engine will be activated by asserting the current time, then the facts related to the current time will be activated.

7.3.5 Design of User Interface

This tool has very simple user interface. For every agent that has joined to the system, the tool creates a frame. Then all the assignments of rights and responsibilities, of norms and assignment of sanctions relevant to this agent, are dynamically reported in the output frame. Whenever an agent leaves the system the associated frame will be apparently destroyed.

7.4 Summary

In summary, this chapter presented the analysis and design of a middleware tool which is an implementation of our proposed methods. This tool is generic, can be connected to any MAS intended to apply our methods, and is independent from the development details of that MAS. The tool is also

independent of the Method chosen to implement dynamic assignment of R&Rs, whether Method 1 or Method 2.

As is usual for application development, we began with an analysis exercise, analyzing the functionality of the tool and its inputs and outputs in Section 7.2; this was followed by an explanation of the design stage in Section 7.3.

One of the important tasks to design such a tool is the creation of the normative knowledge base which we have described in Section 7.3.2. Either of our two proposed methods can be used for creating this normative KB, depending on the features of the MAS. If the MAS has been created on the basis of a role hierarchy, Method 1 would be more appropriate for creating a normative KB than Method2; otherwise Method 2 would be more appropriate.

Furthermore, we motivated other important elements of the middleware tool design, including the design of the timer and the user interface.

Chapter 8

Implementation of the Middleware Tool

8.1 Introduction

In this chapter, we provide the implementation and testing of the development of the middleware tool which we developed to demonstrate the practical feasibility of our methods and of our architecture.

In Section 8.2, we provide general information on the implementation stage of the development process of this application. Then, in Section 8.3, we consider a scenario of runtime occurrences in the auction system for which we have already designed a normative KB in Chapter 7. On the basis of this scenario we

test both of our methods. In these tests, for each runtime occurrence, first, we represent the norms of normative knowledge base which is fired by this occurrence, and second, we show snapshots of the output of our application, which is the assignment of R&Rs or sanctions to an external agent of the normative auction system.

In Section 8.4 , we compare and evaluate the two methods. And finally in Section 8.5, we give a summary of this chapter.

We have attached some snapshots of the Java source of our application in Appendix 10.9.

8.2 Implementation

In the development process, the subsequent stage after analysis and design is implementation and testing. In this section, we provide some general description for implementation of this middleware tool.

Technically, we built this application using the Java language. The development environment we used for this middleware tool was NetBeans IDE version 6.1. We selected Java because our application is required to connect to any MAS and Java has the capability of compatibility for such a connection. The reason that we use the NetBeans environment for programming was that this software is freely available, is popular, and contains Integrated UML Tools to provide an environment for UML analysis and design; it also contains facilities for converting such designs to basic codes in Java. As the design of

this tool was not too complicated, using this UML tool was suitable for achievement of our design goals.

The inference engine we used for performing the task of reasoning was the Jess Rule Engine. We selected this inference engine, first, because it is rule based and appropriate for the norms we wanted to create in the normative knowledge base. The second reason is that Jess is based on Java and its Java APIs can be simply imported and applied in any Java program, as we used.

One of the other features of this implementation is we used a simple function for translating assigned norms (which are the outputs) to natural language. This feature is very helpful because understanding the natural language format of outputs is much easier than the Jess format of them.

8.3 Testing

Throughout this thesis, we have mostly used auction examples for illustration of our ideas. In order to test our application, once again the auction domain will be used. For testing, we should provide some inputs for application and then obtain the outputs of the application.

As mentioned before, the inputs of this tool are the normative KB rule base of the system designed by the legislator, and the runtime occurrences provided by MAS. We described how to create the rule base of the example auction in both methods in Chapter 7. We provided these normative KB in Appendix E and F. In this chapter, we simulate runtime occurrences (as the inputs from the MAS) and launch these inputs to the application for obtaining the outputs of our

application which are the results of dynamic assignment of R&Rs and sanctions.

However, before trying inputs and observing outputs, we provide the list of all norms in our auction system; we provide a scenario in which a number of different occurrences in the auction at runtime will happen.

8.3.1 Scenario

Here we present a scenario of different runtime occurrences in our auction system. We provide various runtime occurrences which contain different cases of events, actions, environmental events, entry and exit of agents and achievement of important times. In these examples, we make this scenario such that all types of legal modalities (including obligations, permissions, rights and prohibitions), all cases of enforcement modes (including punishment, reward and compensation) and all kinds of temporal functions (including *before(t)*, *after(t)*, *between(t1,t2)* and *at(t)*) are included.

The scenario of this auction is as follows: We suppose that *Sarah*, *David*, *Mari*, and *Nina* are members of this auction system. In this session, the *Seller* is *Sarah*. *David* and *Nina* have already logged into the system as *Buyer*. As an example of environmental variables, we suppose every member has a Feedback variable. We initialize feedbacks of members as follows:

hasFeedback(Sarah,0), *hasFeedback(David, 0)*, *hasFeedback(Mari, -2)*,
hasFeedback(Nina, 1).

As an identification of the auction, the name of the auction is *Auction_1*, and the item for the auction is *Gold Watch*. The starting time of the auction is “10:00:00 2/1/2008”. The ending time of the auction is “11:00:00 2/1/2008”.

The current Bid of the Gold Watch is *10GBP*. We assume that the current time is “*10:15 2/1/2008*”.

Here, we list the following runtime occurrences for this scenario, and in the next Sections we apply these occurrences for Method 1 and Method 2.

RO-1) An Action: David places a bid at 10:15 2/1/2008. (We note that buyer first presses a button to place a bid, then he will enter the bid.)

RO-2) An Action: David places the higher bid of 25GBP at 10:15 2/1/2008.

RO-3) An agent enters: Mari joins to the auction at 10:20 2/1/2008.

RO-4) An Action: Mari places a bid at 10:22 2/1/2008.

RO-5) An Action: Mari places a Lower Bid of 22 GBP.

RO-6) An env. Event: Enforcer decreases Feedback of Mari at 10:22 2/1/2008. (Mari’s feedback is -3).

RO-7) An action: Nina places a bid at 10:30 2/1/2008.

RO-8) An Action: Nina places a higher bid of 30GBP at 10:30 2/1/2008.

RO-9) An action: Sarah places a bid at 10:35 2/1/2008.

RO-10) A deadline: Current time is 11:00 2/1/2008.

RO-11) An env.Event: Nina wins the auction at 11:05 2/1/2008.

RO-12) An action: Nina pays the price at 11:05 2/1/2008.

RO-13) An env.Event: Nina’s feedback is 2 at 11:05 2/1/2008.

RO-14) A deadline: Current time is 11:00 9/1/2008 (the next Week of starting time).

RO-15) An action: Nina claims for compensation at 11:10 9/1/2008.

RO-16) An Env. event: Enforcer asks internal agents to perform the compensation for Nina (gets 50% of Nina's money back at 11:11 9/1/2008).

8.3.2 Method 1

As mentioned before, Method 1 is based on role hierarchy such that rules of the Jess rule base are defined for several roles and sub-roles of this role hierarchy. Recall that the rules of the Jess rule base have the format of LHS=>RHS. The LHS of the rules of the normative KB of this method is the occurrence of the assignment of a new role to an agent. And the RHS of this normative KB is the normative command.

In this method, we assumed that dynamic assignment of roles to agents is provided by the MAS and the result of this assignment is reported to our tool and subsequently is asserted to the Jess fact base by this tool. Therefore, whenever a new role assignment occurs in MAS, this event is reported to our tool, this event is recorded to the Jess fact base, followed by the Jess inference engine performing a reasoning task. By Jess reasoning, one (or more) rule(s) may be fired. The result of this reasoning will be collected and analyzed by our tool to check which new norms have been activated, which previously activated norms has been deactivated and which norms have been violated or fulfilled. Consequently, the result of new assignment of rights, responsibilities or sanctions will be released to the external agents of the auction system as the output of this application.

In the following, we provide the inputs for testing the outcomes of our tool on the basis of this method. We use the above scenario for this assessment. As mentioned, this application has two main inputs: the normative KB (created at design time) and event/actions (which come from the MAS at runtime). The full normative KB of this example is listed in Appendix E. This KB should be created based on a role hierarchy. In this case, we created the structured role hierarchy presented in Figure 10, and then applied this hierarchy for creating the normative KB.

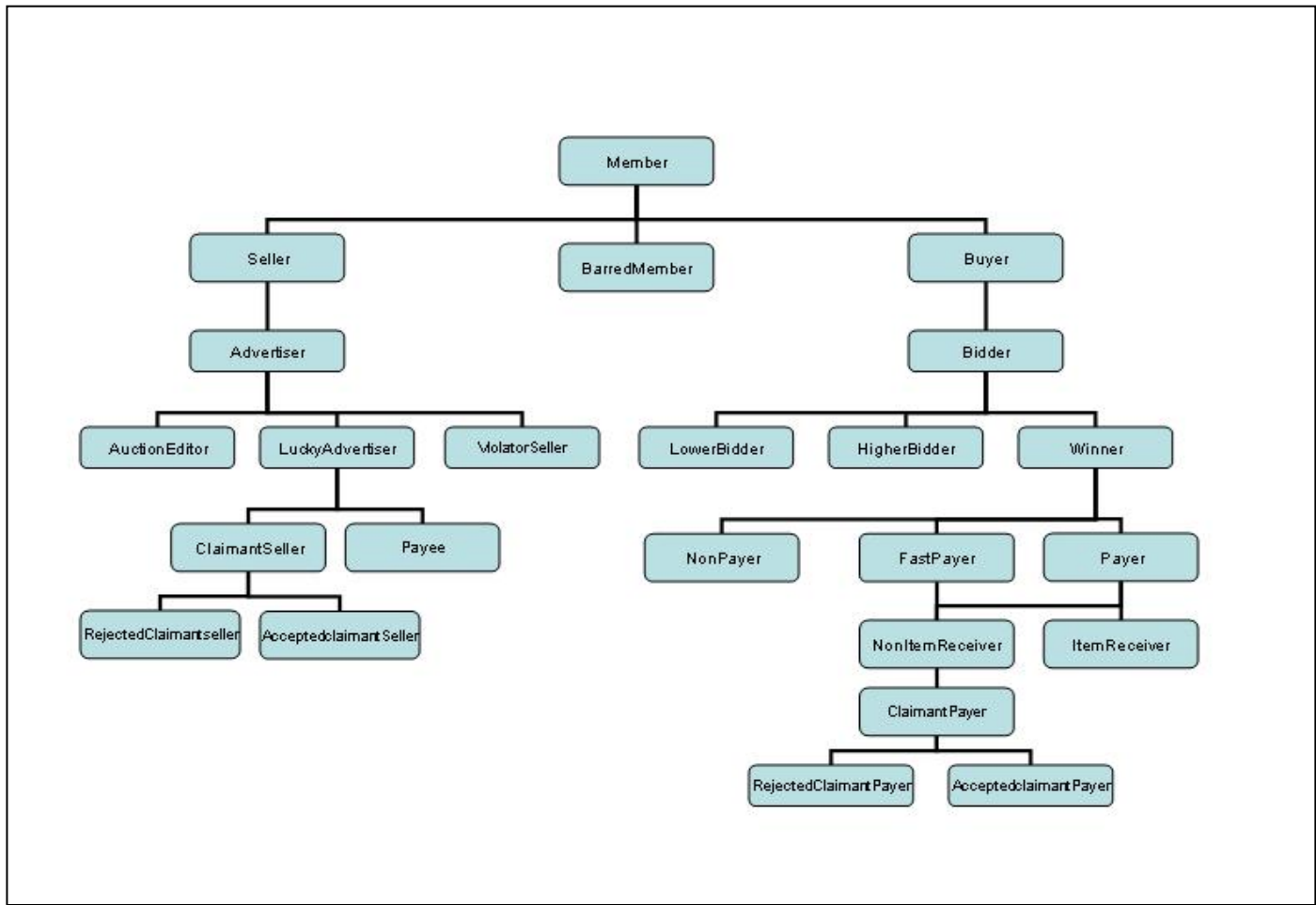


Figure 10-The role hierarchy of auction system for using in Method 1

We suppose that the current status of the auction is as follows:

{Sarah, David, Mari, Ali, Nina } ∈ Member

Sarah: Advertiser , David,Nina: Buyer

hasFeedback(Sarah,0), hasFeedback(David, 0), hasFeedback(Mari, -2),
hasFeedback(Nina, 1)

hasStartTime (Auction_1, “10:00 2/1/2008”)

CurrentTime=“10:15 2/1/2008”

We suppose that so far the following assignment of roles have been undertaken:

Current Stage: In the following:

role (roleTitle advertiser)(agentName Sarah)

role (roleTitle buyer)(agentName David)

role (roleTitle buyer)(agentName Nina)

As the result of the above assertions of facts, the following norms from the normative KB are fired:

;;R2 :Advertiser is forbidden to place Bid in the Auction Time (between Start Time and End Time).

(defrule placingBidForbidden

 ?roleFact <-(role (roleTitle advertiser)(agentName ?x)(AtTime ?time))

=> (assert (norm (status ToBeACTIVATED) (deoMode Frb) (act placeBid)

 (addrsee ?x) (object ?item ?auction) (timeMode BETWEEN)

 (time ?startTime (getEndTime ?startTime)))))

;;R3 : Advertiser is permitted to edit the auction before Start Time.

(defrule editAuctionPermission

 ?roleFact <-(role (roleTitle advertiser)(agentName ?x)(AtTime ?time))

=> (assert (norm (status ACTIVATED) (deoMode Prm) (act editAuction)

 (addrsee ?x) (object ?item ?auction) (timeMode BEFORE) (time ?startTime))))

;;R4 : Advertiser is forbidden to edit the auction after Start Time.

(defrule editAuctionForbidden

 ?roleFact <-(role (roleTitle advertiser)(agentName ?x)(AtTime ?time))

=>(assert (norm (status ToBeACTIVATED) (deoMode Frb) (act editAuction)

 (addrsee ?x) (object ?item ?auction) (timeMode AFTER) (time ?startTime))))

;;R5 :Buyer is permitted to place a Bid between Start Time and the End Time.

(defrule placingBidPermission

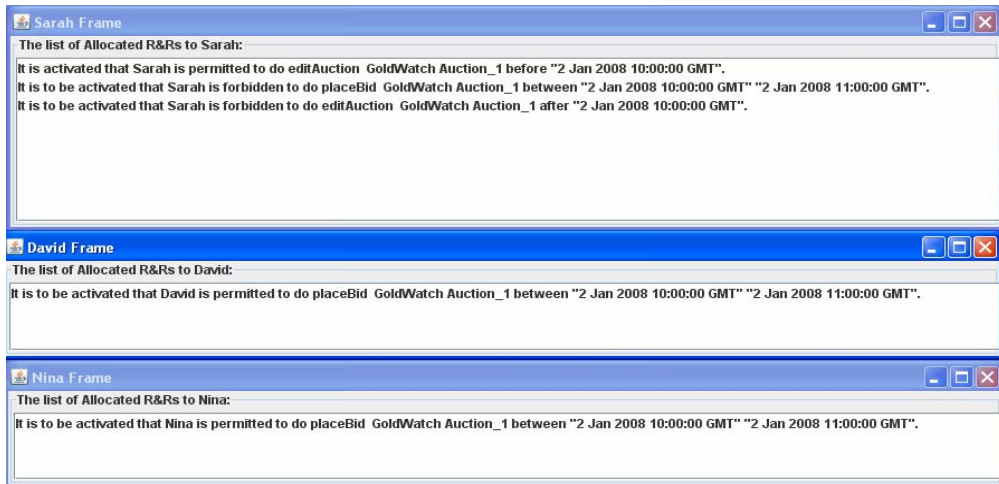
 ?roleFact <-(role (roleTitle buyer)(agentName ?x)(AtTime ?time))

=> (assert (norm (status ToBeACTIVATED) (deoMode Prm) (act placeBid)

 (addrsee ?x)(object ?item ?auction)(timeMode BETWEEN)

 (time ?startTime (getEndTime ?startTime))))))

So far three agents joined in the auction: Sarah, David and Nina. The assigned R&Rs to external agents are shown by our tool as follows:



Suppose that the current time is “10:00:00 2 Jan 2008”, which is the Start Time of the auction. As this time is one of the important times (i.e, that our application creates a timer for), the timer will notify this time. Then, Jess will be activated for performing a reasoning task which here leads to fire a general rule(GR5). This rule changes the status of ToBeActivated norms to ACTIVATED when the time achieves.

```

;GR5
(defrule statusChangeToBeActivatedAfter
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ToBeACTIVATED)(timeMode AFTER)(time ?t))
=>(duplicate ?normFact (status ACTIVATED))(retract ?normFact)

```

The following figure shows how ToBeActivated norms have been activated for Sarah, David and Nina at the start time of the auction.



After that, we input the events - provided in our scenario - as runtime occurrences of this auction system and demonstrate how dynamic assignment of R&Rs and sanctions will be provided.

RO-1) An Action: David is *bidder* at “10:15:12 2/1/2008”.

We consider that the MAS inputs the following events to our tool:

```
event (act placeBid)(actor David)(object GoldWatch Auction_1)

role (roleTitle bidder)(agentName David)
```

By this runtime occurrence, two rules of normative KB are fired: a general rule (GR2) which shows that a permitted action (placing a bid) has been took place. And a domain-related rule (R6) which says the bidder is obliged to place a higher bid.

```
;;;GR2
(defrule statusChangeForPermissionNorm
  ?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p )(AtTime ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Prm)(act ?x)(addressee ?y)(object ?z ?p
  ))
  => ( assert (norm (status FULFILLED)(deoMode Prm)(act ?x)(addressee ?y)
  (object ?z ?p ) (timeMode AtTime)(time ?t))) )
;;;R6 :The obligation for placing Higher bid is fulfilled for higherBidder.
(defrule placingHigherBidObligation
  ?roleFact <-( role (roleTitle bidder)(agentName ?x)(AtTime ?time))
  => (assert (norm (status ACTIVATED) (deoMode Obl) (act placeHigherBid)
  (addressee ?x)(object ?item ?auction)(timeMode BETWEEN)
  (time ?startTime (getEndTime ?startTime))))))
```

The following snapshot shows the assignment of these norms to David.



RO-2) Action: David is *HigherBidder* of 25GBP at “10:15 2/1/2008”.

When David puts his bid, the MAS evaluates the value of his bid; as this value (25 GBP) is greater than the current bid, the role of *higherBidder* will be assigned to David and the following event will be reported to our application by the MAS:

role (roleTitle higherBidder)(agentName David)

By inserting this event to the Jess fact base, and following activation of the Jess inference engine, the following rule from the Jess rule base (normative KB) is fired:

```
;;; R7 :The obligation for placing Higher bid is fulfilled for higherBidder.
(defrule placingHigherBidfulfilled
  ?roleFact <-( role (roleTitle higherBidder)(agentName ?x)(AtTime ?time))
=>(assert (norm (status FULFILLED) (deoMode Obl) (act placeHigherBid)
  (addressee ?x)(object ?item ?auction)(timeMode AtTime)(time ?time)) ) )
```

The following snapshot shows that this obligation is fulfilled which is reported to David.



RO-3) An agent enters: Mari joins to the auction.

At this time, Mari joins the auction and the MAS reports it to our application.

AgentJoint (agentName Mari)

In this case, our application creates a new frame for Mari to report her rights and responsibilities.

RO-4) An Action: Mari is *buyer* at “10:20 2/1/2008”.

Mari opts to be a buyer, so the MAS assigns the role of *Buyer* to Mari and reports this assignment to our application.

role (roleTitle buyer)(agentName Mari)

As this case is similar to the beginning step for the initializing occurrences we do not repeat it here.

RO-5) An Action: Mari is *bidder* at “10:20:8 2/1/2008”.

At this time, Mari wants to place a bid, so MAS assigns the role of *bidder* to Mari, and also reports the following events to our tool.

event (act placeBid)(actor Mari)(object GoldWatch Auction_1)

role (roleTitle bidder)(agentName Mari)

The result of this occurrence is similar to the result of RO1, the rules of GR2 and R6 are fired. The following snapshot shows that the right of placing a bid is fulfilled, and also it shows that Mari is responsible for placing a higher bid.



RO-6) An Action: Mari is lowerBidder of 22GBP at “10:21 2/1/2008”.

Suppose that Mari places 22 GBP which is lower than the current bid. So the MAS assigns the role of *Lower Bidder* to Mari and then the MAS reports the following events to our tool:

event (act placeLowerBid)(actor Mari)(object GoldWatch Auction_1 22GBP)

role (roleTitle lowerBidder)(agentName Mari)

As the result of this occurrence, the following rules will be fired. VR5 is a general rule which detects the violation against obligations that should occur in between two specified times. ENR1 is a punishment rule for the lower bidders. By executing this rule, the addressee of the norm will be informed about his/her punishment.

```

;;;VR5
(defrule violatedObligationBetween
  ?CurTimeFact <-(currentTime (value ?tx))
  ?normFact<-(norm (status ACTIVATED)(deoMode Obl)(timeMode BETWEEN)(time
?t ?tx))
=>(duplicate ?normFact (status DEACTIVATED VIOLATED)
  (timeMode AtTime)(time ?tx))(retract ?normFact))

;;;ENR1 the punishment for wrong bids
(defrule wrongBidPunishment
  ?roleFact <-( role (roleTitle lowerBidder)(agentName ?x)(AtTime ?time))
=> (assert (enforcementNorm (status PUNISHMENT) (addressee ?x)
  (EnfCode P1:toDecreaseFeedbackValue)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P1))))

```

Rule VR5 shows that a violation occurred, as it changes the status of the rule to VIOLATED. Then rule ENR1 specifies the punishment for lowerBidder by a code. When this norm fires, NormAnalyzer detects “EXECUTE” and sends this command to Enforcer for executing. Enforcer passes the code (here P1) of this internal command to the relevant internal agent of the MAS. This code has already been defined for internal agents. In this rule, P1 is the code for decreasing the number of feedbacks of the violator agent. This code is recognized by internal agents and should be executed by them.

The following figure shows the assignment of this punishment to Mari.



RO-7) An Env. Event: Enforcer increases the Neg. feedback of Mari at “10:22 2/1/2008”.

As the result of the previous event, the value of Mari’s feedback (which was -2) has been decreased to -3. The internal agents of the MAS report this change to our application.

feedback (actor Mari)(value -3)

When this environmental event is reported to our application, the following rule will be activated.

```

;;; ENR6 :If agent has feedbacks=-3, agent is forbidden to auctionjoint.
(defrule agentJointprohibition
  ?feedbackFact<-(feedback (actor ?x)(value -3)(AtTime ?time))
  =>(assert(norm (status ACTIVATED) (deoMode Frb) (act auctionJoint) (addressee ?x)(time)
    (assert( role (roleTitle BarredMember)(agentName ?x)(AtTime ?time)))
    (assert (enforcementNorm (status EXECUTE) (addressee ?x)
      (EnfCode barredMember))))))

```

According to rule ENR6, whenever the number of feedbacks of an agent is -3, it is forbidden for him/her to join to the auction anymore and the name of this agent will be added to the list of barredMembers. Subsequently, internal agents

will check the legality of each agent as it tries to login to the system to not be a *barredMember*.

As the following snapshot shows, Mari is forbidden to join to the auction after her feedbacks reach to -3. In fact, this is a sanction or enforcement norm which has been assigned to Mari by our tool.



RO-8) An action: Nina is *bidder* at “10:30 2/1/2008”.

Now suppose that Nina wants to place a bid. So MAS reports the following events to our tool:

event (act placeBid)(actor Nina)(object GoldWatch Auction_1 30GBP)

role (roleTitle bidder)(agentName Nina)

The similar steps for David in RO1 take place for Nina (in this runtime occurrence) rules GR2 and R6 are fired. So we do not repeat this stage again.

RO-9) An action: Nina is *higherBidder* of 30GBP at 10:29:51.

MAS evaluates the value of Nina's bid. As this bid (30 GBP) is higher than the current bid, MAS assigns the role of higherBidder to Nina, and then reports this assignment to our tool as follows:

role (roleTitle higherBidder)(agentName Nina)

This stage is similar to RO2, which rules R7 is fired. The outcome of our tool is shown in the following snapshot.



RO-10) An action: Sarah is *ViolatorSeller* at "10:30 2/1/2008".

Now suppose that Sarah places a bid. The MAS report this event to our tool as follows:

event (act placeBid)(actor Sarah)(object GoldWatch Auction_1)

At runtime, when Sarah places the bid, our application detects her action as a violation based on rule VR6, because this action has been forbidden by R2 (this rule has been activated after start time of the auction). Then based on this rule, the role of violatorSeller will be assigned to Sarah, as follows:

role (roleTitle ViolatorSeller)(agentName Sarah)

Then, based on ENR5-B, she will be punished. The punishment P2 is known and executed by internal agents. In the following we list these norms:

```

;VR6
(defrule violatedForbiddenBetween
  ?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?tx))
  ?normFact<-(norm (status ACTIVATED)(deoMode Frb)(act ?x)
    (addressee ?y)(object ?z ?p)(timeMode BETWEEN)(time ?t1 ?t2))
  (test (> tx t1))
=>(duplicate ?normFact (status DEACTIVATED VIOLATED)
  (timeMode AtTime)(time ?tx))(retract ?normFact))

;ENR5-A
(defrule violatorSeller
  ?normFact<- (norm (status VIOLATED)(deoMode Frb) (act placeBid) (addressee ?x)
    (object ?item ?auction )(timeMode AtTime)(time ?time)))
=>(assert(role (roleTitle violatorSeller)(agentName ?x)(AtTime ?time)))

;ENR5-B
(defrule sellerBidderPunishment
  ?roleFact<-(role (roleTitle violatorSeller)(agentName ?x)(AtTime ?time))
=> (assert (enforcementNorm (status PUNISHMENT) (addressee ?x)
  (EnfCode P2:toDecreaseFeedbackValueOfSeller))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P2))) )

```



RO-11) A deadline: Current Time is “11:00 2/1/2008”.

This event is notified by the timer of our tool which notifies that the current time is 11:00 (as a deadline shows the ending time of the auction). Therefore those rules which are sensitive to this deadline will be fired, for example, Rule GR10, which leads to deactivate the permission of the buyers to place bids.

```

;GR10
(defrule retractPermissionBetween
  ?CurTimeFact <-(currentTime (value ?tx))
  ?normFact<-(norm (status ACTIVATED)(deoMode Prm)
    (timeMode BETWEEN)(time ?t ?tx))
  => (duplicate ?normFact (status DEACTIVATED)(timeMode AtTime)
    (time ?tx))(retract ?normFact)

```

The following snapshot shows how this norm is executed for David and Nina. Based on GR10, the norm - denoted the permission of David and Nina for placing bids - is deactivated.



RO-12) An action: Nina is *winner* at “11:00 2/1/2008”.

After the ending time of the auction, the MAS assigns the role of *Winner* to Nina and *luckyAdvertiser* to Sarah, and then reports the following role assignments to our tool.

role (roleTitle Winner)(agentName Nina)

role (roleTitle luckyAdvertiser)(agentName sarah)

As the result of these assertions, the following rules R9 and R10 are fired.

```
;;;R9, R10 : LuckyAdvertiser has the right to receive the money from the Winner.

(defrule receivePaymentRightAndPaymentObligation
  ?roleFact1 <-( role (roleTitle luckyAdvertiser)(agentName ?l))
  ?roleFact2 <-( role (roleTitle winner)(agentName ?w))
  =>(assert (norm (status ACTIVATED) (deoMode Right) (act recievePayment)
    (addressee ?l)(benef ?w)(object ?item ?auction)(timeMode BEFORE)
    (time (getPaymentDueTime ?startTime)) ) )
  (assert (norm (status ACTIVATED) (deoMode Obl) (act pay)
    (addressee ?w)(benef ?l) (object ?item ?auction) (timeMode BEFORE)
    (time (getPaymentDueTime ?startTime)) ) ) )
```

The following figure shows the assignment of the right of receiving payment to Sarah, and also it shows the assignment of the responsibility of payment to Nina.



RO-13) An action: Nina is *fastPayer* at “11:05 2/1/2008”.

Suppose that Nina pays the payment of the item very soon. Then the MAS will assign the role of *fastPayer* to Nina. This assignment and the occurrence of relevant events will be notified to our application.

role (roleTitle FastPayer)(agentName Nina)

role (roleTitle Payee)(agentName Sarah)

event (act pay)(actor Nina)(object GoldWatch Auction_1 30GBP)

*event(act recievePayment)(actor Nina)(object GoldWatch Auction_1
30GBP)*

In this system, *fastPayer* is a *Winner* who pays within 10 minutes of the ending time of the auction and a reward has been defined for such payer. The occurrence of the above event leads to the execution of the following norms: GR1 deactivates the obligation of any action already undertaken. GR3 deactivates the right of any action undertaken. R11 activates the right of the *fastPayer* for receiving the item. ENR2 is the norm which specifies the reward considered for fastPayers. This reward is the increment of the fastPayer's feedback and is executed by internal agents using the code (R11).

```

;;GR1
(defrule statusChangeForObligationNorm
  ?eventFact<-(event (act ?x)(actor ?y)(AtTime ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Obl)(act ?x)(addressee ?y))
=>( duplicate ?normFact (status DEACTIVATED FULFILLED)
    (timeMode AtTime)(time ?t) (retract ?normFact))

;;GR3
(defrule statusChangeForRightNorm
  ?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p )(AtTime ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Right)(act ?x)
    (addressee ?y)(object ?z ?p ))
=>( assert (norm (status FULFILLED)(deoMode Right)(act ?x)(addressee ?y)
    (object ?z ?p ) (timeMode AtTime)(time ?t)) ) (retract ?normFact ) )

;;R12 : FastPayer has the right to get the item from the luckyAdvertiser.
(defrule getItemRight
  ?roleFact1 <-( role (roleTitle Payee)(agentName ?l))
  ?roleFact2 <-( role (roleTitle fastPayer)(agentName ?p)(AtTime ?time))
=>( assert (norm (status ACTIVATED) (deoMode Right) (act getItem) (addressee ?p)
    (benef ?l)(object ?item ?auction ?bid) (timeMode BEFORE)
    (time (getSendingDueTime ?startTime)) ) ) )

;;ENR2
(defrule rewardForFastPayment
  ?roleFact <-( role (roleTitle fastPayer)(agentName ?x)(AtTime ?time))
=>(assert (enforcementNorm (status REWARD) (addressee ?x)
  (EnfCode R1:toIncreaseFeedbackValue)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode R1))))

```

The following snapshot shows the fulfilment of Nina's responsibility for the payment. It also shows *the assignment of the right of receiving the item* to Nina

and *the assignment of a reward (an enforcement norm)* to Nina.



RO-14) A deadline: Current Time is “10:00 9/1/2008” (Next week).

This event notifies that the current time is “10:00:00 9/1/2008” (7 days after the starting time of the auction) which is a deadline for sending the item by Sarah as a *luckyAdvertiser*. Therefore, at this time our application checks the status of sending the item by the following rule (VR1). As the norm of obligation for sending the item is still ACTIVATED and not FULFILLED, the rule VR1 will be fired at this time, which shows this norm is violated.

```

;VR1
(defrule violatedObligationBefore
  ?CurTimeFact <-(currentTime (value ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Obl)(timeMode BEFORE)
    (time ?t))
=> (duplicate ?normFact (status DEACTIVATED VIOLATED)
    (timeMode AtTime)(time ?t))(retract ?normFact)

```

The following figure shows this norm assignment:



RO-15) An action: Nina is *NonItemReciver* at “10:00 9/1/2008”.

As the result of the violation of *seller* to send the item, the next role assignment is as follows:

role (roleTitle NonItemReciver)(agentName Nina)

The assertion of this fact to our tool the following rule will be activated:

```

;ENR3
(defrule rightToClaimForCompensation
  ?roleFact <-( role (roleTitle NonItemReciver)(agentName ?x)(AtTime ?time))
  =>
  (assert (norm (status ACTIVATED) (deoMode Right) (act claim)(addressee ?x)(object
?item ?auction)(timeMode AFTER) (time ?time)))

```

This rule assigns the right of claim to Nina, as shown in the following snapshot.



RO-16) An action: Nina is *ClaimantPayer* at 11:10 9/1/2008.

Suppose that Nina claims, then MAS will report the following role assignment to for Nina:

role (roleTitle ClaimantPayer)(agentName Nina)

By assertion of this input to our application, the following rule will be activated.

```
;ENR4
(defrule compensationForClaimantPayment
  ?roleFact <-( role (roleTitle claimantPayer)(agentName ?x)(AtTime ?time))
  ?normFact <-(norm (status ACTIVATED) (deoMode Right) (act claim)(addressee
?x))
=> (assert (enforcementNorm (status COMPENSATION) (addressee ?x)
  (EnfCode C1:compensationClaimAccepted)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode C1)))
  (assert (role (roleTitle AcceptedClaimantPayer)(agentName ?x)(AtTime ?time))) )
```

By firing this rule, internal agents are responsible to execute the enforcement code of C1 which is related to such compensation. Then the new assignment is performed for the agent as AcceptedClaimantPayer.


```

;;R1 :Seller is permitted to advertise an item.
(defrule advertiseItemPermission
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  => (assert (norm (status ACTIVATED) (deoMode Prm) (act advertiseItem)
    (addressee ?x)) ))

;;R4 : If Seller advertise an item, Seller is permitted to edit the auction before start time of
the auction
(defrule editAuctionPermission
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act advertiseItem)(actor ?x)(object ?item ?auction ) (AtTime ?time))
  => (assert (norm (status ACTIVATED) (deoMode Prm) (act editAuction)
    (addressee ?x) (object ?item ?auction ) (timeMode BEFORE) (time ?startTime ) ) )

;;R3 : If Seller advertised an item, Seller is forbidden to place Bid during the Auction Time (t
(defrule placingBidForbidden
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event<-(event (act advertiseItem)(actor ?x)(object ?item ?auction ) (AtTime ?time))
  =>(assert (auctionStartTime (value ?startTime)))
    (assert (norm (status ToBeACTIVATED) (deoMode Frb) (act placeBid) (addressee
      ?x) (object ?item ?auction) (timeMode BETWEEN) (time ?startTime
        (getEndTime ?startTime)) ) ) )

;;R5 : Seller is forbidden to edit the auction after Start Time.
(defrule editAuctionForbiden
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act advertiseItem)(actor ?x)(object ?item ?auction ) (AtTime ?time))
  => (assert (norm (status ToBeACTIVATED) (deoMode Frb) (act editAuction)
    (addressee ?x) (object ?item ?auction ) (timeMode AFTER) (time ?startTime)) )

;;R6A : Buyer is permitted to place a Bid between the Start Time and the End Time.
(defrule placingBidPermissionBeforeSTime
  ?roleFact <-( role (roleTitle buyer)(agentName ?x)(AtTime ?time))
  ?sFact<-(auctionStartTime (value ?startTime)) (test (< ?time ?startTime))
  => (assert (norm (status ToBeACTIVATED) (deoMode Prm) (act placeBid)
    (addressee ?x)(timeMode BETWEEN) (time ?startTime(getEndTime ?startTime)))) )

```

Currently there are three agents in the MAS. The result would be as follows:



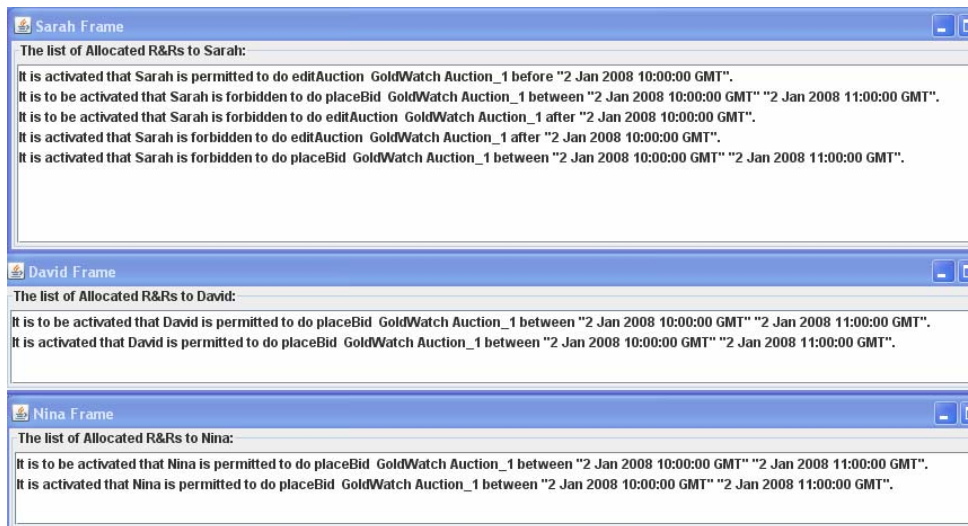
Suppose that the current time is “10:00:00 2 Jan 2008”, which is the Start Time of the auction. As this time is one of the important times (that our application creates a timer for), the timer will be activated at this time, followed by activation of the reasoning task which fires two general rules (GR5 and GR6). The following rule changes the status of ToBeActivated norms to ACTIVATED when the time achieves.

```

;GR5
(defrule statusChangeToBeActivatedAfter
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ToBeACTIVATED)(timeMode AFTER)(time ?t))
=>(duplicate ?normFact (status ACTIVATED))(retract ?normFact)
;GR6
(defrule statusChangeToBeActivatedBetween
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ToBeACTIVATED)(timeMode BETWEEN)(time ?t ?t2))
=>(duplicate ?normFact (status ACTIVATED))(retract ?normFact)

```

The following figure shows that ToBeActivated norms have been activated for Sarah, David and Nina.



RO-1) An Action: David places a bid at “10:15:00 2/1/2008”.

Suppose that the following event is reported to our tool by MAS.

event (act placeBid)(actor David)(object GoldWatch Auction_1)

By this runtime occurrence, two rules are fired: a general rule (GR2) which shows that a permitted action (placing a bid) has been took place. And a domain-related rule (R6) which says the bidder is obliged to place a higher bid. (First, buyer presses a button to place a bid, then he will enter the bid.)

```

;;;GR2
(defrule statusChangeForPermissionNorm
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p )(AtTime ?t))
?normFact<-(norm (status ACTIVATED)(deoMode Prm)(act ?x)(addressee ?y)(object ?z ?p
))
=> ( assert (norm (status FULFILLED)(deoMode Prm)(act ?x)(addressee ?y)
(object ?z ?p ) (timeMode AtTime)(time ?t))) )

;;;R7
(defrule placingHigherBidObligation
?roleFact<-( role (roleTitle buyer)(agentName ?x)(AtTime ?time))
?sFact<-(auctionStartTime (value ?startTime))
?normFact<-(norm (status ACTIVATED)(deoMode Prm)(act placeBid)
(addresssee ?x))
?event <-(event (act placeBid)(actor ?x)(object ?item ?auction )(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Obl) (act placeHigherBid) (addressee
?x) (object ?item ?auction )(timeMode BETWEEN) (time ?time (getEndTime ?startTime))) )
)

```

The following snapshot shows the effect on our tool of this runtime occurrence. It shows that the permission of placing a bid is fulfilled by David and also an obligation has been assigned to David for placing a higher bid.



RO-2) An Action: David places a higher bid of 25GBP at “10:16:46 2/1/2008”.

Now suppose that MAS reports the occurrence of the following event to our tool:

event (act placeHigherBid)(actor David)(object GoldWatch Auction_1 25)

```
;R8 :If Buyer place a Bid, and the bid is a higher bid, the act of placehigherBidder.  
(defrule placingBid  
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))  
  ?event <-(event (act placeHigherBid)(actor ?x)(object ?item ?auction )  
    (AtTime ?time))  
=>(assert (norm (status FULFILLED) (deoMode Obl) (act placeHigherBid)  
  (addressee ?x) (object ?item ?auction ?bid)(timeMode AtTime)(time ?time))) )
```

As a result of this assertion the following rules will be fired:

Then our tool reports that this obligation has been fulfilled, as the following snapshot shows:



RO-3) An agent enters: Mari joins to the auction.

Now we suppose that Mari joins the auction system. So MAS reports her entrance to our tool.

AgentJoint (agentName Mari)

In this case, our application creates a new frame for Mari to report her rights and responsibilities.

RO-4) An Action: Mari is *buyer* at “10:20 2/1/2008”.

Mari chooses to be a buyer, so MAS reports this occurrence to our tool:

role (roleTitle buyer)(agentName Mari)

Here we do not repeat this step, because they are similar to the steps for the initializing occurrences.

RO-5) An Action: Mari places a bid at “10:20:10 2/1/2008”.

Mari places a bid and MAS reports this event to our tool as follows:

event (act placeBid) (actor Mari)(object GoldWatch Auction_1)

The result of this occurrence is similar to RO1, which GR2 and R7 are fired, so we do not repeat this step here.

RO-6) An Action: Mari places a lower bid at “10:21:24 2/1/2008”.

Suppose that this event happens and the MAS reports the following role assignment to our application:

*event (act placeLowerBid)(actor Mari)(object GoldWatch Auction_1
22GBP)*

As the result of this occurrence, the following rules will be fired:

```
;R9 :If Buyer place a Lower Bid, Buyer is violated the obligation of placing higher bid.
(defrule placingHigherBidObligation
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))
  ?event <-(event (act placeLowerBid)(actor ?x)(object ?item ?auction ?bid )
    (AtTime ?time))
=> (assert (norm (status VIOLATED) (deoMode Obl) (act placeHigherBid)
  (addressee ?x)(object ?item ?auction ?bid )(timeMode AtTime)(time ?time)) ) )
;;;;;ENR1 the punishment for wrong bids
(defrule wrongBidPunishment
  ?normFact<-(norm (status VIOLATED)(deoMode Obl)(act placeHigherBid)
    (addressee ?x))
=>(assert (enforcementNorm (status PUNISHMENT)(addressee ?x)
  (EnfCode P1:toDecreaseFeedbackValueOfBuyer)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P1))))
```

The rule R9 shows that a violation occurred, as it changes the status of the rule to VIOLATED. Then rule ENR1 specifies the punishment for the person placed lower bid by a code. When this norm is fires, *NormAnalyzer* detects “EXECUTE” and commands to Enforcer for executing. Enforcer passes the code (here P1) of this internal command to the relevant internal agent of the MAS. This code has already been defined for internal agents. In this rule P1 is decreasing the number of feedbacks of the violator agent.

As the following snapshot shows the dynamic assignment of sanction to Mari is assigning P1: decreaseFeedbackValueOfBuyer.



RO-7) An Env. Event: Enforcer increases the Neg. feedback of Mari at 10:22 2/1/2008.

According to the previous punishment, the value of Mari’s feedback has been changed. So MAS reports this change to our tool.

feedback (actor Mari)(value -3)

When this environmental event reports to our application, the following rule will be activated.

```

;ENR6 : If buyer has feedbacks=-3, Buyer is forbidden to join to the auction anymore.
(defrule agentJointProhibition
  ?feedbackFact<-(feedback (actor ?x)(value -3)(AtTime ?time))
=>(assert(norm (status ACTIVATED) (deoMode Frb) (act auctionJoint) (addressee ?x)
  (timeMode AFTER)(time ?time)))

```

This norm specifies that whenever the number of feedback of an agent is -3, it is forbidden for that agent to join to the auction anymore and the name of this agent will be added to the list of barredMembers. Subsequently, internal agents will check the legality of agents as they try to login to the system.



As this figure shows, Mari is forbidden to join to the auction after her feedback value reaches to -3.

RO-8) An action: Nina places a bid at “10:29:00 2/1/2008”.

Nina places a bid, so MAS reports this event as follows:

event (act placeBid)(actor Nina)(object GoldWatch Auction_1)

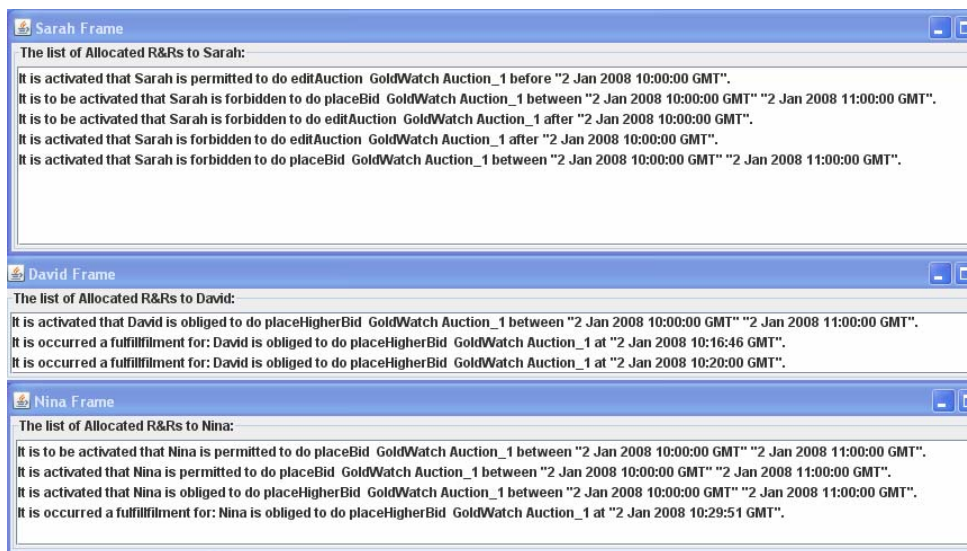
Here, similar to the first runtime occurrence, RO1, rules GR2 and R7 are fired. So we do not repeat this stage.

RO-9) An action: Nina places a higher bid of 30GBP at “10:29:50 2/1/2008”.

Nina places a higher bid, so MAS reports this event as follows:

event (act placeHigherBid)(actor Nina)(object GoldWatch Auction_1 30GBP)

This stage is similar to RO2, which rule R8 is fired. The outcome of our application is shown in the following snapshot.



RO-10) An action: Sarah places a bid at “10:30:21 2/1/2008”.

Suppose that Sarah places a bid. So MAS reports this event as follows:

event (act placeBid)(actor Sarah)(object GoldWatch Auction_1)

In runtime, when Sarah places the bid, our application detects her action as a violation, based on rule VR6. Then based on ENR5 she will be punished. The punishment P2 is known to and executed by internal agents.

```
;;;VR6
(defrule violatedForbiddenBetween
  ?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?tx))
  ?normFact<-(norm (status ACTIVATED)(deoMode Frb)(act ?x)
    (addressee ?y)(object ?z ?p)(timeMode BETWEEN)(time ?t1 ?t2)) (test (> tx t1))
  =>(duplicate ?normFact (status VIOLATED)
    (timeMode AtTime)(time ?tx))(retract ?normFact))
;;;ENR5
(defrule sellerBidderPunishment
  ?roleFact<-(role (roleTitle seller)(agentName ?x))
  ?normFact<-(norm (status VIOLATED)(deoMode Frb) (act placeBid) (addressee
  ?x))
  => (assert (enforcementNorm (status PUNISHMENT) (addressee ?x)
    (EnfCode P2:toDecreaseFeedbackValueOfSeller)))
    (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P2))))
```

The following snapshot shows the assignment of this sanction to Sarah.



RO-11) A deadline: Current time is “11:00:00 2/1/2008”.

This event notifies that the current time is 11:00 which is a deadline shows the ending time of Auction_1. Therefore, the rules sensitive to this deadline will be fired. Such as Rule GR8 which leads to deactivating the permission of the buyers for placing bid.

```
;GR8
(defrule statusChangeForbiddenBetween
  ?CurTimeFact <-(currentTime (value ?tx))
  ?normFact<-(norm (status ACTIVATED)(deoMode Frb)
    (timeMode BETWEEN)(time ?t ?tx))
  => (duplicate ?normFact (status DEACTIVATED)(timeMode AtTime)
    (time ?tx))(retract ?normFact))
```

The following figure shows how this norm is executed for David and Nina.



RO-12) An action: Nina wins Auction_1 with 30GBP at “11:00 2/1/2008”.

By ending the auction, MAS reports the following event to our tool:

event (act win)(actor Nina)(object GoldWatch Auction_1 30GBP)

As the result of occurrence of this event, the rules R10 and R11 are fired.

```
;R10,R11 : If buyer wins the auction Seller has the right to receive the money from the
buyer and buyer is obliged to pay the price of the item.
(defrule receivePaymentRightAndPaymentObligation
  ?roleFact1 <-( role (roleTitle seller)(agentName ?s))
  ?roleFact2 <-( role (roleTitle buyer)(agentName ?b))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act win)(actor ?b)(object ?item ?auction ?price)(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Right) (act recievePayment)
  (addressee ?s)(benef ?b)(object ?item ?auction ?price)(timeMode BEFORE)
  (time (getPaymentDueTime ?startTime))))
  (assert (norm (status ACTIVATED) (deoMode Obl) (act pay)
  (addressee ?b)(benef ?s) (object ?item ?auction ?price) (timeMode BEFORE)
  (time (getPaymentDueTime ?startTime)))) ) )
```

The following snapshot shows the *assignment of the right of receiving the payment* to Sarah. And also it shows the *assignment of the responsibility of the payment* to Nina. (Here we mention that our natural language translator is very primitive, so in some cases the propositions are grammarless.)



RO-13) An action: Nina pays 30GBP to Sarah (fast payment) at “11:05 2/1/2008”.

As Nina pays the payment, the following events are asserted to our tool:

event (act pay)(actor Nina)(object GoldWatch Auction_1 30GBP)

event (act receivePayment)(actor Nina)(object GoldWatch Auction_1 30GBP)

According to the regulations of this system, we suppose that fast payment is a payment paid within 10 minutes of ending the auction by buyer and that a reward has been defined for such payer. The occurrence of the above event leads to the execution of the following norms. GR1 deactivates the obligation of any undertaken action. GR3 deactivates the right of any undertaken action; here, the right of seller for receiving the payment is deactivated. R12 activates the right of the buyer for receiving the item. R13 activates the obligation of sending the item to the buyer. ENR2 is the norm which shows the reward considered for a buyer who has paid the very fast payment. This reward is to increment the buyer's feedback and is executed by internal agents using the code (R1).

```

;;GR1
(defrule statusChangeForObligationNorm
  ?eventFact<-(event (act ?x)(actor ?y)(AtTime ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Obl)(act ?x)(addressee ?y))
=>( duplicate ?normFact (status DEACTIVATED FULFILLED)
    (timeMode AtTime)(time ?t) (retract ?normFact))
;;GR3
(defrule statusChangeForRightNorm
  ?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p )(AtTime ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Right)(act ?x)
    (addressee ?y)(object ?z ?p ))
=>( assert (norm (status FULFILLED)(deoMode Right)(act ?x)(addressee ?y)
    (object ?z ?p ) (timeMode AtTime)(time ?t) ) (retract ?normFact ) )
;;R12 : If buyer pays the price, buyer has the right to get the item.
(defrule getItemRight
  ?roleFact1 <-( role (roleTitle seller)(agentName ?s))
  ?roleFact2 <-( role (roleTitle buyer)(agentName ?b))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act pay)(actor ?b)(forPerson ?s)(object ?item ?auction
?price)(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Right) (act getItem) (addressee
?b)(benef ?s)(object ?item ?auction ?price) (timeMode BEFORE) (time
(getSendingDueTime ?time))) ) )
;;R13 : If buyer pays the price, seller is obliged to send the item
(defrule sendItemObligation
  ?roleFact1 <-( role (roleTitle seller)(agentName ?s))
  ?roleFact2 <-( role (roleTitle buyer)(agentName ?b))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act pay)(actor ?b)(forPerson ?s)(object ?item ?auction
?price)(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Obl) (act sendItem) (addressee ?s)
(benef ?b)(object ?item ?auction ?price) (timeMode BEFORE) (time
(getSendingDueTime ?startTime))) ) )

```

```

;;ENR2; reward for the payments in 10 minutes of end time
(defrule rewardForFastPayment
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))
  ?normFact<-(norm (status FULFILLED)(deoMode Right)(act pay)
    (addressee ?x)(timeMode AtTime)(time ?time))
  ?sFact<-(auctionStartTime (value ?startTime))
  (test(< ?time (getEndTime ?startTime)+600000))
=> (assert (enforcementNorm (status REWARD) (addressee ?x)
  (EnfCode R1:toIncreaseFeedbackValue)))

```

The following figure shows the fulfilment of Sarah's right for receiving the payment and Nina's responsibility for the payment.

In addition, it shows the *assignment of the right of getting the item* to Nina and *the assignment of a reward as an enforcement norm* to Nina.



RO-14) A deadline: Current time is “10:00 9/1/2008” (next week).

This event notifies that the current time is 10:00 9/1/2008 (7 days after starting the auction) which is a deadline for sending the item by Sarah as a Seller. Therefore, at this time, our application checks the status of sending the item by the following rule (VR1).

```

;VR1
(defrule violatedObligationBefore
  ?CurTimeFact <-(currentTime (value ?t))
  ?normFact<-(norm (status ACTIVATED)(deoMode Obl)(timeMode BEFORE)
    (time ?t))
=> (duplicate ?normFact (status DEACTIVATED VIOLATED)
    (timeMode AtTime)(time ?t))(retract ?normFact))

```

As the norm of obligation for sending the item is still ACTIVATED and not FULFILLED, the rule VR1 will be fired at this time, which shows this norm to be violated.

RO-15) An action: Nina does not receive the item by “10:00 9/1/2008”.

As the result of the previous violation detection ENR3 rule is activated:

```

;ENR3
(defrule rightToClaimForCompensation
  ?normFact<-(norm (status VIOLATED)(deoMode Right)(act getItem)
    (addressee ?x)(timeMode AtTime)(time ?time))
=> (assert (norm (status ACTIVATED) (deoMode Right) (act claim)
    (addressee ?x)(object ?item ?auction)(timeMode AFTER) (time ?time)))

```

The next snapshot shows *the assignment of the right of claiming* to Nina:



RO-16) An action: Nina claims for receiving compensation at “11:10 9/1/2008”.

Suppose that Nina claims for compensation, so the MAS reports this event to our tool as follows:

event (act claim)(actor Nina)(object GoldWatch Auction_1)

According to rule ENR4, the claim of Nina is accepted and she will be given the compensation.

```

;ENR4
(defrule compensationForClaimantPayment
  ?roleFact <-( role (roleTitle buyer)(agentName ?x)(AtTime ?time))
  ?eventFact<-( event (act claim)(actor ?x)(object ?y ?z ))

  ?normFact <-(norm (status ACTIVATED) (deoMode Right) (act claim))
=> (assert (enforcementNorm (status COMPENSATION) (addressee ?x)
  (EnfCode C1:compensationClaimAccepted)))

```

8.4 Evaluation and Comparison

After testing our application and representing its functionality by trying the two methods, now we evaluate and compare these methods.

Before comparing these methods, we state that dynamic assignment of R&Rs and sanctions to external agents can be implemented for a MAS intended to use this approach in two ways: the first way is to build a separate tool over a pre-developed MAS. In this case, a MAS which has already been developed is intended to use one of our proposed methods for dynamic assignment of R&Rs to agents. The second way is to apply these methods to the MAS during the initial design of the MAS. In this case, dynamic assignment of R&Rs and sanctions is considered as a stage of design and then is included in the implementation of the MAS.

In order to know which method (Method 1 or Method 2) is most suitable for an MAS intended to apply this technique, we argue that it depends on the features of that particular MAS. If the MAS has a hierarchy of roles, Method 1 is more appropriate, because selecting Method 1 needs less effort for defining norms in a normative KB, compared with Method 2. Because, in Method 1, norms are rarely conditional-norms and they are mostly single constraints.

However, if the particular MAS contains some general roles and does not have a role hierarchy, Method 2 is more suited for that MAS; this is especially so when the MAS has been already developed and adding this approach is desired after the process of development of MAS.

As we mentioned in Section 4.3.1, some of the MASs cited in [61, 63, 64] have the capability of dynamic assignment of roles to external agents. Here we

suppose that the MAS - in which our methods are applied - has this capability and so the MAS dynamically assigns roles to agents.

In Method 1, the system designer should define several sub-roles for the main roles of the system such that the role of an agent will be changed based on the agent's action or environmental events. These sub-roles and the transitions between them are defined by designer of the system using protocols. In fact, at runtime the assignment of roles to agents is undertaken by the MAS and the result of this assignment is reported to our tool. The advantage of this method is that creating norms for the normative knowledge base is easier in this method and needs less effort for the KB designer.

In comparison, Method 2 is based on conditional norms such that assignment of the main roles (e.g. Buyer) to agents is also a part of the conditional norms in the KB. So it is an advantage of Method 2 because providing any changes or updates of regulations of the normative MAS, does not affect any changes at design level; it only needs to change the KB of R&Rs.

In Method 2, the assignment of R&Rs to agents occurs almost in one phase, since there are not so many changes in roles. In order to enable dynamic assignment of the R&R of the roles to agents, first, the Jess fact base should be updated with the new occurrence, and next, a reasoning process should be undertaken by Jess (based on the Jess fact base and the Jess rule base).

However, in Method 1, the assignment of R&Rs to agents has two phases. After occurrence of each action or event, the MAS should assign roles to agents. Then, the R&R of the sub-roles assigns to agents like R&R assignment in Method 2.

8.5 Summary

This chapter provided the implementation and testing stages of the development process of our middleware tool. It started with some general descriptions of implementation of this tool. In order to test our application, we defined an illustrative scenario which covers all different possibilities (we discussed in this thesis) at runtime. Then, we tested both Method 1 and Method 2 using this scenario. Finally, we evaluated these methods and concluded that both methods have their own advantages. Consequently, the method which is more appropriate for applying in any MAS depends on the infrastructure of the particular MAS.

Chapter 9

Discussion

9.1 Introduction

In this chapter, we discuss three main issues: the related research works, our contributions in this area of research and possible future work.

Although in previous chapters we point out some of related works, in this chapter we describe them in detail in Section 9.2. We do so here, rather than in Chapter 2, because we wish to compare these previous systems and methodologies with our proposals, and to do so first required us to explain our proposals. Section 9.3 provides a summary of our work and we emphasize our

contributions to knowledge in Section 9.4. Finally in Section 9.5, we will mention possible future work which can be investigated in this area of research.

9.2 Comparison with Related Work

In this section, we explain the four research areas related to our research including: electronic institutions, dynamic assignment of roles to agents, normative framework for MASs, and research on beneficiary, rights and compensations.

9.2.1 Electronic Institutions

So far various methodologies have been proposed for developing multiagent systems, such as Gaia [83], SODA [59], Tropos [32], MESSAGE [55], Prometheus [60], MaSE[14], and AAIL [41]. In addition, several researchers have made comparisons and evaluations over these different methodologies [3, 12, 13, 37, 50]. These methodologies can still be applicable in many working areas, depending on the system requirements. However, our investigation in [15] shows that the trend of multiagent systems towards increasingly open and distributed systems requires new aspects of methodologies to be considered. This is essentially because the tasks of design and development for distributed and open MAS are very complicated and difficult.

The distributed nature of multiagent systems, on the one hand, and the need for high-level and flexible interactions among autonomous entities, on the other hand, makes the task of design and development very complicated and difficult [38], specially when open MAS are populated by heterogeneous and self-interested agents.

One of the solutions for dealing with this problem of self-interested agents is to introduce regulatory structures for open multiagent systems. In this way, an open system can be seen as an agent society in which normative rules provided over the system define the relations and interactions of agents and specify what actions agents are permitted and forbidden to do.

For this purpose, several agent researchers have focused recently on the introduction of social concepts such as organizations and institutions [17, 79], [81]. As an example, in [17] the authors introduce their own framework for multiagent systems, called OMNI. Electronic institutions (EIs) [2, 21, 56] is another approach which tries to provide a regulatory structure over a multiagent system. EIs define the rules of agent societies similar to those of human institutions, by defining what agents are permitted and forbidden to do. Since the research and applications of EIs are most related to our research, we describe it next.

Electronic Institutions (EI) is one of the topics related to our research. The Project of Electronic Institutions has been mainly studied in Artificial Intelligence Research Institute (IIIA) of the Spanish research council (CSIC) [35]. In this section, we first introduce the notion of EI and consider its applicability; then we present a few criticisms of EI and finally we discuss the relation of this work with our work and we compare the two approaches.

9.2.1.1 Definition of EIs

For the definition of the EI, we refer to the definition in [22] as follows:

“...Agent organizations can be effectively designed and implemented as institutionalized agent organizations ---that henceforth we shall term electronic institutions, or e-institutions for shorter. From this [sic it]

follows that one of our main purposes will be to model the creation, evolution, and consequences of the rules defining institutions and their incorporation into agent organizations.”(Page 127)

Electronic Institutions provide a regulated virtual environment where a variety of participating agents have interactions. This simple view shapes the theoretical components of the institutions including agents and roles, the dialogical framework, scenes, performative structures and normative rules [20, 21]; these notions are described as follows:

The Dialogical framework defines all roles that participating agents can play in the institution, along with a common ontology as a common language which enables agents to exchange knowledge with other agents. A *scene* is a group of agents playing different roles in interaction with each other to realize a given activity. Every scene follows a well-defined Communication Protocol. The *performative structure* is the graphical specification of EIs which performs a network of scenes, and specifies how the agents can move between the different scenes. *Normative Rules* of an EI defines the commitments, obligations and rights of participating agents. Norms capture the consequences of an agent’s actions within institutions.

9.2.1.2 Tools for EIs

The working group of IIIA presented an integrated environment called Electronic Institutions Integrated Development Environment (EIDE) to support the engineering of MAS as electronic institutions [2, 70]. Figure 11 shows the development cycle of EIDE.

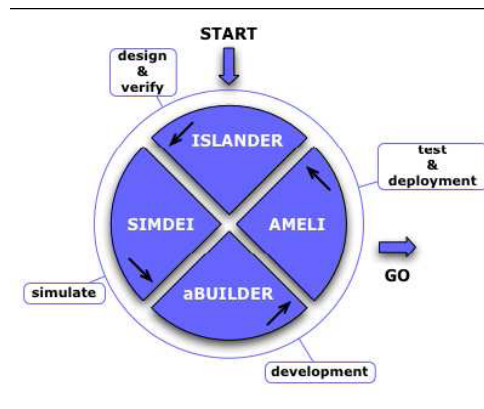


Figure 11-EI Development Cycles reproduced from [2]

As Figure 11 shows, EIDE consists of set of tools as follows [2]:

ISLANDER [19, 67]: This is a graphical editor which provides the design level and static verification in EI development cycle. In this tool, the following components are defined: the dialogical framework, scenes and performative structure, ontologies, illocutions and norms [2, 19]. The EI common ontology contains the vocabulary, datatype and function definitions [18] that agents use for exchanging information along with the collection of norms that regulate their actions (protocol-based norms).

SIMDEI: This is a simulation tool to animate and analyse ISLANDER specification.

aBuilder: This is an agent development tool which generates an agent skeleton for the EI. The generated skeleton can be used on EI simulation by SIMDEI, or in a real execution of the institution AMELI.

AMELI [23]: This is a software platform to run EIs which makes agents participation possible. For running the system, it uses manager agents,

including Institution Manager (IM), Transition Manager (TM), Scene Manager (SM), and Governor (G).

Monitoring tool: This is a tool which allows the monitoring of EI execution run by AMELI. It graphically shows all the events occurring during an EI execution.

The part of this integrated environment most related to our work is the EI's management of external agents. In EIDE, external agents do not participate directly in an electronic institution. Instead, all their interactions are mediated by AMELI through a special type of internal agent called a *governor*. The governors are part of the social layer of EI and manage the communication of an agent with the other agents in the e-institution. Agents communicate to their governors. Governors check the messages sent by agents within the scenes. If messages are correct and according to the institution specifications, governors transmit them to the addressed agents in the scene, otherwise agent messages are not transmitted.

Governors not only filter the messages and exclude the illegal ones but also manage the set of norm expressions that trigger obligations to the agent they represent. This means that governors keep, at any given moment, the pending obligations of their associated agents and they check whether agent interactions activate or de-activate the obligations [2, 80].

9.2.1.3 Applications Using EI

Electronic Institutions and EIDE was a successful and applicable model and these frameworks have been applied to real application domains.

For example, the **Fish Market Project** [73] is a preliminary system founded on EIs which provides an electronic version of a real-life fish market auction. In this electronic marketplace, intermediaries (such as an auctioneer, a market boss and a receptionist) are internal agents who interact with two types of external agents - buyers and sellers. Since these interactions can be associated with standardized illocutions, the target is to perform an electronic version of market interactions using electronic intermediaries, instead of requiring the physical presence of these market participants.

Our second example is **MASFIT** (Multi-Agent System for Fish Trading) [11, 74] a conjunction of the Fish Market project which allows buyer agents to participate in several simultaneous auctions. In MASFIT, customers can participate *remotely* in several fish markets *simultaneously* with the help of software agents, while keeping the traditional auction procedures. In this system, the purchase is not carried out by the user, but rather by one of the intelligent software agents automatically, based on a predefined approach and with a capability of reacting to events as the auction progresses.

Our third example is **HarmonIA** [65, 66] which is a software tool developed in a project by Daniel J. Pastor and Julian Padget at the University of Bath. The main objective of HarmonIA is the automatic generation of any kind of institution. Using the description of institution and ontology file(s) as inputs, this system produces the Java skeleton files of the organization for the JADE [5] platform as output files. These files provide the main notions of the EIs including agents, dialogic frameworks, scenes and performative structures. However, HarmonIA does not yet include norm specifications covering from norms to protocols; this feature is left to future work of the development group.

9.2.1.4 Evaluation of EI

After the development and publication of EIDE, several researchers took notice of and have studied this work, undertaking evaluations, critiques and extensions of EIs. In the following, we summarize some of these discussions and the issues raised, including the benefits and weaknesses of the current implementation of electronic institutions by EIDE tools; we also mention proposed extensions of the current version.

Benefits: Since one of the most important factors in the success of a methodology is the availability of usable development tools [3], the development of EIDE tools and its usability increases the usefulness of the EI model. Furthermore, ISLANDER as the editor tool of EIDE provides a sound model for the domain ontology and has a formal semantics [22]; the specifications done in ISLANDER are independent of any programming language and the outputs generated by the tool can be translated into different languages [18]. In addition, the engineering of an EI is a low-cost implementation since only its participating agents must be programmed. In such an integrated system, maintenance is a very easy task while changes are applied in a new specification, they are ready to be run by AMELI, and agents are easily created [2].

Weaknesses: Besides the benefits, the current implementation of EIs has several weaknesses: these can be summarized in one major problem which is the lack of support for rule-based norms including both main norms and enforcement norms.

So far these tools have implemented just protocol-based norms which are defined using dialogical frameworks, scenes and performative structures in EIs

[30]. Protocol-based norms regulate dialogical actions of the system such as interchanging messages between internal agents [71]. Although the current implementation of EIs provides an interface for defining obligation norms (which is a kind of rule-based norm) for external agents, in practice the actions of external agents are filtered by governors.

As mentioned in Section 3.2, a complete normative system contains regulative norms and distributive norms which we called main norms and enforcement norms in this context. However, the current version of EIs does not support these notions. Vázquez-Salceda and his colleagues in [80] mentioned these weaknesses as follows: (a) ISLANDER does not propose expressiveness to indicate norms involving prohibitions, permissions, or sanctions; (b) using temporal operators (including before, after and between) is not allowed in ISLANDER; and (c) the specification of non-dialogical actions is not allowed in ISLANDER.

For more clarification, we emphasize that so far EIDE tools provide norm regimentation not norm enforcement. Recall from Section 3.4.6, norm regimentation makes the violation of the norms impossible and external agents have no autonomy to violate a norm, while in norm enforcement external agents are autonomous. As mentioned in Section 9.2.1.2, AMELI (which is the environment of running EIs), uses governors to check the validity of the external agent's actions. It means that governors allow only the permitted actions to be performed [26].

Extensions: Followed by the above critique and analysis, several efforts have attempted to extend EIs. These effort started by defining a normative language for electronic institutions in [30, 78, 80]. The latest proposed normative language for electronic institutions is that of Garcia-Camino and his colleagues

[31]. With respect to extending the implementation of norm enforcement in EIs, H. Aldewereld and his colleagues have studied and proposed a formal solution for the mechanism of violation detections and reactions in [1].

Nevertheless, the mentioned extended formalism and mechanism of EIs has not been practically proposed yet.

9.2.1.5 Comparison with Our Work

From a practical point of view, EI has not developed rule-based norms at the application level, although the formalism of these norms has been proposed. In comparison, we have also proposed our methods in application level. In addition, our proposed methods and our application contain many kinds of legal modalities of norms including obligations, prohibitions, permissions and rights, while so far EI applications have not covered all legal modalities. Moreover, our implemented norms are enforced by the application and in the case of violation, reward and compensation will be decided based on the rules predefined by legislator, while enforcement mechanisms have not been implemented in EIs yet. This makes our approach close to the reality of human social interactions.

From the norm formalism viewpoint, the grammar of the normative descriptive language of EI contains some specific elements to present EI's component such as scenes and dialogical frameworks. As our methods can be implemented over any type of MAS, we have not considered these elements in our formalism. One advantage of our formalism is that we have defined the notions of beneficiary, right and compensation in our normative language. Thus our approach is more expressive than the EI approach.

With respect to role assignment, the roles of agents do not change at runtime in EIs. Once an agent is assigned a role, the agent will keep this role for the duration of runtime. Therefore, the assignment of roles to agents is static in EIDE and the system does not permit any role changes in this environment. In our approach, by contrast, we have considered that a mechanism for the dynamic assignment of roles to agents is provided by the MAS intended to use our methods.

In summary, in comparison to EIs, our approach allows for greater dynamism and thus more realistic in its treatment of norms.

9.2.2 Systems and Dynamic Assignment of Roles to Agents

We have already categorized dynamic assignments in three types: including dynamic assignment of roles to agents; dynamic assignment of rights and responsibilities to agents; and dynamic assignment of sanctions to agents. While we worked on dynamic assignment of rights, responsibilities and sanctions to external agents, there exists a similar topic in this research area for dynamic assignment of roles to agents. In this section, we consider research relevant to the context of dynamic assignment of roles to agents.

We first mention the survey research by Partsakoulakis and Vouros in [63]. They worked on roles in multiagent systems and studied different features of roles in methodologies, formal models and implemented MAS. The most related part of this study is dynamic assignment of roles to agents which we focus on here. As mentioned in [63], dynamic assignment of roles to agents is based on the *conditions of roles*, on an *agent's capabilities*, and on the *overall context of actions*. Furthermore, Partsakoulakis and Vouros in [63] concluded that AOSE methodologies, including MaSE, Gaia and AAIL, have no dynamic

assignment of roles to agents; however, the formal model of AALAADIN [24] has this feature.

AALAADIN uses the core concepts of group, agent and role. A group contains a set of roles that are handled by specific agents. There exist a few groups and roles on the platform to handling the kernel system, however, it is remarkable that an agent can create *new groups* and it also can create roles with a transient lifetime in group structure. So the structure of the system organization can be created dynamically. When a new group is created, the creator agent of the group plays the role of group manager who is responsible for handling agent's requests for group admission or role requests at runtime. Therefore, this formal model has dynamic assignment of roles to agents.

In addition, Partsakoulakis and Vouros in [63] have studied the feature of dynamic assignment of roles to agents in implemented MASs and achieved the following results: Teamcore [72] assigns roles to agents dynamically based on an *agent's capabilities*. In Robocup systems [42] this assignment is merely based on *changing positions in the field*. In Role Oriented Programming (ROP) [4], the assignment of roles to agents is on the basis of the *overall context of actions*. Their achievements have been summarized in a table which we present in Appendix A.

Additional related work is that of Kim on dynamic role assignment [40]. He proposed a dynamic role assignment mechanism for a team of cooperative virtual agents working in interactive computer games [40]. In this work, role assignment to agents has two phases: in a static phase, all situation-dependant assignments of roles are predefined at design time, and in a dynamic phase, detailed decisions regarding which member agent has to take what specific role are made at execution time.

Our third example on role assignment to agents has been provided by Odell and his colleagues in [57, 58]. Their work analyzes and classifies the various kinds of dynamic role changes that may occur in a multiagent system. For example, the operations for state transition have been defined as follows: classify, declassify, reclassify, activate, suspend and shift. Such analysis would be useful in developing the formal description of the applications. So their work has focused on theoretical aspect of work rather than the practical aspects.

Our work, in contrast, has considered both dynamic assignment of roles and dynamic assignment of R&Rs and sanctions, to agents and done so in a manner while we have shown can be readily implemented.

9.2.2.1 Comparison with Our Work

As mentioned, dynamic assignment of roles to agents (in the applied systems described above) is based on: the conditions of roles; each agent's capabilities; the overall context of actions; and on factors such as the position of agent in space (e.g. in Robocup). In comparison, our work on dynamic assignment of R&Rs and sanctions to agents is based on a normative knowledge base and runtime occurrences including actions and environmental events.

It is helpful to mention that once we concentrate on the development of dynamic assignment of R&Rs to agents and we desire that our methods can be applied to any kind of MAS, we assume that the MAS intended to use our methods provides this technique. However, it is possible in our method to define the assignment of roles to agents such that the regulation of every role assignment is constituted as a rule, and the same procedure for assignment of R&Rs to agents is used for the assignment of roles to agents.

9.2.3 Normative Framework for MASs

L'opez y L'opez, Luck and d'Inverno have also worked on a normative framework for agent-based systems in a series of papers [45-48]. The primary step of their research uses BDI-like agent architectures (based on beliefs, desires and intentions) in which agents make plans for their actions to reach their goals [47]. Then, in order to achieve a desirable social order in the society of agents and to avoid possible conflicts, the constitution and enforcement of norms has been represented [46]. These norms influence the behaviour of agents.

Before presentation of the framework, a few definitions are needed, one of which is the definition of normative agent, as follows [48]:

“A normative agent is an autonomous agent whose behaviour is partly determined by obligations it must comply with, prohibitions that limit the kind of goals that it can pursue, social commitments that have been created during its social life and social codes which may not carry punishments, but whose fulfilment could represent social satisfaction for the agent. Moreover, autonomous agents can decide whether to adopt or ignore norms.” (Page 731)

For such normative autonomous agents, the norms and the normative multiagent system in which the agents participate, a formal framework has been presented [49]. To describe the pattern of agent behaviours, this model use normative goals instead of predefined actions. So, agents can choose the way to achieve their normative goals. This idea is more compatible with the idea that agents are autonomous. This framework attempts to provide a solution intended to be used by agents that reason about why norms must be adopted and why an adopted norm must be complied with.

9.2.3.1 Comparison with Our Work

In comparison, our work mostly stresses the practical and implementation aspects of normative multiagent systems, while Lopez and her colleagues work proposed a formal framework which has not been practically developed yet. In addition, in our thesis, norms are applied to external agents who join to the normative MAS from outside. These agents have autonomy and may not be aware precisely of the details of the normative regulations of the system. In contrast, norms in the framework of Lopez *et al.* are applied to internal agents who know the norms and consider these norms in their normative goals.

In addition, our work is not limited to BDI-like agents. We make no assumptions regarding the decision-making architecture of the agents in the MAS. Moreover, we considered the concepts of beneficiary, rights and compensation in our work, while these notations have not been defined in the framework of Lopez *et al.*

9.2.4 Beneficiaries, Rights and Compensation

Beneficiaries, rights and claimants have relationships with each other. This topic has been discussed in legal domains by researchers such as Lyons [52] some time ago. However, this topic has not been discussed to any real extent in normative multiagent systems as yet. So we have extended the descriptive language for the first time in Section 3.5 to consider these notions. And then we have implemented and illustrated the notions of beneficiary, right and compensation in this thesis.

9.3 Summary

To sum up this thesis, we would like to review the objectives we sought to achieve at the beginning, in accordance with the questions of the research mentioned in Section 1.1. Then we consider whether and how we have achieved these goals in our research. To do so, we summarize the context of chapters, and mention the relevant research question which was answered in each chapter, as follows:

In Chapter 1, we provided an introduction to the title, context and structure of this thesis. Then, in Chapter 2, we provided the necessary conceptual background to this research, with concepts such as autonomous agent, multiagent systems, rights and responsibilities, external agent and normative multiagent systems.

Chapter 3 is an overview on the normative concepts relevant for normative multiagent systems. This chapter explained all important issues relevant to rights and responsibilities of agents, issues such as norm classifications, the key elements of norms, and norm enforcement. We have taken these concepts from the literature, and then adopted and extended them. Then, after an explanation of descriptive normative languages, we specified the language, building on the work of Silva [71], and the complementary extensions of that language, which we added to support the notions of right and compensation.

In Chapter 4, we addressed the main theme and direction of this research, focusing on the more detailed aspects related to providing dynamic assignment of R&Rs to agents in normative MAS. First, dynamic issues in MAS along with the source of dynamism and changes have been explained. Because such dynamism results in more complicated management of the MAS, dynamic

assignments were presented as a means to improve the performance of the system.

We categorized dynamic assignments in normative MAS as one of three different types: dynamic assignment of roles to agents; dynamic assignment of rights and responsibilities to agents; and dynamic assignment of sanctions to agents. However, we just focused on dynamic assignment of rights and responsibilities to agents and dynamic assignment of sanctions, since dynamic assignment of roles to agents has previously been proposed in earlier research by others as cited in [64]. Therefore, this chapter provides the answer to the following questions:

- *What is the concept of dynamic assignment of Rights and Responsibilities (R&Rs) and of sanctions to agents?*
- *Which factors may cause the corresponding R&Rs and Sanctions of an agent to change at runtime?*

In Chapter 5, we explained a formal definition and specification of two alternative mechanisms by which dynamic assignment of R&Rs and sanctions to external agents can occur. We have proposed two methods for such assignments. Method 1 is based on role hierarchies, while Method 2 is based on conditional norms.

The common features of these two methods are as follows: both of them rely on the concept of role; both use a normative knowledge base; and runtime occurrences are considered in both methods. Since the two methods are not identical, their differences are as follows: the definition of roles is different

because Method 1 uses a role hierarchy; and the definition of a normative KB is different, because Method 2 is based on conditional norms. Because of the importance of formal representation, we provided the formal representation of the function of dynamic assignment of R&Rs and sanctions to external agents based on the common features of both methods. Thus, this chapter provides the answer of the following question:

- *What methods can be proposed to undertake such assignments?*

In Chapter 6, we discussed the main implementation issues of these methods. Analysing the similarities and differences of both methods from an implementation viewpoint, we concluded that the most important differences are in the definition of roles and the normative KB; this means that norms are defined differently in Method 1 and Method 2. But runtime occurrences are the same in both methods. Consequently, we explained different ways of defining roles and the main issues for designing the normative KB in both methods. We also proposed our general architecture on the basis of the common features of the two methods in Chapter 6; this was presented in the form of an architecture diagram, and was followed by a detailed worked example of its application. Thus, this chapter provides the answer of the following question:

- *How can the proposed methods be implemented?*

As we implemented our proposed methods, in Chapter 7, we presented the analysis and design of a middleware tool. This tool is generic, can be connected to any MAS intended to apply our methods, and is independent from the

development details of that MAS. The tool is also independent of the method chosen to implement dynamic assignment of R&Rs, whether our proposed Method 1 or our proposed Method 2.

One of the important tasks in designing such a tool is the creation of the normative knowledge base. Either of our two proposed methods can be used for creating this normative KB, depending on the features of the MAS. If the MAS has been created on the basis of a role hierarchy, Method 1 would be more appropriate for creating the normative KB than Method 2; otherwise, Method 2 would be more appropriate.

Chapter 8 provided the implementation and testing stages of the development process of our middleware tool. After some general descriptions of implementation of this tool, we tested it for both methods using a scenario which covers all different runtime possibilities (we discussed in this dissertation). Finally, we evaluated these methods and concluded that both methods have their own advantages and the most appropriate method for applying in a MAS depends on the infrastructure of MAS.

In fact, the practical implementation of these methods which is described in Chapter 7 and Chapter 8, is an answer for the following question:

- *What is a practical example of such implementation?*

9.4 Our Contributions

At the conclusion of this thesis, in this section, we briefly mention our main contribution in this area of knowledge. Generally, this thesis has addressed one

particular problem for open and normative multiagent systems which have dynamic environments. We explicitly identify, clarify and address the problem of dynamic assignment of rights, responsibilities (R&Rs) and sanctions to external agents in normative multiagent systems. This is the first explicit treatment of dynamic assignment of R&Rs and sanctions.

In order to provide an approach for dynamic assignment of R&Rs and sanctions to external agents, the background setting of this work deals with the topic of dynamism in normative MASs; it attempts to address and combine some issues regarding dynamic resources in MASs and different types of norms in legal systems. These types of norms consist of various types of legal modalities, including obligation, prohibition, permission and right, enforcement modalities including punishment, reward and compensation and key elements of norms, such as addressee, beneficiary, temporal notions, and preconditions. Specifically, we incorporated the notions of beneficiary, right and compensation which have not previously been considered from an application viewpoint.

We proposed two alternative methods for dynamic assignment of R&Rs and sanctions to external agents, along with a proposal for a formalism to represent common sense understanding of our solution. The first method is based on role hierarchies in multiagent systems and the second method is on the basis of conditional norms in normative MASs. Both methods have common features, including a reliance on the role concept, using a normative KB, and sensitivity to runtime occurrences of the MAS. The significant differences of these mechanisms are different ways of defining roles and normative KBs.

Furthermore, we considered aspects of implementation based on common features of our proposed methods, followed by presentation of a general

architecture for implementation of dynamic assignment of R&Rs and sanctions to external agents. Because of the importance of the normative knowledge base in this technique, we analyzed the structure of such a knowledge base. Then we proposed that such normative knowledge base composed of two parts: general rules and domain-related rules. We proposed general rules which are common and can be applied for the implementation of norm enforcement in any MAS intended to be used in our approach. We proposed guidelines for system designers for creating the domain-related parts as well.

Finally, using these implementation issues and general guidelines, we develop a generic application to demonstrate the practical feasibility of our approach and of our architecture. This application enables the provision of our dynamic assignment methods in normative MASs.

With this perspective, using an auction example we examined the functionality of this application for both methods. The normative knowledge base of this auction example (for each method) contains various types of mentioned legal notions. Then, in order to show practical assignment of R&Rs and sanctions at runtime, we produced a scenario containing several cases of runtime occurrences, such that applying this scenario represents dynamic assignment of R&Rs and sanctions at runtime to external agents.

9.4.1 Our Assumptions and Limitations

Here we summarize the assumptions of our work, and then outline the limitation of the work. These assumptions and limitations may provide a basis for future work.

One of our main assumptions is related to the separation of internal and external agents. We assumed that the MAS - intended to use our approach for

dynamic assignment of R&Rs and sanctions - contains two types of agents: firstly, internal agents who work on behalf of the MAS based on the predefined protocols at design time such that these agents have no autonomy to violate norms; secondly, external agents who join the MAS and have the autonomy to either follow or violate the norms of the system. Because we wanted to represent the capability of our approach over autonomous agents, the assumption of having external agents with autonomy to follow or violate the norms was beneficial in our work.

The second assumption is related to the definition of a normative multiagent system. As mentioned before - based on Boella's definition of normative multiagent system - in normative MASs, agents affect norms and norms influence the agents' behaviors, but in our approach we assumed that agents do not influence norms. We therefore did not consider the case of agents being able to change norms.

The third assumption is that we supposed the normative knowledge base contains the main legal modalities, not detailed normative positions. In Section 3.3.1, when we explained legal modalities, we also mentioned that there are additional detailed legal modalities, for example, obligative rights and permissive rights. However, in this work we basically attempted to provide a mechanism for dynamic assignment of R&Rs and sanctions to agents. Considering normative positions has no additional benefits at this primary stage.

The final main assumption is that we supposed that the MASs using our approach have already had the facility of dynamic assignment of roles to agents, because we wanted to focus on dynamic assignment of R&Rs and sanctions to agents. However, if the task of assignment of roles to agents is a

regulated task and these regulations are able to be formulated as norms in the normative knowledge base, our approach is also able to provide the dynamic assignment of roles to agents.

With respect to the limitations of our work, we note five key limitations as follows: First, in some cases, a legislator may define conflicting norms in a normative knowledge base. For example, in one norm an action is permitted while another norm imposes an obligation for doing that action. For more clarification, the following example is shown. Suppose that norms have been defined in the normative knowledge base:

Norm1: *“Buyer is permitted to place a bid during the auction”*.

Norm2: *“If a buyer has 3 negative feedbacks, he is forbidden to place a bid during the auction”*.

Suppose that David is a buyer and has got the permission to place a bid at the beginning of the auction for the whole time of the auction. But during the auction time he has got 3 negative feedbacks which leads to assignment of Norm 2 to him. As a result, two opposite norms are assigned to David, one of them is a permission for doing an action while the other is a prohibition for doing the same action. In the current version of this application we did not consider norm confliction. Thus, we have no mechanism for identifying, resolving or mitigating such norm conflicts.

Second, in the implementation of our approach we mentioned that the application uses an inference engine for reasoning. Currently this task is undertaken for each runtime occurrence to detect possible R&R assignment.

However, some of these occurrences do not lead to such assignments; also, reasoning is a time-consuming task. This means that currently the times of reasoning in our implemented application are not yet optimized, and, so redundant reasoning tasks may happen in some cases.

Third, we have not considered issues of scalability, either in terms of numbers of agents, or numbers of norms. The proposed methods may not scale very well.

Fourth, network problems (including disconnections, low speed and transmission delay in sending message) can be significant environmental events; however, we have not considered these issues in our methods.

Fifth, we do not consider issues of the connections, if any, between decisions inside the normative MAS and actions in the external world. An agent in the system may violate a norm and thus receive a sanction, imposed by an enforcer. If this sanction (e.g., a mandatory fine) is not fulfilled, further external sanctions may be applied (e.g., exclusion from the MAS from the external world). The link between events inside the MAS and events outside it (including permission to join and forced exclusion) has not been explored in this thesis.

9.5 Future Work

Here, we mention some of the potential future research avenues which it would be interesting to investigate. Many of these topics address limitations in our model too.

- **Extending the Normative Language to Support Normative Positions:** In Section 3.3.1, when we explained legal modalities, we also mentioned that there are additional detailed legal modalities, for example, obligative rights and permissive rights. Therefore, a possible area of future research would be to formalize the detailed legal modalities and develop methods for the dynamic assignment of these modalities to external agents.

In order to extend the legal modalities of our proposed methods, we would need to further investigate normative positions in the descriptive normative language. This means that the descriptive normative language should be extended to support these normative positions. Subsequently, the extended descriptive normative language could be used in normative knowledge base.

Therefore, adding the above normative notions would only affect the normative knowledge base of the system, not the entire mechanism proposed for dynamic assignment of rights and responsibilities to external agents.

- **Detection of Norm Conflicts:** In the previous section we mentioned that norm conflict is a limitation in our work. In order to avoid such conflicts, it would be possible to equip our application with a component to check the correctness of norms against norm conflicts, for example, using the techniques of minimizing legal modalities (mentioned in Section 3.3.2.3). In this case, remaining with our existing general architecture, an additional component could also be included in our model in order to identify norm conflicts.

- **Semantics of Methods:** In Section 5.5, we provided the formalism of our two methods. In this proposal, we have presented the syntax of the formalism, and so developing a formal semantics for that formalism is a possible future task. Formal semantics would be useful and desirable, since they can ensure that different development teams all share a common understanding of the syntax; subsequently, they can also assist with implementation of the MAS, by facilitating software development and programming.
- **Optimizing the Number of Times Reasoning Occurs:** In the previous Section we mentioned that the times and amounts of reasoning in our implemented tool have not been optimized, and so our system may have extra redundant times of reasoning task. Therefore, working on a method to optimize the number of times an application needs such reasoning would be a worthwhile improvement of the system. For example, while currently each occurrence of an event or an action leads to a reasoning task, a possible solution would be as follows: Comparing the list of defined acts in NKB, one can filter out the irrelevant events and actions which never lead to satisfaction of a norm in the knowledge base and, consequently, never any dynamic assignment.
- **Applying this Approach over Internal Agents:** We defined this approach and also developed our application over external agents. This is because we assumed that currently internal agents of the MASs do not have enough autonomy to behave contrary to whatever their predefined protocols specify. Therefore, as we attempted to provide the mentioned dynamic assignment over autonomous agents, we have applied our methods to external agents which have sufficient autonomy to follow or violate the norms.

In fact, currently most existing MASs place limitations on the autonomy of agents, although they are trying to consider a degree of autonomy for their agents. As an example, in Electronic Institutions there exists an internal role as auction manager. For instance, if the institution defines the responsibility of the management of the auction for the auction manager, then the system is designed in such a way that the auction manager should follow this responsibility and would be unable to violate it. Therefore, it is clear that in such MASs internal agents have insufficient autonomy to violate the system's norms.

Thus, our approach could be extended to cover autonomous internal agents in the future, for systems where internal agents have autonomy to follow or violate the norms.

Creating Legislator Agents: As mentioned before - based on Boella's definition of normative multiagent - in normative MASs, agents affect norms and norms influence agent's behaviors, but we did not consider the influence of agents on norms in our work. Thus, developing methods which would allow authorized internal agents in MASs to create or define new norms at runtime for themselves would be interesting work for the future as well.

Generally speaking, the particular regulations of normative systems may vary over time. For instance, before April 2008, eBay auctions had a norm which said that, "*Sellers can put negative feedback for the Buyer of their item*". But, in April 2008, this norm was changed to, "*Sellers cannot put negative feedback for the Buyer of their item*". So this example shows how norms can vary during the life time of a MAS.

In Our thesis, we supposed the normative MAS using our approach has fixed norms. As an example, during an auction session, the auction regulations do not change. Although in our current design and implementation we supposed that norms are fixed, expanding our general architecture to support norm modification would be very straightforward.

In order to add the feature of norm modification to our system, one could design a legislator agent who has authority to access the normative knowledge base. Accessing the contents of the normative knowledge base, the legislator agent can modify the norms of the system. However, we mention that adding this feature will not change the overall architecture we proposed, but the architecture would just need to include a new component.

Note that although our existing application can be used for any normative MAS, in our thesis most of examples and application tests were based on auction example. We used the auction domain because we could easily ignore the influence of agents on norms in a small auction system. Here we provide a non-auction based example which can apply our methods and application. Suppose that a University has a student record system in which all the stakeholders are represented by software agents. Students and lecturers, through their agents, are provided with various capabilities in the system to record events and interactions, for example, student-supervisor meetings, and to create reports. Postgraduate students may be required to complete progress reports and submit these to their supervisors, for annotation, and potentially for subsequent action by their supervisors, by the Head of Department, or by others. One could easily imagine that the rights and

responsibilities of different stakeholders in this system need to be dynamic, since the status of participants may change over time. Using our methods and application, the R&Rs and sanctions can be dynamically assigned to students as their status changes over time. Of course, in such a major system, the rules may change over time also, and so authorized agents may need to apply many modifications of rules, something our application currently does not support. For example, the University of Liverpool system had a norm which said that, “*PhD students are permitted to use the PDR (Personal Development Record) system*”. But in September 2004, this norm was changed to, “*PhD students are obliged to report their works in the PDR (Personal Development Record) system*”.

- **Integration of our approach with AOSE methodologies:** Similarly for using dynamic assignment of roles to agents in methodologies and formal models, the integration of our approach for dynamic assignment of R&Rs and sanctions to agents with AOSE methodologies could also be a valuable effort. Currently, no standard AOSE methodologies has the feature of dynamic assignment of rights, responsibilities and sanctions to external agents.

It would be a very valuable effort to integrate our approach with AOSE methodologies, because our proposed mechanism helps the management of the MAS, speeds up the dialogues, enforces norms, and reduces the need for a system designer to identify and exclude all behaviors at design time.

- **Linking These Methods to Reputation Systems:** Our methods could also be integrated with reputation systems, such as those used on eBay

(where participants rate each other after each transaction). Currently, eBay has a static system for applying its regulations. Regulations of this auction system are based on predefined protocols or are controlled by its human resources staff to detect violations. These regulations are not assigned dynamically to the buyers or sellers (as external agents), even when a violation occurs.

As an example, eBay has the following norms, where the term of Shill Bidding is defined as *“Shill bidding occurs whenever a seller places a bid on their own item, either directly or through others”*.

Norm 1: *“An eBay seller is forbidden to do shill bidding”*.

Norm 2: *“If an eBay seller uses shill bidding, his account will be suspended”*.

Currently, if a seller uses shill bidding, this violation is not detected at runtime and the auction may be continued normally. However, it is possible that eBay staff detects this violation after a while and suspend the account of the seller. So this example shows that assignment of norms in this case - detection of violation - is not a dynamic task in eBay.

As a result, our application would be useful for such repetitive systems to assign rights and responsibilities to agents dynamically. For example, if a participant obtains more negative ratings than some threshold level, a change in the rights or responsibilities or the imposition of some penalty could be applied, at runtime. In addition, our approach would be helpful when sanctions are dynamically assigned to external agents, for instance, in shill bidding cases.

Because we have developed our application independently as a middleware tool, this tool could be connected to any multiagent system intended to use our methods.

Bibliography

1. H. Aldewereld, A. García-Camino, F. Dignum, P. Noriega, J. A. Rodríguez-Aguilar and C. Sierra (2007), "Operationalisation of norms for electronic institutions", *Coordination, Organization, Institutions and Norms in Agent Systems II*, vol. 4386, Springer-Verlag, 163-176.
2. J. L. Arcos, M. Esteva, P. Noriega, J. A. Rodríguez-Aguilar and C. Sierra (2005), Engineering open environments with Electronic Institutions, *Engineering Applications of Artificial Intelligence*, 18, 191-204.
3. B. Bauer and J. Muller (2004), "Methodologies and Modeling Languages", *Agent-Based Software Development*, M. Luck, R. Ashri and M. d'Inverno (Editors), Artech House Publishers, 77-131.
4. M. Becht, T. Gurzki, J. Klarmann and M. Muscholl (1999), ROPE: role oriented programming environment for multiagent systems, *Proceeding of International Conference on Cooperative Information Systems (IFCIS)*, 325 - 333.

5. F. Bellifemine, A. Poggi and G. Rimassa (1999), JADE - A FIPA-compliant Agent Framework, *Practical Application of Intelligent Agents and Multi-agents*, Practical Application Company, London, 97-108.
6. T. J. M. Bench-Capon and P. E. Dunne (2007), Argumentation in artificial intelligence, *Artificial Intelligence*, 171 (10-15), 619-641.
7. G. Boella, R. Damiano, J. Hulstijn and L. v. d. Torre (2006), Role-based semantics for agent communication: embedding of the 'mental attitudes' and 'social commitments' semantics, *Proceeding of the 5th International Conference on Autonomous Agents and Multiagent Systems*, ACM Press New York, NY, USA, Hakodate, Japan, 688 - 690.
8. G. Boella, J. Hulstijn and L. W. N. v. d. Torre (2005), Virtual Organizations as Normative Multiagent Systems, *Proceeding of the 38th Hawaii International Conference on System Sciences (HICSS-38 2005)*, IEEE Computer Society, Big Island, HI, USA.
9. G. Boella, L. v. d. Torre and H. Verhagen (2005), Introduction to normative multiagent systems, *Proceeding of NorMas Symposium (AISB'05)*, Hatfield, England, 1-7.
10. G. Boella, L. W. N. v. d. Torre and H. Verhagen (2006), Introduction to normative multiagent systems, *Computational & Mathematical Organization Theory*, 12 (2-3), 71-79.
11. G. Cuni, M. Esteva, P. Garcia, E. Puertas, C. Sierra and T. Solchaga (2004), MASFIT: Multi-agent system for fish trading, *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004*, IOS Press, Valencia, Spain, 710-714.
12. K. H. Dam (2003), 'Evaluating and Comparing Agent-Oriented Software Engineering Methodologies', Msc Thesis, RMIT University, Melbourne, Australia.
13. K. H. Dam and M. Winikoff (2003), Comparing agent-oriented methodologies, *Proceedings of International Conference Workshop on Agent-Oriented Information Systems (AAMAS03)*, Melbourne, 52-59.
14. S. A. DeLoach, M. F. Wood and C. H. Sparkman (2001), Multiagent Systems Engineering, *International Journal of Software Engineering and Knowledge Engineering*, 11 (3), 231-258.
15. F. Derakhshan (2007), An overview on Electronic Institutions, *Proceeding of The International Conference on Digital Communications and Computer Applications (DCCA 2007)*, Irbid, Jordan, 557-568.
16. F. Derakhshan, P. McBurney and T. B. Capon (March 2008), Towards dynamic assignment of rights and responsibilities to external agents, *Proceeding of the 13th International CSI Computer Science*,

Communications in Computer and Information Science, Springer-Verlag (To be published).

17. V. Dignum, J. Vázquez-Salceda and F. Dignum (2004), OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations, *Proceeding of the International Workshop on Programming Multiagent Systems Languages and Tools (PROMAS)*, International Workshop on Programming Multiagent Systems Languages and tools, 181-198.
18. M. Esteva (2003), 'Electronic Institutions: from specification to development', PhD Thesis, Technical University of Catalonia, Spain.
19. M. Esteva, D. d. l. Cruz and C. Sierra (2002), ISLANDER: An Electronic Institutions Editor, *Proceeding of the 1st International Joint Conference on Autonomous Agents and Multiagent systems (AAMAS '02)*, 1045-1052.
20. M. Esteva, J. A. Rodríguez-Aguilar, J. L. Arcos, C. Sierra and P. Garcia (2000), Institutionalising open multi-agent systems, *Proceeding of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, Boston, 381-383.
21. M. Esteva, J. A. Rodríguez-Aguilar, J. L. Arcos, C. Sierra and P. Garcia (2000), Formalising agent mediated Electronic Institutions, *Catalan Congres on AI (CCIA 00)*, 29-38.
22. M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia and J. L. Arcos (2001), "On the Formal Specifications of Electronic Institutions", *AgentLink*, Springer, 1991/2001, 126-147.
23. M. Esteva, B. Rosell, J. A. R. guez-Aguilar and J. L. Arcos (2004), AMELI: An Agent-Based Middleware for Electronic Institutions, *Proceeding of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, IEEE Computer Society, New York, NY, USA, 236-243.
24. J. Ferber and O. Gutknecht (1998), AALAADIN: A meta-model for the analysis and design of organizations in multi-agent systems, *Proceeding of the 3rd International Conference on MultiAgent Systems (ICMAS98)*, IEEE Computer Society, Paris, France, 128-135.
25. FIPA (1995). *Foundation for Intelligent Physical Agents (ACL)*, Available from: <http://www.fipa.org>, (Accessed on: September 2008).
26. N. Fornara and M. Colombetti (2007), Specifying and enforcing norms in artificial institutions, *Dagstuhl Seminar 07122, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI)*, Schloss Dagstuhl, Germany.

27. E. Friedman-Hill (2003), *Jess in Action: Rule-Based Systems in Java*, Manning.
28. E. Friedman-Hill (2003), "What are rule-based systems?" *Jess in Action: Rule-Based Systems in Java*, Manning, 13-27.
29. E. J. Friedman-Hill (2007). *Jess and XML*. Sandia National Laboratories, Available from: <http://herzberg.ca.sandia.gov/docs/70/xml.html>, (Accessed on: October 2007).
30. A. García-Camino, P. Noriega and J. A. Rodríguez-Aguilar (2005), Implementing norms in electronic institutions, *Proceeding of International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, ACM 2005, Utrecht, The Netherlands, 667-673.
31. A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra and W. W. Vasconcelos (2007), "Norm-oriented programming of electronic institutions: A rule-based approach", *Proceeding of the Coordination, Organization, Institutions and Norms in Agent Systems II*, vol. 4386/2007, Springer-Verlag, 177-193.
32. P. Giorgini, H. Mouratidis and N. Zannone (2006), "Modeling Security and Trust with Secure TROPOS", *Integrating Security and Software Engineering: Advances and Future Vision*, IDEA, 160-189.
33. D. Grossi, H. Aldewereld and F. Dignum (2006), Ubi lex Ibi poena: Designing norm enforcement in electronic institutions, *Pre-proceedings of the AAMAS06 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN@AAMAS'06)*, Hakodate, Japan.
34. D. Grossi, H. Aldewereld, J. Vázquez-Salceda and F. Dignum (2005), Ontological Aspects of the Implementation of Norms in Agent-Based Electronic Institutions, , *Proceeding of the 1st International Symposium on Normative Multiagent Systems (NorMAS2005)*, Springer Netherlands, Hatfield, England, 251-275.
35. E. I. Group (2008). *Electronic Institutions*. Artificial Intelligence Research Institute (IIIA) of the Spanish research council (CSIC), Available from: <http://e-institutions.iiia.csic.es>, (Accessed on: September 2008).
36. H. Herrestad and C. Krogh (1995), "Obligations directed from bearers to counterparts ", *Proceedings of the 5th international conference on Artificial intelligence and law*, Publisher ACM New York, NY, USA, 210 - 218.
37. W. Huang (2006), 'Agent-oriented software engineering : application to the management of community care provision', PhD Thesis, The University of Liverpool, UK.

38. N. R. Jennings, K. Sycara and M. Wooldridge (1998), A Roadmap of Agent Research and Development, *Autonomous Agents and Multi Agent Systems*, 1 (1), 7-38.
39. K. Binmore (1992), Fun and Games: a text on Game Theory, *D.C. Heath and Company, Lexington, MA*.
40. I. C. Kim (2006), "Dynamic Role Assignment for Multi-agent Cooperation ", *Computer and Information Sciences – ISCIS 2006*, vol. 4263/2006, Springer Berlin / Heidelberg, 221-229
41. D. Kinny, M. P. Georgeff and A. S. Rao (1996), A methodology and modelling technique for systems of BDI agents, *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 56-71.
42. H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda and M. Asada (1998), "The RoboCup synthetic agent challenge 97 ", *Book RoboCup-97: Robot Soccer World Cup I* vol. 1395/1998, Springer, Berlin / Heidelberg, 62-73.
43. KQML (1993). *Knowledge Query Manipulation Language*. Available from: www.cs.umbc.edu/kqml, (Accessed on: June 2008).
44. R. W. V. Kralingen, P. R. S. Visser, T. J. M. Bench-Capon, H. J. V. D. Herik and (1999), A principled approach to developing legal knowledge systems, *International Journal of Human-Computer Studies archive*, 51 (6), 1127-1154.
45. F. Lopez-y-Lopez and M. Luck (2004), "A model of normative multi-agent systems and dynamic relationships", *Regulated Agent-Based Social Systems*, G. Lindemann, D. Moldt and M. Paolucci (Editors), vol. 2934, Springer-Verlag, 259–280.
46. F. Lopez-y-Lopez, M. Luck and M. d'Inverno (2002), Constraining autonomy through norms, *Proceedings of International Joint Conference on Autonomous Agents and Multi Agent Systems AAMAS'02*, ACM Press, 674–681.
47. F. Lopez-y-Lopez, M. Luck and M. d'Inverno (2001), A framework for norm-based inter-agent dependence, *Proceedings of the 3rd Mexican International Conference on Computer Science*, 31-40.
48. F. Lopez-y-Lopez, M. Luck and M. d'Inverno (2004), Normative agent reasoning in dynamic societies, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems AAMAS'04*, ACM Press, 730–737.

49. F. Lopez-y-Lopez, M. Luck and M. d'Inverno (2005), A normative framework for agent based systems, *Computational & Mathematical Organization Theory*, 12 (2 - 3), 227-250.
50. M. Luck (2004). Slides of "*Methodologies and Modelling Languages*". Available from: <http://www.ecs.soton.ac.uk/~mml/absd/contents.html>, (Accessed on: April 2008).
51. M. Luck, R. Ashri and M. d'Inverno (2004), *Agent-Based Software Development*, Artech House Publishers.
52. D. Lyons (1969), Rights, claims and beneficiaries, *American Philosophical Quarterly*, 6, 173-185.
53. M. A. McMorran and J.E.Nicholls (1989), *Z User Manual*, IBM UK Laboratories.
54. A. Myatt (2007), *Pro NetBeans™ IDE 5.5 Enterprise Edition*, Apress Berkeley, CA, USA.
55. M. Nikraz, G. Caireb and P. A. Bahria (2006), A Methodology for the Analysis and Design of Multi-Agent Systems using JADE, *International Journal of Computer Systems Science and Engineering*.
56. P. Noriega and C. Sierra (2002), "Electronic institutions: future trends and challenges", *Cooperative Information Agents VI*, Springer, 2246, 14 - 17.
57. J. Odell, H. V. D. Parunak, S. Brueckner and J. A. Sauter (2003), Changing roles: Dynamic role assignment, *Journal of Object Technology*, 2 (5), 77-86
58. J. Odell, H. V. D. Parunak, S. Brueckner and J. A. Sauter (2003), Temporal aspects of dynamic role assignment, *Proceedings of the Conference of Agent Oriented Software Engineering (AOSE 2003)*, 201-213.
59. A. Omicini (2001), "SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems", *Agent Oriented Software Engineering*, New York Springer, 1957/2001, 185-193.
60. L. Padgham and M. Winikoff (2002), Prometheus: a methodology for developing intelligent agents, *Agent-Oriented Software Engineering III (AAMAS 2002)*, ACM, Bologna, Italy, 37-38.
61. I. Partsakoulakis and G. Vouros (2002), Importance and properties of roles in MAS organization: A review of methodologies and systems, *Proceeding of the workshop on MAS Problem Spaces and Their Implications to Achieving Globally Coherent Behavior*, Bologna, Italy Available from: <http://www.icsd.aegean.gr/incosys/Publications/ProblemSpaces02.pdf>.

62. I. Partsakoulakis and G. Vouros (2002), Importance and properties of roles in MAS organization: A review of methodologies and systems, *Proceeding of the workshop on MAS Problem Spaces and Their Implications to Achieving Globally Coherent Behavior*, Bologna, Italy.
63. I. Partsakoulakis and G. Vouros (2004), "Roles in MAS: Managing the Complexity of Tasks and Environments, in Multi-Agent Systems", *An Application Science for Multi-Agent Systems*, T. Wagner (Editor), vol. 10, Springer, 133-154.
64. I. Partsakoulakis and G. Vouros (2004), "Roles in MAS: Managing the Complexity of Tasks and Environments, in Multi-Agent Systems: An Application Science", T. Wagner (Editor), Kluwer Academic Publisher.
65. D. J. Pastor (2003), 'Automatic generation of E-organizations from institution specifications', MSc, Universitat Politècnica de Catalunya, Spain.
66. D. J. e. Pastor and J. Padget (2003), Towards HARMONIA: automatic generation of eorganisations from institution specifications, *Proceeding of the Workshop on Ontologies in Agent Systems (OAS 2003)*, Melbourne, Australia, 31-38.
67. E. I. Project (2002). *ISLANDER Editor*. Available from: <http://e-institutor.iiia.csic.es/islander/islander.html>, (Accessed on: September 2008).
68. G. Sartor (2006), Fundamental legal concepts: A formal and teleological characterisation, *Artificial Intelligence and Law*, 14 (1-2), 101-142.
69. J. Searle (1969), *Speech Acts*, Cambridge University Press.
70. C. Sierra, J. A. Rodriguez-Aguilar, P. Noriega, M. Esteva and J. L. Arcos (2004), Engineering Multi-agent Systems as Electronic Institutions, *European Journal for the Informatics Professional*, 4, 33-39.
71. V. T. d. Silva (2007), Implementing Norms that Govern Non-Dialogical Actions, *Dagsuhl Seminar Proceeding 07122*, Available from: <http://drops.dagstuhl.de/opus/volltexte/2007/927>.
72. M. Tambe, D. V. Pynadath and N. Chauvat (2000), Building Dynamic Agent Organizations in Cyberspace, *IEEE Internet Computing*, 65-73.
73. The (2006). *Fish Market Project*. Available from: <http://www.iiia.csic.es/Projects/fishmarket/newindex.html>, (Accessed on September 2008).
74. The (2006). *Project of Multi-Agent System for Fish Trading*. Available from: www.masfit.net, (Accessed on: November 2007).
75. G. Therborn (2002), Back to norms! on the scope and dynamics of norms and normative action, *Current Sociology*, 50, 863-880.

76. Timestamp (2008). Available from: <http://en.wikipedia.org/wiki/Timestamp>, (Accessed on: April 2008).
77. UML (2008). *Modelling Learning Trail*. NetBeans Community, Available from: <http://www.netbeans.org/kb/trails/uml.html>, (Accessed on: June 2008).
78. J. Vázquez-Salceda, H. Aldewereld and F. Dignum (2004), "Implementing norms in multiagent systems", *Multiagent System Technologies*, 3187, Springer Verlag, Berlin / Heidelberg, 313-327.
79. H. Van Dyke Parunak and J. J. Odell (2002), Representing Social Structures in UML, *Lecture Notes in Computer Science* (2222), 1-16.
80. J. Vázquez-Salceda, H. Aldewereld and F. Dignum (2005), Norms in multiagent systems: from theory to practice, *International Journal of Computer Systems Science & Engineering CRL publishing*, 20, 225-236.
81. J. Vazquez-Salceda and F. Dignum (2003), "Modelling Electronic Organizations", In V. Marik, J. Muller and M. Pechoucek (eds.) *Multi-Agent Systems and Applications III*, LNAI 2691, Springer Verlag, Berlin, 2003, 584-593.
82. M. Wooldridge (2002), *An Introduction to Multiagent Systems*, John Wiley & Sons, Chichester, England.
83. M. Wooldridge, N. R. Jennings and D. Kinny (2000), The Gaia Methodology for Agent-Oriented Analysis and Design, *Proceeding of the conference of Autonomous Agents and Multi-Agent Systems*, 3, 285-312.

Glossary

Action: An action is what an agent does, which is the ability of the agent to affect its environment. Abstractly, an agent receives information about events and their effects from its environment, then by some means, it selects an action to perform and finally it performs the action. (See Section 2.2.1)

Addressee: The addressee of the norm is the norm's subject that can be specified by the norm for an individual, an agent, the public or the system. In the other words, the addressee is the agent or person who does the act described in the norm. (See Section 3.3.1)

Beneficiary: The beneficiary of a norm is someone who benefits from the action specified in the norm. The beneficiary of the norm is as important as the addressee of the norm. (See Section 3.3.1)

Check Norms: Check norms specify the operationalization of the norms of the normative system for detection of any violation, compensation or reward cases. Different systems have different mechanisms to do these checks. Some of systems have random checks to detect violation, compensation or reward cases, while other systems check based on a regular schedule. (See Section 3.4.2)

Compensation: Compensation is a service that the normative system anticipates for the beneficiary of the norm when an obligation or prohibition is violated and the beneficiary of the norm thereby loses his/her rights. Such facilities might provide the whole right to the beneficiary, or a part of that, or any other form of compensation. (See Section 3.4.3)

Constitutive norms: Constitutive norms are non-regulative norms which have a classificatory or definitional character. For example, the rules of chess constitute the activities of the game. Such activities are dependent on these norms, as opposed to the regulative norms, where activities are independent from the norms. (See Section 3.2)

Descriptive Normative Language: This is a descriptive language which provides a precise definition of norms considering all norm elements and enforcement norm elements. (See Section 3.5)

Distributive Norms: Distributive norms or enforcement norms define how rewards, costs and punishments are assigned to the social system. The main contribution of this type of norm is in the enforcement of the norms; specifying the rewards for executing a legal action, the punishment after a violation, or the compensation, if applicable. (See Section 3.2 and Section 3.4)

Domain-Related Norms: These are norms which are specifically related to the domain of MAS intended to be normative. These norms specify all regulations of the MAS for external agents who join the system. These norms are defined by a normative system designer or a legislator of the system. (See Section 7.3.2.2)

Dynamic Assignment of R&Rs and sanctions to agents: In this PhD thesis, we proposed a solution for dynamic assignment of R&Rs to agents in which, taking account of the current role of the external agent, runtime occurrences, and the static normative knowledge base, R&Rs or sanctions of external agents are assigned at runtime to agents, by one of our proposed methods. (See Sections 4.3.2 and 4.3.3)

Dynamic Assignment of Roles to Agents: The method of dynamic assignment of roles to agents is defined as a systematic way in which, taking account of conditions of roles, the capabilities of agents and the overall context of actions, roles are assigned at runtime to agents, by a group organizer or a management system of the MAS. (See Section 4.3.1)

Dynamic Sources: We have defined all sources of changes and dynamism in multiagent systems as dynamic sources. The sources of changes includes actions, environmental events (comprises of actions of other agents, parameter changes and passage of time and network events). (See Section 4.2)

EIs: An abbreviation of Electronic Institutions. (See Section 9.2.1)

Enforcement Norm Elements: The key elements of enforcement norms in our proposed methods includes enforcement mode (punishment, reward and compensation). (See Section 3.4.5)

Enforcement Norms: See *Distributive Norms*.

Enforcer: In MASs, enforcers are internal agents who detect and enforce the norms, subject to norms defined in the system obliging them to do so. This is similar to human organizations in which different roles, for example, police, and inspectors, have the duties of enforcing laws and regulations. (See Section 3.4.7)

ENR: The abbreviation of ENforcement Rules that has been used for numbering the norms of the Auction normative knowledge base. (See Section 10.5 and Section 10.6)

Event: An event is a significant occurrence or change in the agent's environment or internally within the agent itself, to which the agent should respond by some actions. (See Section 2.2.1)

External Agent: The external agents are agents who join the MAS to use its facilities. For example, Buyer is an external agent who joins an auction system to participate in an auction or auctions. (See Section 2.3)

General Architecture: In this PhD thesis, this is the generic implementation architecture which we proposed for the implementation of our methods. This architecture is generic because it can be applied for implementing a standalone middleware tool which can be connected to any MAS intended to apply our methods for dynamic assignment of R&Rs and sanctions to external agents. (See Section 6.5)

General Rules: A set of norms which we have defined in our general architecture as check norms to detect violation, reward or compensation cases.

In addition, some of these general rules have been defined for changing the status of norms (e.g. from Activated to Fulfilled). In our implementation, these rules are generic for every domain application and automatically added to the normative knowledge base. (See Section 7.3.2.3)

GR: The abbreviation of General Rules that has been used for numbering the general rules of the Auction normative knowledge base. For example, GR1 means the first general rule. (See Section 10.7)

Internal Agent: Internal agents work on behalf of the MAS, whereas external agents are agents that join the MAS to use its facilities. The internal roles can only be played by internal (or staff) agents on behalf of the MAS. (See Section 2.3)

Jess: Jess is a Java-based rule engine and its Java APIs can be simply used in Java applications as well. This rule base engine is used by a variety of users in many different application domains. (See Section 6.5.1)

Legal Modalities: Legal modality or deontic modality determines whether the norm is either an *obligation* (ought), a *prohibition* (not ought) or a *permission* (may). In addition to these legal modalities, we consider *right* as a separate legal modality. (See Section 3.3.1)

Method 1: The first method we proposed for dynamic assignment of R&Rs and Sanctions to external agents which is based on role hierarchies. (See Section 5.2.2)

Method 2: The second method we proposed for dynamic assignment of R&Rs and Sanctions to external agents which is based on conditional norms. (See Section 5.2.1)

Norm Elements: Regulative norms contain several key norm elements such as addressee, beneficiary, act, scope, time and condition. (See Section 3.3.1)

Norm Enforcement: The implementation mechanisms that a normative multiagent system considers for executing the norms; for example, if a violation occurs, if a norm enacts or if there is a claim for compensation, the system can detect and respond to these behaviors. (See Section 3.4)

Norm Regimentation: Norm Regimentation is the process of making non-compliance with norms technically impossible. Norm Regimentation is an obvious way in which the fulfilment of the norms of a normative MAS can be implemented by making the violation of the norms impossible, so that norm compliance is inevitable. Regimentation guarantees the fulfilment of the norms in a multiagent system. (See Section 3.4.6)

Normative Knowledge Base: The knowledge base contains all norms and regulations of the multiagent system which external agents supposed to follow them. This knowledge base is based on the descriptive normative language and is defined by the normative system designer or legislator of the system. (See Section 5.5.10)

Normative Multiagent Systems: In our work, a normative multiagent system is a multiagent system together with normative systems in which agents can decide whether to follow the explicitly represented norms. Note that in

comparison with Boella's definition, we have not considered the case where agents can modify the norms in runtime. (See Section 2.4)

Obligation: An obligation is an action which should be performed by the addressee. So one can say the addressee "*is obliged to*", "*ought*", "*must*", "*has the duty to*" or "*is responsible to*" do an action. If the action is not performed, the addressee may be subject to some punishment or forfeit some rights. (See Section 3.3.1)

Permission: A permission is an action that addressee is allowed to do. In fact, the addressee "*is permitted*", "*is allowed*" to do the action and allowed not to do an action. (See Section 3.3.1)

Prohibition: A prohibition is an action which according to the law, should not be done by the addressee. In this case, one can say the addressee "*should not do*" an action or the action "*is banned*"/"*is forbidden*" for the addressee. Like an obligation, the addressee may be subject to some punishment or sanction if the norm is violated. (See Section 3.3.1)

Protocol-based Norms: Protocol-based norms are related to all the necessary conventions for agent interactions. This type of norm establishes the permitted actions at each instant of time, considering the past actions of agents. These protocols are statically designed at design time. This fact means that the system designer defines all norms or regulations of agents in the format of protocols at design time. (See Section 4.4)

Punishment: Punishments are actions to punish the violator when a norm violation occurs. In other words, after detecting the norm violation, punishment

norms define what the system responses to the violation are. (See Section 3.4.3)

R&Rs: The abbreviation of Rights and Responsibilities in this context.

Reaction Norms: Reaction norms define a plan of action to respond to the actions of agents relevant to norms, in order to complete norm enforcement after the detection stage (using check norms). Such a plan would be a *punishment* when a violation occurs or a *reward* when a norm is retained or compensation when occurrence of a violation causes some loss of rights of a beneficiary. (See Section 3.4.3)

Regulative Norms: Regulative norms help to regulate existing actions of agents. This type of action can be done ignoring the regulations as well as following them, but regulative norms are used to regulate actions which could be performed in any case. Regulative norms describe obligations, permissions and prohibitions. (See Section 3.3)

Rights and Responsibilities: In this PhD thesis, we denote all norms (including obligations, prohibitions, permissions and rights) which are defined for external agents as “Rights and Responsibilities”. (See Section 2.3 and Chapter 2)

Reward: Rewards are services which a normative system may provide for agents whose acts align with regulations, to encourage agents comply with the law. Rewards are supplied when the norms are executed and no violation of such norms has occurred. They are often used for prompt compliance. In some cases, rewards are for encouragement of people for undertaking a permitted action. (See Section 3.4.3)

Right: When a norm has a beneficiary, with a right, if the norm does not follow, the beneficiary will lose something and s/he can complain to some agent in authority for compensation as a result. (See Section 3.3)

Rule-based Norms: Rule-based norms are defined by a certain type of first-order formulae that set up a dependency relation between actions. These norms specify that under certain conditions, new commitments will be produced for agents to do some actions. Rule-based norms are applied to autonomous agents who can decide whether to follow or violate the norms; therefore these norms should be executed at runtime because autonomous behaviors are very difficult to anticipate at design time; in opposite to protocol-based norms. (See Section 4.4)

Runtime Occurrence: In this PhD thesis, we refer to all dynamic events that happen during runtime as runtime occurrences. See *dynamic sources*.

Sanction: In this PhD thesis, we use the term of sanction for all case of punishments, rewards and compensations applied to external agents in a normative MAS. (See Section 3.4.3)

VRn: This is abbreviation of Violation detection Rules used for numbering some of the general rules of the Auction normative knowledge base. For example VR1 means the first violation detection rule. (See Section 10.7)

Chapter 10
Appendices

10.1 Appendix A: The table of evaluation of role properties

Property	MaSE, Gaia, AAIL	AALAADIN	Teamcore	Robocup	ROP	Panzarasa
Specification	Roles are not realized in the final system. They are specified as abstract behaviors and are used to define abstract agent types or classes	A role is viewed as an abstract representation of an agent function, service or identification	A role is an abstract specification of a set of activities and inherits the requirements from each plan it has been assigned	A role is a specification of agent's position and interposition behavior	Set of activities for achieving a specific goal, associated with constraints (contextual, capability and mental state) and effects	A role is viewed as a set of mental attitudes. Attitudes are characterized, either as mandatory or optional
Assignment	Static. No deliberative assignment	Dynamic	Dynamic. Reactively based on agents' capabilities	Merely dynamic by changing positions in the field	Dynamic, including deliberation in the overall context of action	Dynamic, based on the agent's ability to achieve a particular state
Static vs. Dynamic	Static. Agents cannot decide which roles should be employed	Dynamic	Static. Agents cannot decide which roles should be employed	Merely dynamic by deciding which strategy to follow	Dynamic. Agents decide which roles should be employed	Dynamic. Agents can decide which roles should be employed
Cardinality	Many-to-Many. Many agents can play a role. Each agent has a copy of that role	Many-to-Many	Many-to-Many. Many agents are assigned to a group role and each agent may play more than one role	One-to-One	Many-to-Many. Many agents assigned to a role means either that each agent has a copy of that role or that the agents collectively execute that role	One-to-Many. One agent is assigned to many roles but one role is assigned only to one agent
Lifespan	Long-Lived	Transient (it can be long-lived)	Long-Lived	Transient (strategies may define different agent types)	Transient	Seems transient

Table 2-The evaluation of role properties in different methodologies and systems reproduced from [64]

10.2 Appendix B: Silva's Grammar

This is a part of the grammar of Descriptive Normative Language defined by Silva cited in [71] The whole grammar of this normative language is available at: <http://maude.sip.ucm.es/~viviane/grammar.grm>

```
<norm> ::= <deontic_concept> <norm_description>
<norm_description> ::= '{(' <action> ')}'
| '{(' <action> ')<sanction>}'
| '{(' <action> <temporal_situation> ')<sanction> }{'
| '{(' <action> 'IF' <if_condition> ')<sanction> }{'
| '{(' <action> 'IF' <if_condition> ')<sanction> }{' <action>
<temporal_situation>'IF'<if_condition>)'<sanction>}'
| '{(' <action> <temporal_situation> ')}'
| '{(' <action> 'IF' <if_condition> ')}'
| '{(' <action> <temporal_situation> 'IF' <if_condition> ')}'

<deontic_concept> ::= 'OBLIGED' | 'FORBIDDEN' | 'PERMISSION'
<sanction> ::= <punishments> <rewards>
| <punishments>
| <rewards>
<action> ::= <non_dialogical_action> | <dialogical_action>
<non_dialogical_action> ::= <entity> 'EXECUTE' <exec>
<entity> ::= <agent> ':' <role> | <role> | <agent> | <group> | 'ALL'

<temporal_situation> ::= BEFORE '(' <situation> ')'
| AFTER '(' <situation> ')'
| BETWEEN '(' <situation> ',' <situation>
')'
<if_condition> ::= <situation>
| 'NOT' <situation>
| <situation> <opl> <if_condition>
| 'NOT' <situation> <opl> <if_condition>
<situation> ::= <action>
| <action> <opl> <situation>
| <action> <temporal_situation>
| <expression>
| <expression> <opl> <situation>
| <time_expression>
| <time_expression> <opl> <situation>
| <numerical_expression> <op>
<numerical_expression>
| 'ACTIVATE' <norm>
| 'ACTIVATE' <norm> <opl> <situation>
| 'DEACTIVATE' <norm>
| 'DEACTIVATE' <norm> <opl> <situation>
| 'FULFILLED' <norm>
| 'FULFILLED' <norm> <opl> <situation>
| 'VIOLATED' <norm>
| 'VIOLATED' <norm> <opl> <situation>
```

Continued...

10.3 Appendix C: The Functions in Our Architecture

Functions of Our General Architecture: The following table shows the functions of the general architecture proposed in Section 6.5, and the entities and communications processes involved in executing each function.

Functions	Involved Resources	Involved entities	Involved Communication Processes
Norm Translation	<ul style="list-style-type: none"> • Static KB • Jess Rule-base 	<ul style="list-style-type: none"> • Norm Translator (E1) 	<ul style="list-style-type: none"> • D1 • D2
Event/Action Handling		<ul style="list-style-type: none"> • GUI • MAS Internal Agents • Event handler (E2) 	<ul style="list-style-type: none"> • C1
Event Recording	<ul style="list-style-type: none"> • Jess Fact Base 	<ul style="list-style-type: none"> • Event/action Handler(E2) • Time Holder(E8) • Enforcer (E9) • Event Recorder(E3) 	<ul style="list-style-type: none"> • C1 • C4.2 • C6.2 • D3, C2
Reasoning	<ul style="list-style-type: none"> • Jess Rule Base • Jess Fact Base 	<ul style="list-style-type: none"> • Event Recorder (E3) • Inference Engine 	<ul style="list-style-type: none"> • C2 • D6
Analyzing		<ul style="list-style-type: none"> • Inference Engine • Analyzer (E4) 	<ul style="list-style-type: none"> • D6 • C3, C4, C5, C6
Reporting R&R		<ul style="list-style-type: none"> • Analyzer (E4) • R&R Reporter (E5) 	<ul style="list-style-type: none"> • C3 • D7
Activating Norm	<ul style="list-style-type: none"> • Jess Fact Base 	<ul style="list-style-type: none"> • Analyzer (E4) • Activator (E6) 	<ul style="list-style-type: none"> • C4 • C4.1, D8
Deactivating Norm	<ul style="list-style-type: none"> • Jess Fact Base 	<ul style="list-style-type: none"> • Analyzer (E4) • Deactivator (E7) 	<ul style="list-style-type: none"> • C5 • C5.1, D9
Time Holding		<ul style="list-style-type: none"> • Activator (E6) • Deactivator (E7) • Time Holder (E8) 	<ul style="list-style-type: none"> • C4.1 • C.5.1 • C4.2
Enforcing Norm		<ul style="list-style-type: none"> • Analyzer (E4) • Enforcer (E9) 	<ul style="list-style-type: none"> • C6 • C6.1 • C6.2
Responding Queries	<ul style="list-style-type: none"> • Jess Fact Base 	<ul style="list-style-type: none"> • GUI • Query Responder 	<ul style="list-style-type: none"> • C7 • D10

10.4 Appendix D: Message Sequence Chart

Message Sequence Chart of Our General Architecture: The following charts are message sequence charts of the general architecture proposed in Section 6.5. Here we provide a listing of the order of execution of the processes, showing which entity and which process is involved at each step via a Message Sequence Chart.

There are two diagrams. The first one shows the process of norm translation and the second one shows the main process of dynamic assignment of R&Rs and sanctions to agents.

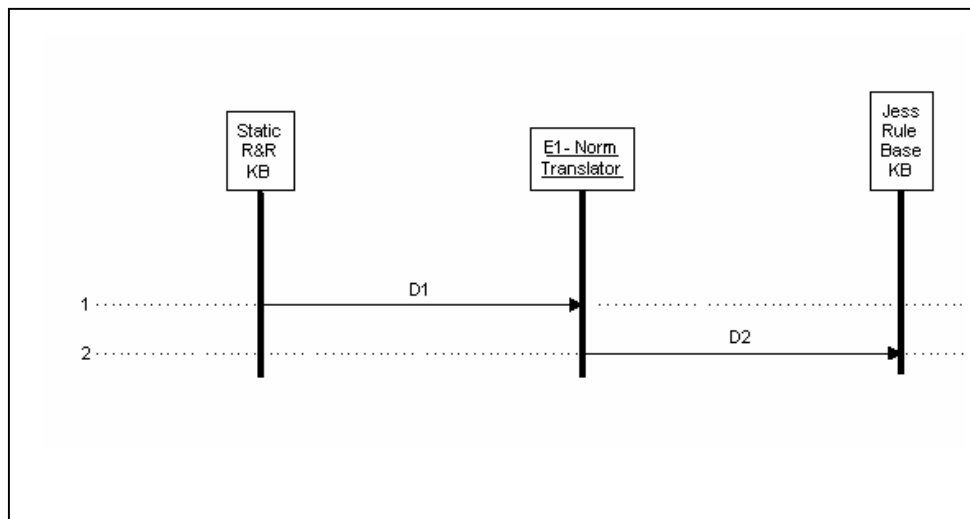


Figure 12- Message Sequence chart, norm translation

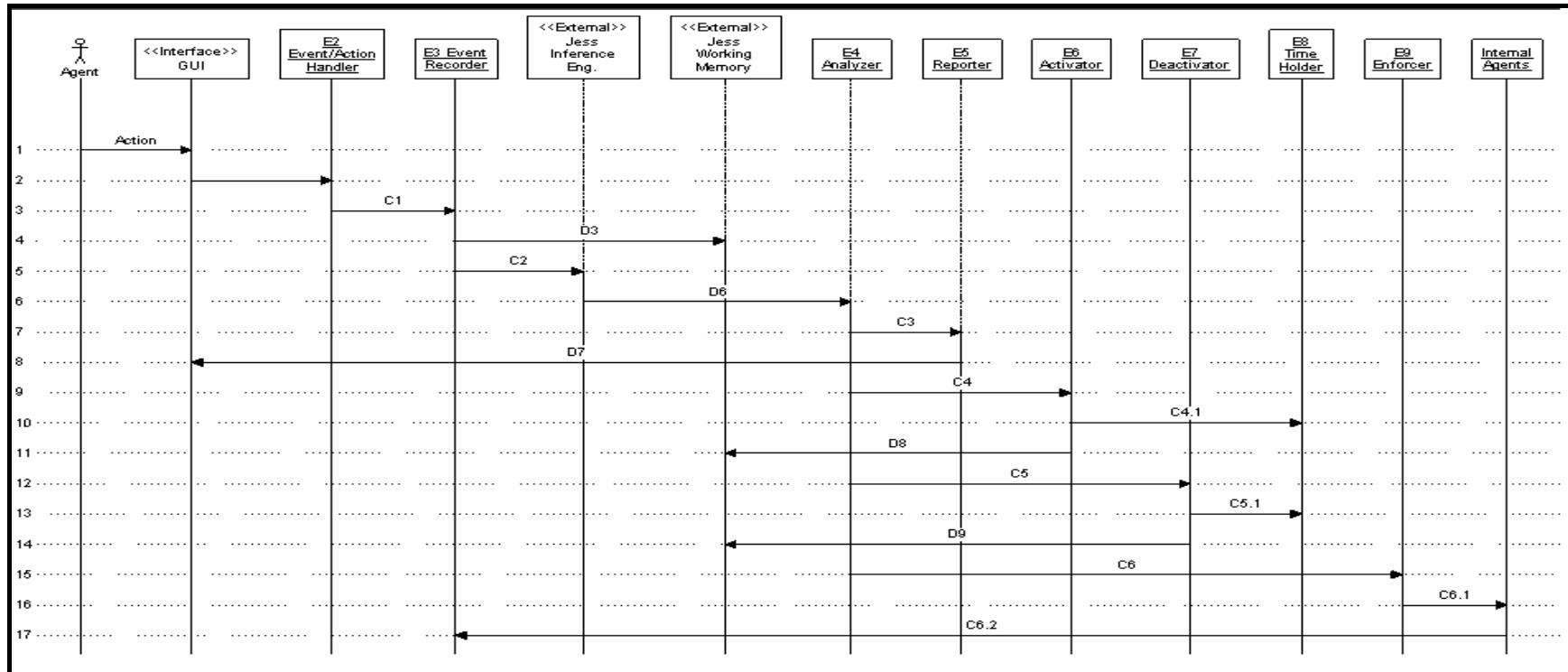


Figure 13-Message Sequence Diagram, Dynamic Assignment of R&Rs to Agents

10.5 Appendix E: Auction Normative KB for Method 1

The normative knowledge base of auction application for Method 1: In the following, we present an example of a domain-related part of the normative KB of Method 1. The domain of this knowledge base is an auction. Here, we define norms such that each norm has a number and a short description starting with “;”. The language of this knowledge base is Jess rule base. This example has been tested and used as an input for testing our application. The template of rule base is as follows:

(defrule ruleName “description” LHS =>RHS)

where LHS (Left-hand Side) contains the assignment of role to agent or an event and RHS (Right Hand Side) contains facts, rules or functions which is fired when the condition is satisfied.

These rules include both main norms and enforcement norms. The template we used for *norm* and *enforcementNorm* are as follows:

*(deftemplate norm (multislot status) (slot deoMode) (slot act) (slot addressee) (slot benef) (multislot object) (slot timeMode)
(multislot time))*

(deftemplate enforcementNorm (slot status)(slot addressee)(slot EnfCode))

```
.....  
..... AUCTION RULE BASE ..... For METHOD1 .....  
.....AUTHOR: FARNAZ DERAKHSHAN .....  
..... Template Defeinitions .....  
.....
```

```
(deftemplate AgentJoint (slot agentName)(slot AtTime))  
(deftemplate AgentLeft (slot agentName)(slot AtTime))  
(deftemplate role (slot roleTitle)(slot agentName)(slot AtTime))  
(deftemplate event (slot act)(slot actor)(slot forPerson)(multislot object)(slot AtTime))  
(deftemplate norm (multislot status) (slot deoMode) (slot act) (slot addressee)(slot benef) (multislot object) (slot timeMode) (multislot time))  
;in norm definition benef stands for beneficiary or benefactory  
(deftemplate currentTime (slot value))  
(deftemplate auctionStartTime (slot value))  
(deftemplate auctionValue (slot item)(slot auctionID)(slot price))  
(deftemplate enforcementNorm (slot status)(slot addressee)(slot EnfCode))  
(deftemplate feedback (slot actor)(slot value)(slot AtTime))
```



```

.....
..... AUCTION RULES ..... Using Method 1.....
..... AUTHOR FARNAZ DERAKHSHAN .....
.....FUNCTION DEFINITION;.....

```

;;;; based on our auction regulation the start time of the auction for advertised item is one hour after advertisement time $startTime=advertisementTime+1$ hours

```

(deffunction getStartTime (?x)(bind ?date (new java.util.Date))(bind ?longFrmt (call ?date parse ?x))(bind ?lf (+ ?longFrmt 3600000))(bind ?endDateObj (new
java.util.Date ?lf))(bind ?strFrmt (call ?endDateObj toGMTString))(return ?strFrmt))

```

;;;; based on our auction regulation ending time of the auction is one hour after starting time: $endTime =startTime(Ts)+1$ hr

```

(deffunction getEndTime(?x)(bind ?date (new java.util.Date))(bind ?longFrmt (call ?date parse ?x))(bind ?lf (+ ?longFrmt 3600000))(bind ?endDateObj (new
java.util.Date ?lf))(bind ?strFrmt (call ?endDateObj toGMTString))(return ?strFrmt))

```

;;;; based on our auction regulation payment due time of the auction is one day after ending time of the auction: $paymentDueTime =startTime(Ts)+1$ day

```

(deffunction getPaymentDueTime (?x)(bind ?date (new java.util.Date))(bind ?longFrmt (call ?date parse ?x))(bind ?lf (+ ?longFrmt 90000000))(bind ?endDateObj
(new java.util.Date ?lf))(bind ?strFrmt (call ?endDateObj toGMTString))(return ?strFrmt))

```

;;;; based on our auction regulation the deadline for sending the item to the buyer is 7 days after payment time of the item: $sendingDueTime(Tsd)=startTime+7$ days

```

(deffunction getSendingDueTime (?x)(bind ?date (new java.util.Date))(bind ?longFrmt (call ?date parse ?x))(bind ?lf (+ ?longFrmt 613800000))(bind ?endDateObj
(new java.util.Date ?lf))(bind ?strFrmt (call ?endDateObj toGMTString))(return ?strFrmt))

```

```

..... Auction RULE DEFINITION .....

```

;R1 : Seller has right to advertise an item.

```
(defrule advertiseItemPermission
```

```
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
```

```
=> (assert (norm (status ACTIVATED) (deoMode Prm) (act advertiseItem) (addressee ?x) ))
```

```
.....  
;R2 : Advertiser is forbidden to place Bid during the Auction Time (between Start Time and End Time).
```

```
(defrule placingBidForbidden
```

```
  ?roleFact <-( role (roleTitle advertiser)(agentName ?x)(AtTime ?time))
```

```
=>
```

```
  (assert (norm (status ToBeACTIVATED) (deoMode Frb) (act placeBid) (addressee ?x) (object ?item ?auction) (timeMode BETWEEN) (time ?startTime  
(getEndTime ?startTime)) ) )
```

```
.....  
;R3 : Advertiser is permitted to edit the auction before start time of the auction
```

```
(defrule editAuctionPermission
```

```
  ?roleFact <-( role (roleTitle advertiser)(agentName ?x)(AtTime ?time))
```

=> (assert (norm (status ACTIVATED) (deoMode Prm) (act editAuction) (addressee ?x) (object ?item ?auction) (timeMode BEFORE) (time ?startTime)))

.....

;R4 : Advertiser is forbidden to edit the auction after Start Time.

(defrule editAuctionForbidden

 ?roleFact <-(role (roleTitle advertiser)(agentName ?x)(AtTime ?time))

=>(assert (norm (status ToBeACTIVATED) (deoMode Frb) (act editAuction) (addressee ?x) (object ?item ?auction) (timeMode AFTER) (time ?startTime)))

.....

;R5 : Buyer is permitted to place a Bid between Start Time and the End Time.

(defrule placingBidPermission

 ?roleFact <-(role (roleTitle buyer)(agentName ?x)(AtTime ?time))

=> (assert (norm (status ToBeACTIVATED) (deoMode Prm) (act placeBid) (addressee ?x)(object ?item ?auction)(timeMode BETWEEN) (time ?startTime (getEndTime ?startTime)))))

.....

;R6 : The obligation for placing Higher bid is fulfilled for higherBidder.

(defrule placingHigherBidObligation

 ?roleFact <-(role (roleTitle bidder)(agentName ?x)(AtTime ?time))

```
=>(assert (norm (status ACTIVATED) (deoMode Obl) (act placeHigherBid) (addressee ?x)(object ?item ?auction)(timeMode BETWEEN)(time ?startTime (getEndTime ?startTime))))
```

.....

```
;R7 : The obligation for placing Higher bid is fulfilled for higherBidder.
```

```
(defrule placingHigherBidfulfilled
```

```
  ?roleFact <-( role (roleTitle higherBidder)(agentName ?x)(AtTime ?time))
```

```
=>(assert (norm (status FULFILLED) (deoMode Obl) (act placeHigherBid) (addressee ?x)(object ?item ?auction)(timeMode AtTime)(time ?time)) )
```

.....

```
;R8 : LowerBidder is violated the obligation of placing higher bid.
```

```
(defrule placingHigherBidViolated
```

```
  ?roleFact <-( role (roleTitle lowerBidder)(agentName ?x)(AtTime ?time))
```

```
=> (assert (norm (status VIOLATED) (deoMode Obl) (act placeHigherBid) (addressee ?x)(object ?item ?auction ) (timeMode AtTime)(time ?time)) ) )
```

.....

```
;R9,10 : LuckyAdvertiser has the right to receive the money from the Winner. Winner is obliged to pay the price of the item to the luckyAdvertiser.
```

```
(defrule receivePaymentRightAndPaymentObligation
```

```

?roleFact1 <-( role (roleTitle luckyAdvertiser)(agentName ?l))
?roleFact2 <-( role (roleTitle winner)(agentName ?w))
=>(assert (norm (status ACTIVATED) (deoMode Right) (act recievePayment) (addressee ?l)(benef ?w)(object ?item ?auction)(timeMode BEFORE) (time
(getPaymentDueTime ?startTime))) )
(assert (norm (status ACTIVATED) (deoMode Obl) (act pay) (addressee ?w)(benef ?l) (object ?item ?auction) (timeMode BEFORE) (time
(getPaymentDueTime ?startTime))) )

```

.....

;R11 : Payer has the right to get the item from the luckyAdvertiser.

```

(defrule getItemRight
?roleFact1 <-( role (roleTitle luckyAdvertiser)(agentName ?l))
?roleFact2 <-( role (roleTitle Payer)(agentName ?p)(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Right) (act getItem) (addressee ?p) (benef ?l)(object ?item ?auction ?bid) (timeMode BEFORE) (time
(getSendingDueTime ?startTime))) )

```

.....

;R12 : fastPayer has the right to get the item from the luckyAdvertiser.

```

(defrule getItemRight
?roleFact1 <-( role (roleTitle luckyAdvertiser)(agentName ?l))
?roleFact2 <-( role (roleTitle fastPayer)(agentName ?p)(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Right) (act getItem) (addressee ?p) (benef ?l)(object ?item ?auction ?bid) (timeMode BEFORE) (time
(getSendingDueTime ?startTime))) )

```

.....
;R13 : Payee is obliged to send the item to the payer.

(defrule sendItemObligation

 ?roleFact1 <-(role (roleTitle Payee)(agentName ?pe))

 ?roleFact2 <-(role (roleTitle Payer)(agentName ?pr)(AtTime ?time))

=> (assert (norm (status ACTIVATED) (deoMode Obl) (act sendItem) (addressee ?pe) (benef ?pr)(object ?item ?auction ?bid) (timeMode BEFORE) (time (getSendingDueTime ?startTime)))))

.....
;R14 : Payee is obliged to send the item to the fastPayer.

(defrule sendItemObligation

 ?roleFact1 <-(role (roleTitle Payee)(agentName ?pe))

 ?roleFact2 <-(role (roleTitle fastPayer)(agentName ?pr)(AtTime ?time))

=> (assert (norm (status ACTIVATED) (deoMode Obl) (act sendItem) (addressee ?pe) (benef ?pr)(object ?item ?auction ?bid) (timeMode BEFORE) (time (getSendingDueTime ?startTime)))))

.....ENFORCEMENT RULES (ENR);.....

;ENR1

(defrule wrongBidPunishment

 ?roleFact <-(role (roleTitle lowerBidder)(agentName ?x)(AtTime ?time))

```
=> (assert (norm (status VIOLATED)(deoMode Obl)(act placeHigherBid)(addressee ?x)(object ?item ?auction )(timeMode AtTime)(time ?time)))
      (assert (enforcementNorm (status PUNISHMENT) (addressee ?x) (EnfCode P1:toDecreaseFeedbackValue)))
      (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P1))) )
```

.....

;ENR2

```
(defrule rewardForFastPayment
  ?roleFact <-( role (roleTitle fastPayer)(agentName ?x)(AtTime ?time))
  =>
  (assert (enforcementNorm (status REWARD) (addressee ?x) (EnfCode R1:toIncreaseFeedbackValue)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode R1))) )
```

.....

;ENR3

```
(defrule rightToClaimForCompensation
  ?roleFact <-( role (roleTitle NonItemReciver)(agentName ?x)(AtTime ?time))
  =>
  (assert (norm (status ACTIVATED) (deoMode Right) (act claim)(addressee ?x)(object ?item ?auction)(timeMode AFTER) (time ?time)) )
```

.....

;ENR4

```
(defrule compensationForClaimantPayment
```

```

?roleFact <-( role (roleTitle claimantPayer)(agentName ?x)(AtTime ?time))
?normFact <-(norm (status ACTIVATED) (deoMode Right) (act claim))
=>
(assert (enforcementNorm (status COMPENSATION) (addressee ?x) (EnfCode C1:compensationClaimAccepted)))
(assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode C1)))
(assert (role (roleTitle AcceptedClaimantPayer)(agentName ?x)(AtTime ?time))) )

```

.....

;ENR5-A : If advertiser violates, he is a violatorSeller.

```

(defrule violatorSeller
  ?roleFact<-(role (roleTitle advertiser)(agentName ?x))
  ?normFact<- (norm (status VIOLATED)(deoMode Frb) (act placeBid) (addressee ?x) (object ?item ?auction )(timeMode AtTime)(time ?time))
  =>(assert(role (roleTitle violatorSeller)(agentName ?x)(AtTime ?time))))

```

.....

;ENR5-B

```

(defrule sellerBidderPunishment
  ?roleFact<-(role (roleTitle violatorSeller)(agentName ?x)(AtTime ?time))
  => (assert (enforcementNorm (status PUNISHMENT) (addressee ?x) (EnfCode P2:toDecreaseFeedbackValueOfSeller)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P2))) )

```

.....

;ENR6 : If agent has feedbacks=-3, agent is forbidden to join auction.

(defrule agentJointprohibition

 ?feedbackFact<-(feedback (actor ?x)(value -3)(AtTime ?time))

=>(assert(norm (status ACTIVATED) (deoMode Frb) (act auctionJoint) (addressee ?x)(timeMode AFTER)(time ?time)))

 (assert(role (roleTitle BarredMember)(agentName ?x)(AtTime ?time)))

 (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode barredMember))))

.....

10.6 Appendix F: Auction Normative KB for Method 2

The normative knowledge base of auction application for Method 2: In the following, we present an example of a domain-related part of the normative KB of Method 2. The domain of this knowledge base is an auction. Here, we define norms such that each norm has a number and a short description starting with “;” . The language of this knowledge base is Jess rule base. This example has been tested and used as an input for testing our application. The templates of rule base are the same as definitions of templates in Method 1. Therefore we do not repeat the definition of them.

In the definition of the rules, this normative KB uses the same templates and functions as the KB of Method 1 in Attachment E. So here we just put the main rules.

```
.....  
..... AUCTION RULE BASE ;.....; For METHOD1 ;.....  
.....;AUTHOR: FARNAZ DERAKHSHAN ;.....  
;...; Auction RULE DEFINITION ;.....  
;R1 :Seller has right to advertise an item.
```

(defrule advertiseItemPermission

```
?roleFact <-( role (roleTitle seller)(agentName ?x))
=> (assert (norm (status ACTIVATED) (deoMode Prm) (act advertiseItem) (addressee ?x)) )
```

.....
;R2 : If seller advertised an item, startTime and auctionValue is initialized.

```
(defrule advertiseItem
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?event<-(event (act advertiseItem)(actor ?x)(object ?item ?auction ?price)(AtTime ?time))
  =>(bind ?startTime (getStartTime ?time))(assert (auctionStartTime (value ?startTime)))
      (assert (auctionValue (item ?item)(auctionID ?auction)(price ?price))))
```

.....
;R3 : If seller advertised an item, Seller is not allowed to place Bid during the Auction Time (between Start Time and End Time).

```
(defrule placingBidForbidden
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?event<-(event (act advertiseItem)(actor ?x)(object ?item ?auction ?price)(AtTime ?time))
  =>(bind ?startTime (getStartTime ?time))(assert (auctionStartTime (value ?startTime)))
      (assert (norm (status ToBeACTIVATED) (deoMode Frb) (act placeBid) (addressee ?x) (object ?item ?auction) (timeMode
BETWEEN) (time (getStartTime ?time) (getEndTime ?startTime)) ) ) )
```

.....
R4 : If Seller advertise an item, Seller is permitted to edit the auction before start time of the auction.

```

(defrule editAuctionPermission
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act advertiseItem)(actor ?x)(object ?item ?auction ?price)(AtTime ?time))
  => (assert (norm (status ACTIVATED) (deoMode Prm) (act editAuction) (addressee ?x) (object ?item ?auction ?price) (timeMode
BEFORE) (time ?startTime) ) ) )

```

.....
;R5 : Seller is forbidden to edit the auction after Start Time.

```

(defrule editAuctionForbidden
  ?roleFact <-( role (roleTitle seller)(agentName ?x))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act advertiseItem)(actor ?x)(object ?item ?auction ?price)(AtTime ?time))
  => (assert (norm (status ToBeACTIVATED) (deoMode Frb) (act editAuction) (addressee ?x) (object ?item ?auction ?price)
(timeMode AFTER) (time ?startTime) ) ) )

```

.....
;R6A : Buyer is permitted to place a Bid between the Start Time and the End Time.

```

(defrule placingBidPermissionBeforeSTime
  ?roleFact <-( role (roleTitle buyer)(agentName ?x)(AtTime ?time))
  ?sFact<-(auctionStartTime (value ?startTime))
  (test (< ?time ?startTime))
  => (assert (norm (status ToBeACTIVATED) (deoMode Prm) (act placeBid) (addressee ?x)(timeMode BETWEEN) (time ?startTime
(getEndTime ?startTime) ) ) ) )

```

.....
;R6B : Buyer is permitted to place a Bid between the Start Time and the End Time.

```
(defrule placingBidPermission
  ?roleFact <-( role (roleTitle buyer)(agentName ?x)(AtTime ?time))
  ?sFact<-(auctionStartTime (value ?startTime))
  (test (> ?time ?startTime))
  => (assert (norm (status ACTIVATED) (deoMode Prm) (act placeBid) (addressee ?x)(timeMode BETWEEN) (time ?startTime
(getEndTime ?startTime)))) )
```

.....
;R7: If Buyer placed a bid, s/he is obliged to place higher bid.

```
(defrule placingHigherBidObligation
  ?roleFact<-( role (roleTitle buyer)(agentName ?x)(AtTime ?time))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?normFact<-(norm (status ACTIVATED)(deoMode Prm)(act placeBid)(addressee ?x))
  ?event <-(event (act placeBid)(actor ?x)(object ?item ?auction )(AtTime ?time))
  => (assert (norm (status ACTIVATED) (deoMode Obl) (act placeHigherBid) (addressee ?x)(object ?item ?auction )(timeMode
BETWEEN) (time ?time (getEndTime ?startTime)))) )
```

.....
;R8 :If Buyer place a Bid, and the bid is a higher bid, the act of placehigherBidder is fulfilled.

```
(defrule placingBid
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))
  ?event <-(event (act placeHigherBid)(actor ?x)(object ?item ?auction )(AtTime ?time))
=>(assert (norm (status FULFILLED) (deoMode Obl) (act placeHigherBid) (addressee ?x)
              (object ?item ?auction ?bid)(timeMode AtTime)(time ?time)) ) )
```

.....
;R9 :If Buyer place a Lower Bid, Buyer has a punishment :decrease the feedback number.

```
(defrule placingHigherBidObligation
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))
  ?event <-(event (act placeLowerBid)(actor ?x)(object ?item ?auction ?bid )(AtTime ?time))
=> (assert (norm (status VIOLATED) (deoMode Obl) (act placeHigherBid) (addressee ?x)(object ?item ?auction ?bid )(timeMode
AtTime)(time ?time)) ) )
```

.....
;R10,R11 : If buyer wins the auction Seller has the right to receive the money from the Buyer.

```
(defrule receivePaymentRightAndPaymentObligation
  ?roleFact1 <-( role (roleTitle seller)(agentName ?s))
  ?roleFact2 <-( role (roleTitle buyer)(agentName ?b))
  ?sFact<-(auctionStartTime (value ?startTime))
  ?event <-(event (act win)(actor ?b)(object ?item ?auction ?price)(AtTime ?time))
=> (assert (norm (status ACTIVATED) (deoMode Right) (act recievePayment) (addressee ?s)(benef ?b)(object ?item ?auction
?price)(timeMode BEFORE) (time (getPaymentDueTime ?startTime)))) )
```

```
(assert (norm (status ACTIVATED) (deoMode Obl) (act pay) (addressee ?b)(benef ?s) (object ?item ?auction ?price) (timeMode BEFORE) (time (getPaymentDueTime ?startTime)) ) )
```

```
.....  
;R12 : If buyer pays the price, buyer has the right to get the item.
```

```
(defrule getItemRight  
  ?roleFact1 <-( role (roleTitle seller)(agentName ?s))  
  ?roleFact2 <-( role (roleTitle buyer)(agentName ?b))  
  ?sFact<-(auctionStartTime (value ?startTime))  
  ?event <-(event (act pay)(actor ?b)(forPerson ?s)(object ?item ?auction ?price)(AtTime ?time))  
=> (assert (norm (status ACTIVATED) (deoMode Right) (act getItem) (addressee ?b) (benef ?s)(object ?item ?auction ?price) (timeMode BEFORE) (time (getSendingDueTime ?time)) ) )
```

```
.....  
;R13 : If buyer pays the price, seller is obliged to send the item.
```

```
(defrule sendItemObligation  
  ?roleFact1 <-( role (roleTitle seller)(agentName ?s))  
  ?roleFact2 <-( role (roleTitle buyer)(agentName ?b))  
  ?sFact<-(auctionStartTime (value ?startTime))  
  ?event <-(event (act pay)(actor ?b)(forPerson ?s)(object ?item ?auction ?price)(AtTime ?time))  
=> (assert (norm (status ACTIVATED) (deoMode Obl) (act sendItem) (addressee ?s) (benef ?b)(object ?item ?auction ?price) (timeMode BEFORE) (time (getSendingDueTime ?startTime)) ) )
```

```
.....
.....
.....ENFORCEMENT RULES.....
.....
;ENR1
```

```
(defrule wrongBidPunishment
  ?normFact<-(norm (status VIOLATED)(deoMode Obl)(act placeHigherBid)(addressee ?x))
=>(assert (enforcementNorm (status PUNISHMENT)(addressee ?x)(EnfCode P1:toDecreaseFeedbackValueOfBuyer)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P1))) )
.....
;ENR2; reward for the payments in 10 minutes of end time
```

```
(defrule rewardForFastPayment
  ?roleFact <-( role (roleTitle buyer)(agentName ?x))
  ?normFact<-(norm (status FULFILLED)(deoMode Right)(act pay)(addressee ?x)(timeMode AtTime)(time ?time))
  ?sFact<-(auctionStartTime (value ?startTime))
  (test(< ?time (getEndTime ?startTime)+600000))
=> (assert (enforcementNorm (status REWARD) (addressee ?x) (EnfCode R1:toIncreaseFeedbackValue)))
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode R1))) )
.....
;ENR3
```

```
(defrule rightToClaimForCompensation
  ?normFact<-(norm (status VIOLATED)(deoMode Right)(act getItem)(addressee ?x)(timeMode AtTime)(time ?time))
```



```
=> (assert (norm (status ACTIVATED) (deoMode Right) (act claim)(addressee ?x)(object ?item ?auction)(timeMode AFTER) (time ?time)) )
```

```
.....  
;ENR4
```

```
(defrule compensationForClaimantPayment
```

```
  ?event <-(event (act claim)(actor ?x)(object ?item ?auction)(AtTime ?time))
```

```
  ?normFact <-(norm (status ACTIVATED) (deoMode Right) (act claim)(addressee ?x)(object ?item ?auction))
```

```
  => (assert (enforcementNorm (status COMPENSATION) (addressee ?x) (EnfCode C1:compensationClaimAccepted)))
```

```
    (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode C1))))
```

```
.....  
;ENR5
```

```
(defrule sellerBidderPunishment
```

```
  ?roleFact<-(role (roleTitle seller)(agentName ?x))
```

```
  ?normFact<-(norm (status VIOLATED)(deoMode Frb) (act placeBid) (addressee ?x))
```

```
=> (assert (enforcementNorm (status PUNISHMENT) (addressee ?x) (EnfCode P2:toDecreaseFeedbackValueOfSeller)))
```

```
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode P2))))
```

```
.....  
;ER6 : If buyer has feedbacks=-3, Buyer is forbidden to join to the auction.
```

```
(defrule agentJointprohibition
```

```
  ?feedbackFact<-(feedback (actor ?x)(value -3)(AtTime ?time))
```

```
=>(assert(norm (status ACTIVATED) (deoMode Frb) (act auctionJoint) (addressee ?x)(timeMode AFTER)(time ?time)) )
```

```
  (assert (enforcementNorm (status EXECUTE) (addressee ?x) (EnfCode barredMember))) )
```

10.7 Appendix G: General Rules

General Rules for any normative knowledge base uses our method: Here we provide the definition of general rules which will be added by our tool to the rule base of the system. The related explanation of these norms is available in Section 7.3.2.3. These rules includes General Rules (GR) and Violation detection Rules (VR).

```
.....General Rule Definitions .....  
;GR1
```

```
(defrule statusChangeForObligationNorm  
?eventFact<-(event (act ?x)(actor ?y)(AtTime ?t))  
?normFact<-(norm (status ACTIVATED)(deoMode Obl)(act ?x)(addressee ?y))  
=>( duplicate ?normFact (status DEACTIVATED FULFILLED)(timeMode AtTime)(time ?t) (retract ?normFact))
```

```
.....  
;GR2
```

```
(defrule statusChangeForPermissionNorm  
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?t))  
?normFact<-(norm (status ACTIVATED)(deoMode Prm)(act ?x)(addressee ?y)(object ?z ?p ?q))  
=>( assert (status FULFILLED)(deoMode Prm)(act ?x)(addressee ?y)(object ?z ?p ?q) (timeMode AtTime)(time ?t) )
```

```

.....
;GR3
(defrule statusChangeForRightNorm
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?t))
?normFact<-(norm (status ACTIVATED)(deoMode Right)(act ?x)(addressee ?y)(object ?z ?p ?q))
=>( assert (norm (status FULFILLED)(deoMode Right)(act ?x)(addressee ?y)(object ?z ?p ?q) (timeMode AtTime)(time ?t)) ) (retract
?normFact ) )
.....

```

```

;GR4

(defrule statusChangeForForbidden
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ACTIVATED)(deoMode Frb)(timeMode BEFORE)(time ?t))
=> (duplicate ?normFact (status DEACTIVATED)(timeMode AtTime)(time ?t))(retract ?normFact))
.....

```

```

/*this rule covers all cases of tobeactivated norms including, all deontic modes (Obl, Frb, Prm and right)and after and between time
functions.*/

```

```

;GR5

(defrule statusChangeToBeActivatedAfter
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ToBeACTIVATED)(timeMode AFTER)(time ?t))

```

```
=>(duplicate ?normFact (status ACTIVATED))(retract ?normFact))
```

```
.....
```

```
;GR6  
(defrule statusChangeToBeActivatedBetween  
?CurTimeFact <-(currentTime (value ?t))  
?normFact<-(norm (status ToBeACTIVATED)(timeMode BETWEEN)(time ?t ?t2))  
=>(duplicate ?normFact (status ACTIVATED))(retract ?normFact))
```

```
.....
```

```
;GR7  
(defrule retractPermissionBefore  
?CurTimeFact <-(currentTime (value ?t))  
?normFact<-(norm (status ACTIVATED)(deoMode prm)(timeMode BEFORE)(time ?t))  
=>(duplicate ?normFact (status DEACTIVATED)(timeMode AtTime)(time ?t))(retract ?normFact))
```

```
.....
```

```
;GR8  
  
(defrule statusChangeForbiddenBetween  
?CurTimeFact <-(currentTime (value ?tx))
```

```
?normFact<-(norm (status ACTIVATED)(deoMode Frb)(timeMode BETWEEN)(time ?t ?tx))
=> (duplicate ?normFact (status DEACTIVATED)(timeMode AtTime)(time ?tx))(retract ?normFact))
```

.....

;GR9

```
(defrule statusChangePermissionBetween
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?tx))
?normFact<-(norm (status ACTIVATED)(deoMode Prm)(act ?x)(addressee ?y)(object ?z ?p ?q)(timeMode BETWEEN)(time ?t1 ?t2))
(test (> tx t1))
=>(duplicate ?normFact (status FULFILLED)(timeMode AtTime)(time ?tx))(retract ?normFact))
```

.....

;GR10

```
(defrule retractPermissionBetween
?CurTimeFact <-(currentTime (value ?tx))
?normFact<-(norm (status ACTIVATED)(deoMode Prm)(timeMode BETWEEN)(time ?t ?tx))
=> (duplicate ?normFact (status DEACTIVATED)(timeMode AtTime)(time ?tx))(retract ?normFact))
```

.....

;In the following, we provide the list of rules added by our tool to the rule base for detection of violation.

.....VIOLATION DETECTION;.....

;VR1

```
(defrule violatedObligationBefore
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ACTIVATED)(deoMode Obl)(timeMode BEFORE)(time ?t))
=>(duplicate ?normFact (status DEACTIVATED VIOLATED)(timeMode AtTime)(time ?t)(retract ?normFact))
```

.....

;VR2

```
(defrule violatedForbiddenBefore
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?tx))
?normFact<-(norm (status ACTIVATED)(deoMode Frb)(act ?x)(addressee ?y)(object ?z ?p ?q)(timeMode BEFORE)(time ?t))
(test (< tx t))
=>(duplicate ?normFact (status DEACTIVATED VIOLATED)(timeMode AtTime)(time ?t)(retract ?normFact))
```

.....

;VR3

```
(defrule violatedRightBefore
?CurTimeFact <-(currentTime (value ?t))
?normFact<-(norm (status ACTIVATED)(deoMode Right)(timeMode BEFORE)(time ?t))
```

=>(duplicate ?normFact (status DEACTIVATED VIOLATED)(timeMode AtTime)(time ?t))(retract ?normFact))

.....
;
;VR4

(defrule violatedForbiddenAfter
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?tx))
?normFact<-(norm (status ACTIVATED)(deoMode Frb)(act ?x)(addressee ?y)(object ?z ?p ?q)(timeMode AFTER)(time ?t))
(test (> tx t))
=>(duplicate ?normFact (status DEACTIVATED VIOLATED)(timeMode AtTime)(time ?t))(retract ?normFact))

.....
;VR5

(defrule violatedObligationBetween
?CurTimeFact <-(currentTime (value ?tx))
?normFact<-(norm (status ACTIVATED)(deoMode Obl)(timeMode BETWEEN)(time ?t ?tx))
=>(duplicate ?normFact (status DEACTIVATED VIOLATED)(timeMode AtTime)(time tx))(retract ?normFact))

.....
;VR6

(defrule violatedForbiddenBetween
?eventFact<-(event (act ?x)(actor ?y)(object ?z ?p ?q)(AtTime ?tx))

10.8 Appendix H: Tables for Extracting General Rules

The following tables shows that how we achieved the general rules of the normative knowledge base. We categorized norms, first based on temporal functions of Before, After and Between; then, based on deontic modes including obligation, prohibition (forbidden), permission and right. After that, for each case we wrote a general norm. Then we constituted check norms and reaction norms of these general norms for fulfilled and violated cases.

For example, the first row of the first table shows that:

If the norm is: “*If Cond. → ACTIVATED (Obl (B, BEFORE (t)))*”.

The second column says that: “*If (B, AT(Tx))*”, it means that *If the action B has been done at time Tx, the norm is fulfilled* and the following should be undertaken by our tool:

1. Jess retraction: *ACTIVATED (Obl (B, BEFORE (t)))*

2. Jess assertion: *FULFILLED (Obl (B, BEFORE (t))), AT(Tx)*

3. Tool task: Remove (t, TimerList)

These tables shows that how our tool can check the fulfilment and violation of norms.

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm	
			(FULFILLED)	(VIOLATED)
Before	Obl	<p>If Cond. →</p> <p><i>ACTIVATED</i> (Obl (B, BEFORE (t)))</p>	<p>If (B, AT(Tx)) →</p> <p>1. Jess retraction: <i>ACTIVATED</i> (Obl (B, BEFORE (t)))</p> <p>2. Jess assertion: <i>FULFILLED</i> (Obl (B, BEFORE (t))), AT(Tx))</p> <p>3. Tool task: Remove (t, TimerList)</p>	<p>If (curTime= t) & (<i>ACTIVATED</i> (Obl (B, BEFORE (t))))→</p> <p>1. Jess retraction: <i>ACTIVATED</i> (Obl (B, BEFORE (t)))</p> <p>2. Jess assertion: <i>VIOLATED</i> (Obl (B, BEFORE (t)))</p> <p>3. Jess assertion: EXECUTE(PUNISHMENT(Obl B))</p> <p>4. Tool Task GetResultsFromEnforcer()</p>
	Frb	<p>If Cond. →</p> <p><i>ACTIVATED</i> (Frb (B, BEFORE (t)))</p>	<p>If (curTime= t) & <i>ACTIVATED</i> (Frb (B, BEFORE (t))) →</p> <p>1. Jess retraction: <i>ACTIVATED</i> (Frb (B, BEFORE (t)))</p> <p>2. Jess assertion: <i>DEACTIVATED</i> (Frb (B, BEFORE (t)))</p> <p>1. Tool task: Remove (t, TimerList)</p>	<p>If (B, AT(Tx)) & <i>ACTIVATED</i> (Frb (B, BEFORE (t))) →</p> <p>1. Jess assertion: <i>VIOLATED</i> (Frb (B, BEFORE (t)))</p> <p>2. Jess assertion: EXECUTE(PUNISHMENT(Frb B))</p> <p>3. Tool Task GetResultsFromEnforcer()</p>

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm	
			(FULFILLED)	(VIOLATED)
Before	Prm	If Cond. → <i>ACTIVATED</i> (Prm (B, BEFORE (t)))	If (B, AT(Tx)) → 1. Jess assertion: <i>FULFILLED</i> (Prm (B, BEFORE (t))) If (curTime= t) & <i>ACTIVATED</i> (Prm (B, BEFORE (t))) → 1. Jess retraction: <i>ACTIVATED</i> (Prm (B, BEFORE (t))) 2. Tool task: Remove (t, TimerList)	No Violation Legislator may define a prohibition against the permission for doing this permitted action for the time after(t), otherwise permissions never violated
	Right ¹	If Cond. → <i>ACTIVATED</i> (Right (B, BEFORE (t)))	If (B, AT(Tx)) → 1. Jess retraction: <i>ACTIVATED</i> (Right (B, BEFORE (t))) 2. Jess assertion: <i>FULFILLED</i> (Right (B, BEFORE (t))) 3. Tool task: Remove (t, TimerList)	If (curTime= t) & (<i>ACTIVATED</i> (Right (B, BEFORE (t)))→ 1. Jess retraction: <i>ACTIVATED</i> (Right (B, BEFORE (t))) 2. Jess assertion: <i>VIOLATED</i> (Right(B, BEFORE (t))) 3. Jess assertion: EXECUTE(Compensation(Right B)) 4. Tool Task GetResultsFromEnforcer()

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm (FULFILLED) (VIOLATED)	
After	Obl	If Cond. → <i>ToBeACTIVATED</i> (Obl (B, AFTER (t)))	If (curTime= t) & (<i>ToBeACTIVATED</i> (Obl(B, AFTER (t)) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Obl(B, AFTER(t))) 2. Jess assertion: <i>ACTIVATED</i> (Obl(B, AFTER (t))) 3. Tool task: Remove (t, TimerList) If (B, AT(Tx))& <i>ACTIVATED</i> (Obl(B, AFTER (t)))→ 1. Jess retraction: <i>ACTIVATED</i> (Obl (B, AFTER(t))) 2. Jess assertion: <i>FULFILLED</i> (Obl(B, AFTER(t)), AT(Tx))	No Violation can be detected. Because is the addressee of the norm has not any limitation for doing the action on a specific time. This is a rare case, an Obligation for unlimited time. Such norm is applicable for the lifetime, and the punishment for violation would be for the other world.
	Frb	If Cond. → <i>ToBeACTIVATED</i> (Frb (B, AFTER(t)))	If (curTime= t) & <i>ToBeACTIVATED</i> (Frb (B, AFTER(t)) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Frb (B, AFTER(t))) 2. Jess assertion: <i>ACTIVATED</i> (Frb (B, AFTER(t))) 3. Tool task: Remove (t, TimerList)	If (B, AT(Tx)) & (<i>ACTIVATED</i> (Frb (B, AFTER(t))) → 1. Jess assertion: <i>VIOLATED</i> (Obl(B, AFTER(t)), AT(Tx)) 2. Jess assertion: EXECUTE(PUNISHMENT(Frb B))

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm (FULFILLED) (VIOLATED)	
After	Prm	If Cond. → <i>ToBeACTIVATED</i> (Prm (B, <i>AFTER</i> (t)))	If (curTime= t) & (<i>ToBeACTIVATED</i> (Prm (B, <i>AFTER</i> (t))) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Prm (B, <i>AFTER</i> (t))) 2. Jess assertion: <i>ACTIVATED</i> (Prm (B, <i>AFTER</i> (t))) 3. Tool task: Remove (t, TimerList) If (B, AT(Tx)) & (<i>ACTIVATED</i> (Prm (B, <i>AFTER</i> (t)))) → <i>FULFILLED</i> (Prm (B, <i>AFTER</i> (t)), At(Tx))	No Violation Legislator may define a prohibition against the permission for doing this permitted action for the time after(t), otherwise permissions never violated
	Right	If Cond. → <i>ToBeACTIVATED</i> (Right (B, <i>AFTER</i> (t)))	If (curTime= t) & <i>ToBeACTIVATED</i> (Right (B, <i>AFTER</i> (t))) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Right (B, <i>AFTER</i> (t))) 2. Jess assertion: <i>ACTIVATED</i> (Right (B, <i>AFTER</i> (t))) 1. Tool task: 3. Remove (t, TimerList) If (B, AT(Tx)) & (<i>ACTIVATED</i> (Right(B, <i>AFTER</i> (t)))) → 1. Jess assertion: <i>FULFILLED</i> (Right(B, <i>AFTER</i> (t)), At(Tx))	No Violation can be detected. Because the addressee of the norm has not any limitation for doing the action on a specific time.

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm (FULFILLED) (VIOLATED)	
Between	Obl	If Cond. → <i>ToBeACTIVATED</i> (Obl (B, <i>BETWEEN</i> (t1,t2)))	If (curTime= t1) & (<i>ToBeACTIVATED</i> (Obl(B, <i>BETWEEN</i> (t1))) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Obl(B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>ACTIVATED</i> (Obl(B, <i>BETWEEN</i> (t1,t2))) 3. Tool task: Remove (t1, TimerList) If (B, AT(Tx)) → 1. Jess retraction: <i>ACTIVATED</i> (Obl (B, <i>BEFORE</i> (t1,t2))) 2. Jess assertion: <i>FULFILLED</i> (Obl (B, <i>BEFORE</i> (t1,t2)), AT(Tx)) 3. Tool task: Remove (t2, TimerList)	If (curTime= t2) & (<i>ACTIVATED</i> (Obl (B, <i>BETWEEN</i> (t1,t2)))→ 1. Jess retraction: <i>ACTIVATED</i> (Obl (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>VIOLATED</i> (Obl (B, <i>BETWEEN</i> (t1,t2)) 3. Jess assertion: EXECUTE(PUNISHMENT(Obl B)) 4. Tool Task GetResultsFromEnforcer()

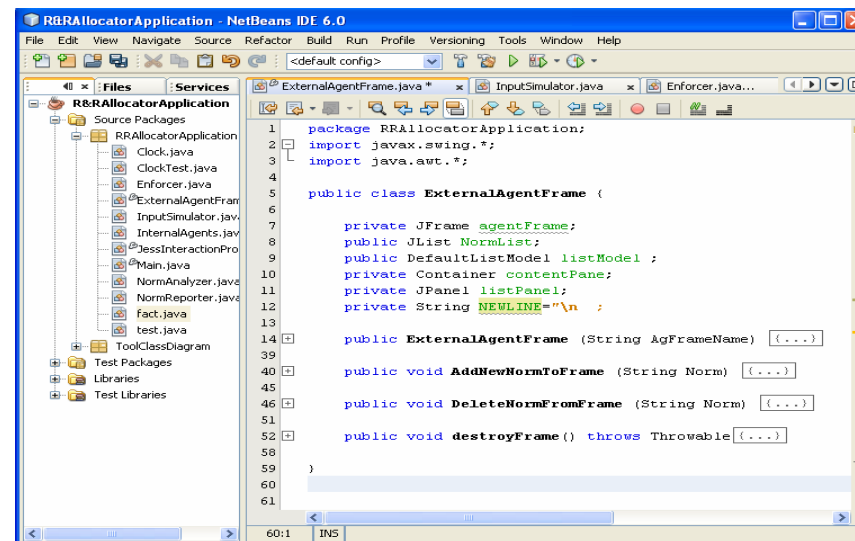
Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm (FULFILLED) (VIOLATED)	
Between	Frb	If Cond. → <i>ToBeACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2)))	If (curTime= t1) &(<i>ToBeACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>ACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) 3. Tool task: Remove (t1, TimerList) If (curTime= t2) &(<i>ACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) → 1. Jess retraction: <i>ACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>DEACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) 3. Tool task: Remove (t2, TimerList)	If (B, AT(Tx)) & <i>ACTIVATED</i> (Frb (B, <i>BETWEEN</i> (t1,t2))) → 1. Jess assertion: <i>VIOLATED</i> ((<i>ObI</i> (B, <i>BETWEEN</i> (t1,t2)), AT(Tx)) 2. Jess assertion: EXECUTE(PUNISHMENT(Frb B)) 3. Tool Task GetResultsFromEnforcer()

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm (FULFILLED) (VIOLATED)	
Between	Prm	If Cond. → <i>ToBeACTIVATED</i> (Prm (B, <i>BETWEEN</i> (t1,t2)))	If (curTime= t1) & (<i>ToBeACTIVATED</i> (Prm (B, <i>BETWEEN</i> (t1,t2)))) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Prm (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>ACTIVATED</i> (Prm (B, <i>BETWEEN</i> (t1,t2))) 3. Tool task: Remove (t1, TimerList) If (curTime= t2) & (<i>ACTIVATED</i> (Prm (B, <i>BETWEEN</i> (t1,t2)))) → 1. Jess retraction: <i>ACTIVATED</i> (Prm (B, <i>BETWEEN</i> (t1,t2))) 2. Tool task: Remove (t2, TimerList)	No Violation Legislator may define a prohibition against the permission for doing this permitted action for the time before (t1) or after(t2), otherwise permissions never violated

Time Notion	Deo Mode	Norm	Check Norm and Reaction Norm (FULFILLED) (VIOLATED)	
Between	Right	If Cond. → <i>ToBeACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2)))	If (curTime= t1) & (<i>ToBeACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2)))) → 1. Jess retraction: <i>ToBeACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>ACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2))) 3. Tool task: Remove (t1, TimerList) If (B, AT(Tx))& (<i>ACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2)))) → 1. Jess retraction: <i>ACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>FULFILLED</i> (Right (B, <i>BETWEEN</i> (t1,t2))) 3. Tool task: Remove (t2, TimerList)	If (curTime= t2) & <i>ACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2))) → 1. Jess retraction: <i>ACTIVATED</i> (Right (B, <i>BETWEEN</i> (t1,t2))) 2. Jess assertion: <i>VIOLATED</i> (Right(B, <i>BETWEEN</i> (t1,t2))) 3. Jess assertion: <i>EXECUTE</i> (Compensation(<i>Right</i> B)) 4. Tool Task GetResultsFromEnforcer()

10.9 Appendix I: The Java Source Code of Application

Here, we present some screenshots of the Java source code of the R&R Allocator Application which we have developed in Chapter 8.



```
1 package RRAllocatorApplication;
2 import javax.swing.*;
3 import java.awt.*;
4
5 public class ExternalAgentFrame (
6
7     private JFrame agentFrame;
8     public JList NormList;
9     public DefaultListModel listModel ;
10    private Container contentPane;
11    private JPanel listPanel;
12    private String NEWLINE="\n" ;
13
14    public ExternalAgentFrame (String AgFrameName) {...}
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40    public void AddNewNormToFrame (String Norm) {...}
41
42
43
44
45    public void DeleteNormFromFrame (String Norm) {...}
46
47
48
49
50
51    public void destroyFrame () throws Throwable {...}
52
53
54
55
56
57
58
59 )
60
61
```

Figure 14- ExternalAgentFrame creates frames for each external agent to show the assigned R&Rs and sanctions.

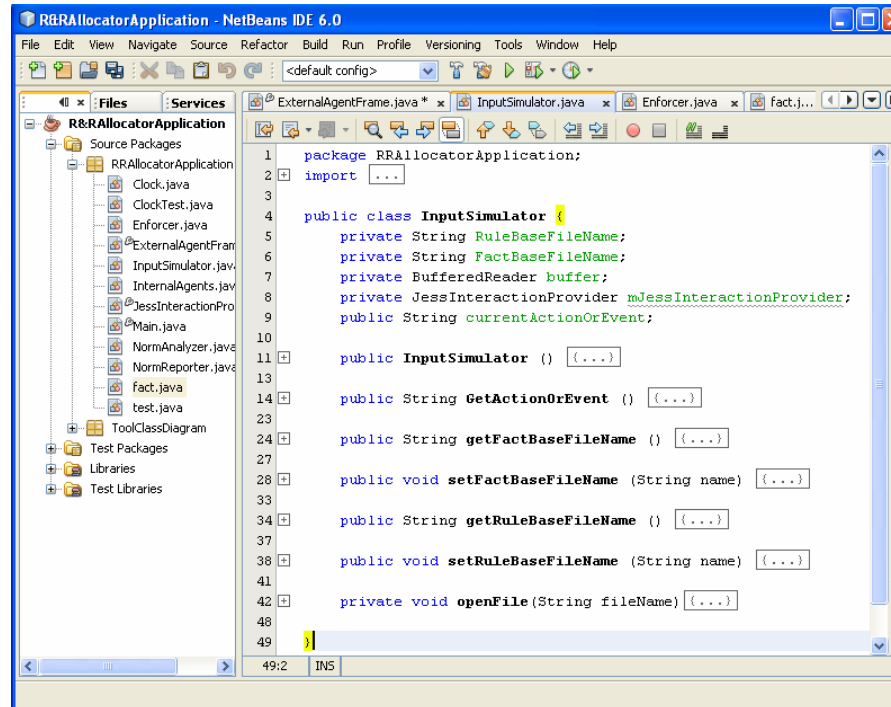


Figure 15- InputSimulator provides inputs of the application; it has been assumed that these inputs are actions and events coming from the MAS

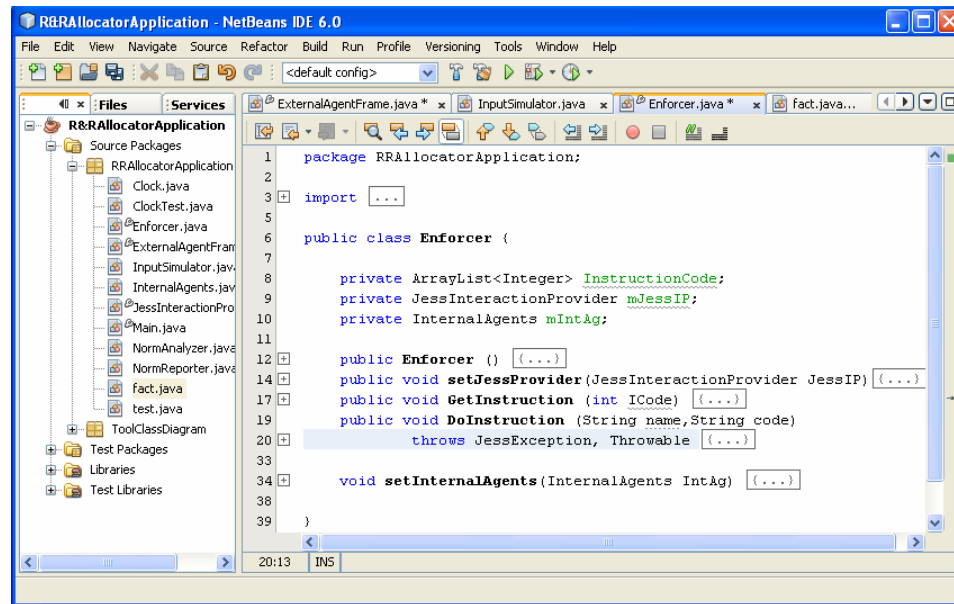


Figure 16- Enforcer executes the enforcement norms asking internal agents.

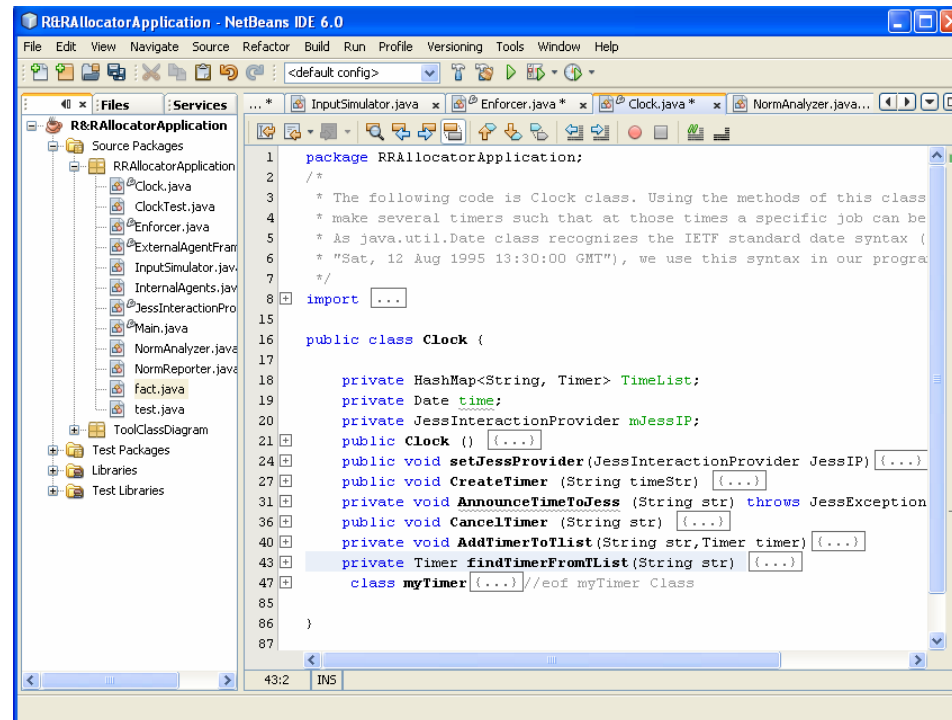


Figure 17-Clock creates timer, holds the time and announces the important times.

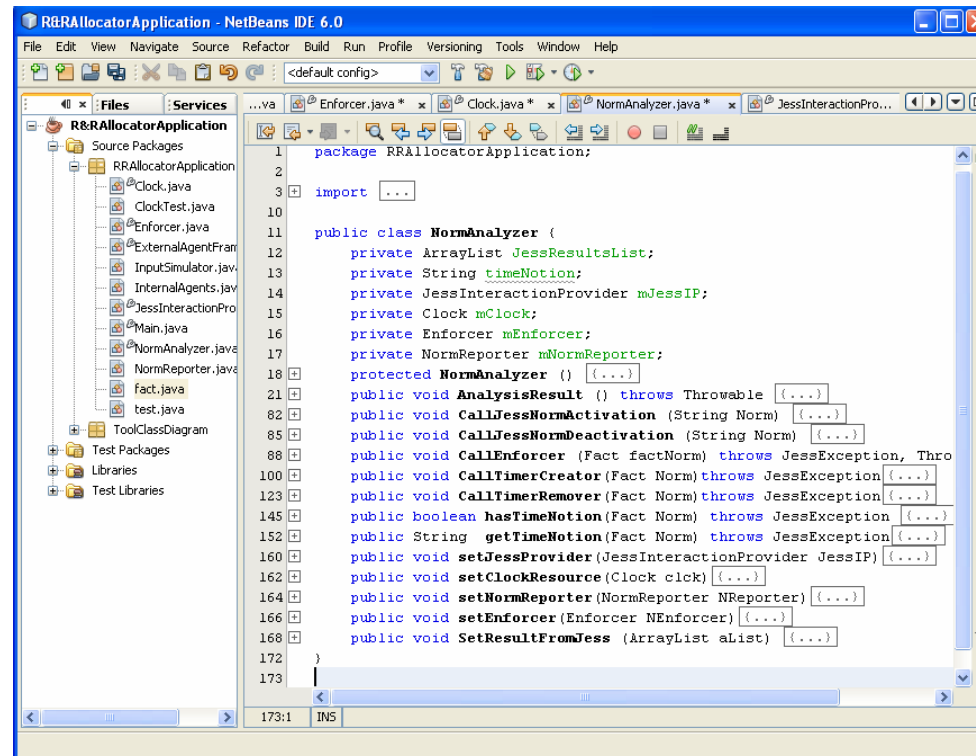


Figure 18- NormAnalyzer analyzes the result of every Jess reasoning; for example, it detects which norms has been recently activated or deactivated. Then, it sends the result of these analyses to the NormReporter.

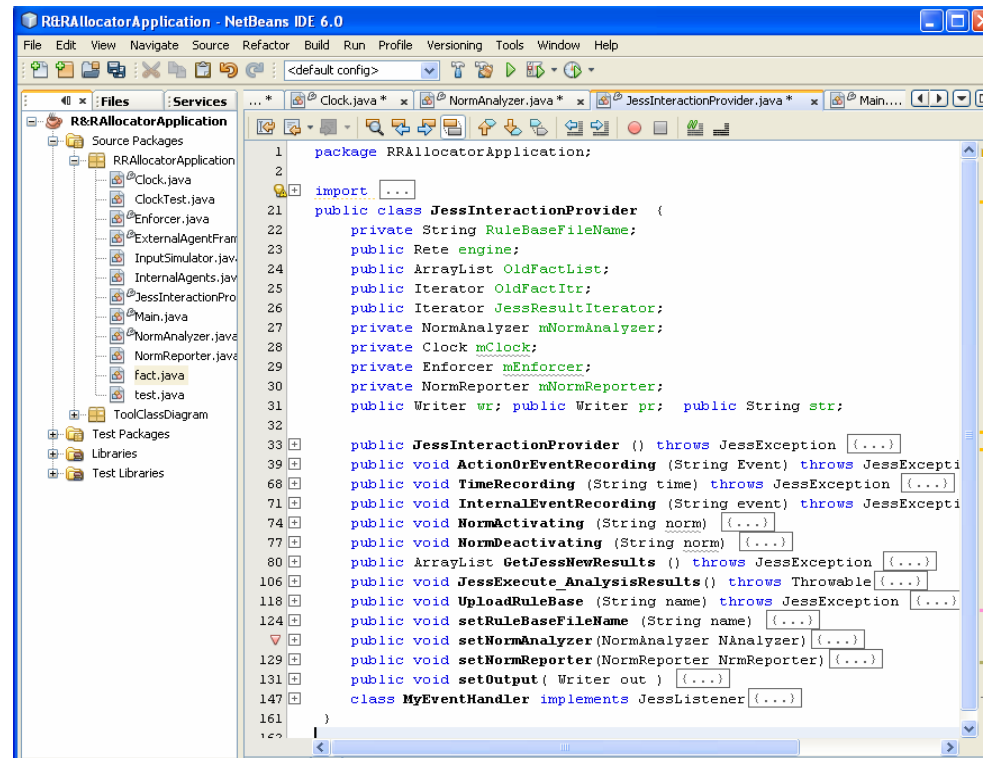


Figure 19-JessInteractionProvider provides Jess connection, Jess reasoning and collects the result of Jess reasoning. It also records actions, events and important time in the Jess fact-base.

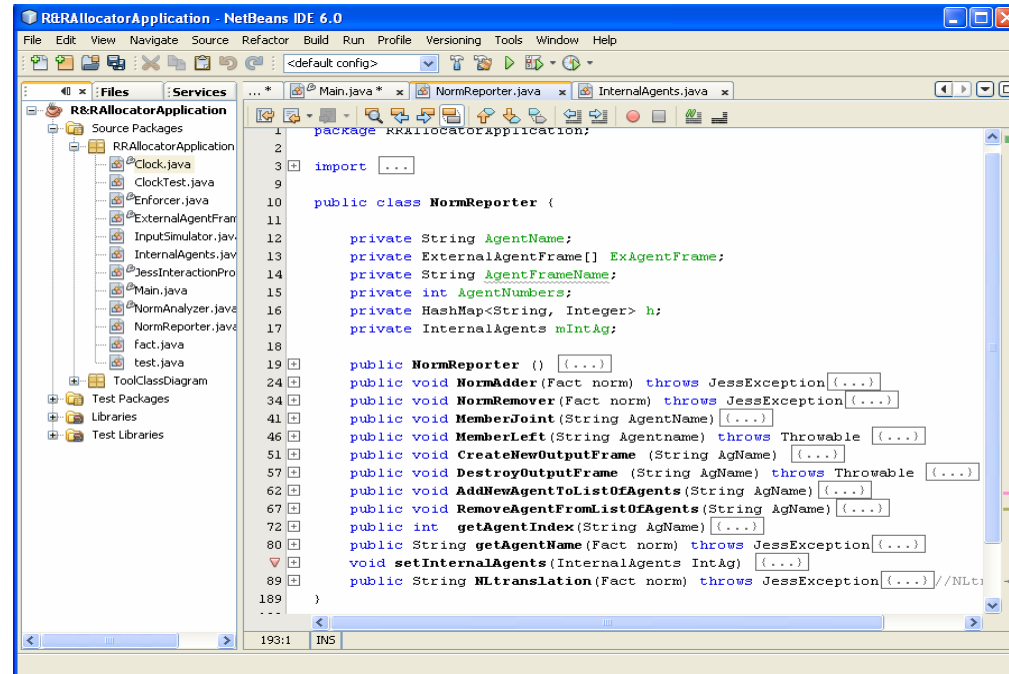


Figure 20-NormReporter gets the result of NormAnalyzer. This class translates these results to Natural Language using NLtranslation(). Then, it adds the new assigned norms to (or remove deactivated norms from) the frame of the relevant external agent. If a new agent joins to the system, NormReporter creates a frame for that external agent.


```
1 package RRAllocatorApplication;
2
3
4 import ...
5
6
7 /**
8  * @author Farnaz Derakhshan
9  * This is an application for allocating dynamic rights and responsibilities
10  * to external agents. To do so, this tool uses an input simulator for providing
11  * events as inputs.
12  */
13 public class Main {
14
15     public static void main(String[] args) throws JessException, IOException, Throwable {
16
17         InputSimulator IS=new InputSimulator(); //Creating an instance of input Simulator to pr
18
19         JessInteractionProvider JP=new JessInteractionProvider(); //Creating an instance of JessIntraction Provide
20
21         Clock clk=new Clock(); //Creating an instance of Clock
22         clk.setJessProvider(JP);
23         InternalAgents IntAg=new InternalAgents(); //Creating Internal Agent
24
25         Enforcer Enfrcr=new Enforcer(); //Creating an instance of Enforcer
26         Enfrcr.setJessProvider(JP);
27         Enfrcr.setInternalAgents(IntAg);
28
29         NormReporter mNormReporter=new NormReporter(); //Creating an instance of NormReporter
30         mNormReporter.setInternalAgents(IntAg);
31
32         Normalizer Nln=new Normalizer(); //Creating Norm analyzer class for analyzing Jess
33     }
34 }
```

Figure 21- This is a part of the main body of the application. Here, first we have created an instance of the predefined classes.

```
32 NormAnalyzer NA=new NormAnalyzer (); //Creating Norm Analyzer class for analysing Jess
33 NA.setJessProvider (JP);
34 NA.setClockResource (clk);
35 NA.setEnforcer (Enfrcr);
36 NA.setNormReporter (mNormReporter);
37
38 JP.setNormAnalyzer (NA) ;
39 JP.setNormReporter (mNormReporter);
40
41 //**** RULE BASE
42 //For testing Method1:
43 // IS.setRuleBaseFileName("D:\\Farnaz\\_My PracticalWork\\JessTest\\Method1\\RuleBaseMethod1.clp");
44 //For testing Method 2
45 IS.setRuleBaseFileName("D:\\Farnaz\\_My PracticalWork\\JessTest\\Method2\\GeneralRuleBase2.clp"); /*("D:\\
46 JP.UploadRuleBase (IS.getRuleBaseFileName ()); //Getting rule base file nameAllocating rule bas
47
48 //**** FACT BASE
49 //Fact base for testing Method 1
50 //IS.setFactBaseFileName("D:\\Farnaz\\_My PracticalWork\\JessTest\\Method1\\AuctionFactBaseMethod1-2.clp")
51 //Fact base for testing Method 2
52 IS.setFactBaseFileName("D:\\Farnaz\\_My PracticalWork\\JessTest\\Method2\\AuctionFactBaseMethod2.clp");///
53
54 while ( IS.GetActionOrEvent ()!=null){
55     String event=IS.currentActionOrEvent;
56     System.in.read ();
57     JP.ActionOrEventRecording (event); //Recording action or event in Jess as a new fact
58     JP.JessExecute_analysisResults ();
59
60 } //endof while
61 }
62 }
```

Figure 22-This figure shows the second part of the main body of the application. We set the Jess rule base (the normative knowledge base) and also the Jess fact base (which we made it for the auction scenario). Then, for occurrence of each event or action we get the output of our application, which is dynamic assignment of R&Rs and sanctions to external agents.